Vorlesung: Internettechnologien I

Thema: dynamische Webtechniken auf Serverseite

Gliederung:

- PHP
  - Motivation, Prinzip, Beispiel
  - Historie: PHP-Versionen im Überblick
  - Architektur, Konfiguration
  - Einbindung von PHP
  - Grundlagen (Ausgaben, Variablen, Datentypen, Kontrollstrukturen, Operatoren, Funktionen, Objekte)
  - Web-Formularanbindung
  - Datenbankanbindung
  - Cookies & Sessions
- Node.js als weitere serverseitige Technik

Warum PHP?

- Serverseitig eingesetzte Sciptsprache, die
  - leicht zu erlernen,
  - vielseitig einsetzbar,
  - Open Source,
  - für alle gängigen System verfügbar und
  - sehr gut dokumentiert ist.

Quelle: Wikipedia

Beispielskript: Hello World 1000

test.php

Die Datei ist strenggenommen keine HTML-Datei mehr, sondern eine PHP-Datei (Endung .php oder .phtml), damit dies der Server auch erkennt.

```
<html>
<head>
  <title>Testskript PHP</title>
</head>
<body>
<P>Der folgende Text wurden von PHP erzeugt:</P>
<?php
for ($i=1; $i<1000; $i++) {
  echo "Zeile " . $i . ": Hello World! <BR>";
}
?>
</body>
</html>
```

HTML-Code

PHP-Code

HTML-Code

PHP Historie

- Mitte der 90er Jahre: Vorläufer PHP/FI (=Personal Home Page / Forms Interpreter) durch Rasmus Lerdorf (Kanada); Set von Perl Skripten zur Erfassung der Zugriffe auf einen Webauftritt
- später mit **C-Sourcecode** als Toolsammlung neu implementiert und als Open-Source frei gegeben

PHP3 – Akronym für „PHP: Hypertext Preprocessor“ - ab Mitte 1998

- komplette Überarbeitung von PHP/FI durch eine Kooperation von Andi Gutmans, Zeev Suraski und Rasmus Lerdorf mit den Schwerpunkten: Erweiterbarkeit (-> interessant für vielen andere Entwickler -> PHP-Boom), Datenbankschnittstellen, Netz-Protokollzugriff und weitere APIs

PHP4 – Freigabe 2000

- viele neue Leistungsmerkmale in einer neuen Kernel-Engine – als 'Zend Engine bezeichnet (aus Zeev / Andi), stark verbesserte Performance und Unterstützung für viele weitere Webserver, HTTP-Sessions, Ausgabepufferung, sicherere Wege im Umgang mit Benutzereingaben, und verschiedene neue Sprachkonstrukte.

PHP5 – ab 2004

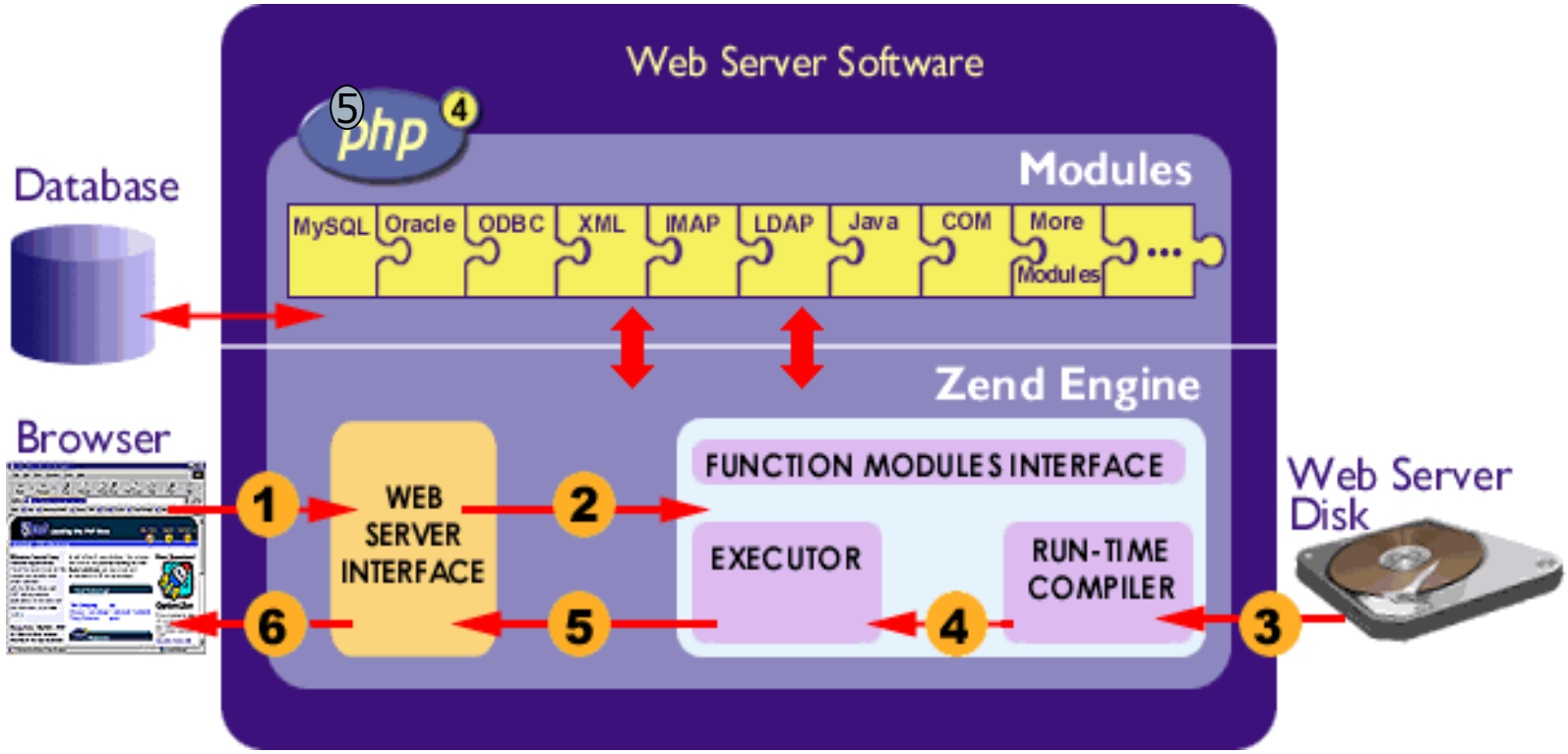
- nochmalige Leistungsverbesserungen in einer neuen ZEND-Engine 2.0, Detailänderungen bei der Einbindung verschiedener Bibliotheken

PHP7 – seit Dez. 2015

- Leistungsverbesserungen um ca. 30 % z.B. Durch Neuimpemetierung von Hashtabellen (gegenüber Konkurenz HipHop Virtual Machine 3.7 (HHVM) von Facebook

Die Software-Architektur von PHP

- PHP besteht aus
  - der Zend-Engine, die über einen Run-Time-Compiler (Interpiler) die Skripte ausführt und
  - einer Vielzahl von Modulen zum Zugriff auf externe Ressourcen.



Konfiguration von PHP (1)

- PHP wird global über die Konfigurationsdatei „php.ini“ konfiguriert. Wichtige Konfigurations-Direktiven:

Direktive	Bedeutung
`memory_limit = 8M` `max_execution_time = 30`	Festlegung der max. Ausführungszeit je Skript und der Speichernutzung

`display_errors = on`
`; log_errors = on`
`; error_Log = “C:\error.log“`
	Festlegung wie PHP mit Fehlern umgehen soll.

`SMTP=smtp.server.de` `sendmail_from = php@server.de`	Festlegung der Konfiguration zum automatischen Verschicken von eMails.
`safe_mode = on` `open_basedir = /pfad/zum/www-ordner`	Sicherheitseinstellungen
`short_open_tag = on; asp tags= off`	Spracheinstellungen: Einbindung in HTML
`extension=php_pdf.dll`	Laden von PHP-Erweiterungsmodulen

- Mittels der Funktion phpinfo() kann man sich aktuelle Konfiguration anzeigen lassen.
- Weitere Direktiven in http://www.php.net/manual/de/ini.list.php **[1]**

Konfiguration von PHP (2)

PHP-Einstellungen können als serverweite Einstellung **global über die Datei php.ini** vorgenommen werden, aber auch teilweise **pro Webserver Account (z.B. Apache – httpd.conf), pro Verzeichnis über eine .htaccess Datei** bzw. **innerhalb von Scripten**

- Nicht alle Einstellungen können überall vorgenommen werden (siehe **[1]**):

PHP_INI_USER	Der Eintrag kann in Skripten von Benutzern gesetzt werden
PHP_INI_PERDIR	Der Eintrag kann in *php.ini*, *.htaccess* oder *httpd.conf* gesetzt werden
PHP_INI_SYSTEM	Der Eintrag kann in *php.ini* oder *httpd.conf* gesetzt werden
PHP_INI_ALL	Der Eintrag kann überall gesetzt werden

Einstellungen via .htaccess Datei pro Verzeichnis vornehmen

- Wenn auf einem Server die Optionen "AllowOverride Options" oder "AllowOverride All" gesetzt sind, besteht die Möglichkeit PHP-Einstellungen über eine .htaccess Datei pro Verzeichnis zu steuern.
- **Beispiel**:

```
php_flag register_globals On
php_value display_errors 1
```

Konfiguration von PHP (3)

- **Einstellungen innerhalb von Scripten vornehmen**

  Beispiel:

```
<?php
    error_reporting(E_ALL);
    ini_set("display_errors", 1);
    include("file_with_errors.php");
?>
```

- Auch möglich, Wert von Direktive anzufragen:

```
if (!ini_get("display_errors")) {
    ini_set("display_errors", 1);
}
```

Einbindung von PHP in den HTML-Code (1)

- Kurzschreibweise

```
<?
   echo „Hallo, ich bin die erste Anweisung<BR>";
   echo „Und ich die zweite!“;
?>
```

- XML-konforme Schreibweise

```
<?php
   echo „Hallo, ich bin die erste Anweisung<BR>";
   echo „Und ich die zweite!“;
?>
```

- Lange Schreibweise

```
<script language="php"> echo „Hallo“ </script>
```

Einbindung von PHP in den HTML-Code (2)

- Beliebige PHP-Abschnitte innerhalb von HTML

```
<html><body>
 <?php  echo "Hallo Welt !";     ?>
</body> </html>
```

echo – gibt Argument an Output-stream

- PHP kann selbst temporär wieder zu HTML wechseln, ohne daß dies Bedingungen oder Ausdrücke stört

```
<?php  if ($expression) {?>
    <strong>Das ist richtig.</strong>
    <?php   } else  ?> {
    <strong>Das ist falsch.</strong>
    <?php }  ?>
```

- Komplette HTML-Ausgabe mit PHP

```
 <?php  echo "<strong>Das ist HTML</strong>" ?>
```

Sprachreferenz: Variablen

- Variablen werden durch $ am Anfang des Bezeichners markiert

```
$wert1 = 1;
```

- Es wird zwischen Groß- und Kleinschreibung unterschieden

```
$zaehler  !=  $Zaehler
```

- Datentypen und Typenumwandlung
  - Variablen müssen nicht deklariert werden.
  - Der Datentyp einer Variablen wird bei der Zuweisung eines Wertes automatisch ermittelt (ggf. automatische Typenumwandlung)
  - Interne Datentypen: integer, double, string, array, object

```
$wert1 = 1;        // Wert1 ist eine Integer-Variable
$wert2 = "Hallo"; // Wert2 ist eine String-Variable
```

- Variablen auf ihre Existenz testen bzw. löschen
  - Test mit: isset: `if(isset($a)) { echo "Es gibt a."}`
  - Variablen löschen: `unset($a); echo "Es gibt a nicht mehr.";`

Sprachreferenz: Datentypen

Verfügbare primitive Datentypen

- Boolean : `$flag = TRUE ; $Flag2 = FALSE;`
- Integer : `$i = -123; $z_hex = 0x1A;`

  (Integer-Überlauf führt automatisch zu float -> keine Gefahr !)
- Fließkomma-Zahl (float) : `$d1 = 1.234; $d2 = 1.2e-3;`
- String / Zeichenkette : können beliebig lang sein, 3 Arten der Def.:

```
$t1 = 'single quoted text'  //Keine Auswertg.
$t2 = "Das waren $anzahl werte !"; //double quoted-> PARSER
```

Heredoc-Syntax :

- frei gewählter Bezeichner – hier EOFT1 begrenzt den Text
- Bez. muß in 1. Spalte stehen !

```
$t3  = <<<EOFT1
Beispiel eines Strings über mehrere
Script-Zeilen durch Gebrauch
der heredoc-Syntax.
EOFT1;
```

Sprachreferenz: Datentypen und Operatoren

- Strings (string): Zeichenketten,
  z.B. `"Abc", "Er sagte: \"Ich gehe!\" und ging."`.
  Punkt-Operator (.) für Konkatenation : `$abc = „abc“ . „def“`
- Integer z. B. -1, 1000, 3, 7, 12, -1000
  z.B. `$i = 15; $j=5`
  arithmetische Operatoren (Kurzschreibweise erlaubt): `$i += $j ;`
  `$i -= $j; $i *= $j; $i /= $j; $i \%= $j;`
- Double oder real: z.B. -1.2, -100.001, 2.0e-12.
  Merke: . (Punkt) statt , (Komma) in PHP als Dezimaltrenner.
- Wahrheitswerte (bool): true oder false, bzw. 1 oder 0.

Überprüfung auf Datentyp:

```
is_bool($x),
is_int($x), is_integer($x), is_long($x),
is_double($x), is_float($x), is_real($x),
```

Beispiel: Datentypen und Operatoren

```php
<!-- Dateiname: plusOperator.php -->
<?php
$s1 = "ab";
$s2 = "cd";
$i = 5;
$j = "6";
$text = $s1 + $s2 . "<br/>"; // ergibt "0"
$text .= $s1 . $s2 . "<br/>"; // ergibt abcd"
$text .= $s1 + $i . "<br/>"; // ergibt "5"
$text .= $s1 . $i . "<br/>"; // ergibt ab5"
$text .= $j . $i . "<br/>"; // ergibt "65"
$text .= $j + $i . "<br/>"; // ergibt "11"
echo $text;
?>
```

Implizite Typumwandlung bei Anwendung der Operatoren, Achtung bei Plus-Operator!

Weitere Operatoren

- Vergleichsoperatoren

```
<!-- TRUE und FALSE; Dateiname: vergleichsOperatoren.php //-->
<?php
$s1="ac";
$s2="ad";
$i=5;
$j="5";

echo "$s1 != $s2: " . ($s1 != $s2 ) . "<br/>";// ac != ad: 1
echo "$s1 < $s2: " . ($s1 < $s2 ) . "<br/>";  // ac < ad: 1
echo "$s1 == $s2: " . ($s1 == $s2) . "<br/>"; // ac == ad:
echo "$s1 = $s2: " . ($s1 = $s2) . "<br/>";   // ac = ad: ad
echo "$i === $j: " . ($i === $j ) . "<br/>";  // 5 === 5:
echo "$j !== $i: " . ($j !== $i) . "<br/>";   // 5 !== 5: 1
echo "$i == $j: " . ($i == $j ) . "<br/>";    // 5 == 5: 1
echo "$j != $i: " . ($j != $i ) . "<br/>";    // 5 != 5:
echo "$j = $i: " . ($j = $i ) . "<br/>";      // 5 = 5: 5
echo "$i === $j: " . ($i === $j ) . "<br/>";  // 5 === 5: 1
?>
```

- Logische Operatoren
  - UND: `and`, ODER: `or`, NEGATION: `!`, XOR: `^`

Sprachreferenz: Arrays

Schlüssel => Wert - Paar
Schlüssel (Integer, String)
Wert (beliebiger Typ)

Komplexe Datentypen

Arrays : - sind Listen mit Zugriff über beliebige ID´s

```
$a1=array('Index0'=>1,1,1,8=>1,4=>1,19,3=>13,100=>'hallo','TEST');
$a1[2] = 25; echo "$a1[1] <br/>"; //Ausgabe:1
print_r($a1);//Ausgabe: Array([Index0]=>1 [1]=>1 [2]=>25 [8]=>1
             //[4]=>1 [9]=>19 [3]=>13 [100]=>hallo [101]=>TEST
```

Nützliche Funktionen:

- `unset($arr[5]);// Entfernt das Element aus dem Array`
- `explode, implode`: Konvertierung einer CSV-Datei/ Strings mit Trennzeichen in ein Array und zurück
- Sortierfunktionen: `sort`: je nach Angabe der sort_flags kann Sortierverhalten verändert werden (http://php.net/manual/de/function.sort.php)
- `json_encode($json, JSON_FORCE_OBJECT)` – Umwandlung eines Arrays ($json) in ein JSON - Objekt

Sprachreferenz: Klassen, Vererbung, Objekte

- Klassen und Objekte ist Java-ähnlich, aber:
  - Klassen dürfen nicht public sein,
  - Attribute/ Methoden haben Modifizierer, also **`private`**, **`protected`** oder **`public`**
  - Methoden und Konstruktoren sind Funktionen
  - Max. einen Konstruktor: `__construct(...)`
  - Referenzierung auf Attribute und Objektmethoden erfolgt mit Pfeiloperator ->
  - statische Attribute (Konstanten) oder Methoden (Klassenfunktionen) werden mit Doppel-Doppelpunktoperator :: referenziert

```
class Person{
 private $name;  //Deklaration
 public function __construct(){
   $this->name=$GLOBALS['firstname'];
 } }
// Instanziierung
 $p1  = new Person();
```

```
class Student extends Person{
 private $matrnr;
 public function __construct
                    ($matr){
   parent::__construct();
   $this->matrnr =$matr;
 }}      //Vererbung
```

Traits

- Benannte Gruppe von Methoden, Konstanten und Klassenvariablen
- Keine Instanziierung möglich, keine Subklassen
- Verwendet um Namensräume für statische Methoden und Konstanten zu erstellen oder Mixins zu programmieren (siehe Bsp.), die man mit `use` in eine Klasse „hineinmischen“ (mix in) kann

```php
<?php
trait Mathematik {
 public function mehrwertsteuer($brutto) {
 return 0.19/1.19 * $brutto;
}
}

class Rechnung {
 use Mathematik;
}

$rechnung = new Rechnung();
echo $rechnung->mehrwertsteuer(29.90);
?>
```

Sprachreferenz: Globale und lokale Variablen

- In PHP wird unterschieden zwischen
  - Globalen Variablen, die im gesamten Skript bekannt sind und
  - Lokalen Variablen, die nur in der aktuellen Funktion bekannt sind.

- Beispiel: Lokale Variable

```
function test_lokal($zahl) {
    $zweitezahl = 10;  //diese Variable ist nur in Fkt. bekannt
    return $zahl * $zweitezahl;
}
```

- Beispiel: Globale Variable

```
function test_global($zahl) {
    global $zweitezahl;   // Zugriff auf globale Variable
    return $zahl * $zweitezahl;
}

$zweitezahl = 10; // globale Variable definieren
echo test_global(10);
```

Sprachreferenz: Superglobale Variablen

SUPERGLOBALS sind automatische Variablen, auf die im gesamten Script zugegriffen werden kann. Als Schlüssel für den Zugriff auf die assoziativen Arrays dienen die Name der Variablen.

$GLOBALS : alle globalen Variablen – damit wird die Spezifikation: global $Varname unötig.
$_SERVER : Informationen betreffend Header, Script, Pfade
$_GET : Zugriff auf die Parameter einer GET-Anfrage z.B. `$_GET["name"]`
$_POST : Zugriff auf die Parameter einer POST-Anfrage z.B. `$_POST["name"]`

$_SESSION : beinhaltet alle Variablen einer aktuellen Session
Weitere Variablen s.
http://www.php.net/manual/en/language.variables.superglobals.php

Selbstdefinierte Konstanten: `define("FOO", "irgendwas");`
Magische Konstanten : `__LINE__` Die aktuelle Zeilennummer einer Datei.

Beispiele zu Variablen und Kommentaren

```php
<?php
    $arg = "foo";
    $val = "bar";

    //${$arg$val} = "invalid"; // Inval
    ${$arg . $val} = "working";
    echo $foobar;              // "working";
    /*echo $arg$val;           // Invalid
      echo ${$arg$val};        // Invalid */
    echo ${$arg . $val};       // "working"
?>
<?php
    $foo = 'Nach dem Mittag';
    $bar = &$foo; //$foo verweist auf $bar.
    $bar = "$bar …Zeit für ein Schläfchen!";
    echo $bar; // $bar's Inhalt?
    echo $foo; // $foo's Inhalt?.
?>
```

Variable Variablen: alle Token innerhalb geschweifter Klammern werden für die Formung des Variablennamens genutzt.

Einzeilige Kommentare und mehrzeilige Kommentare

werden nicht an den Browser weitergeleitet

Nur benannte Variablen können referenziert werden; keine arithm. Ausdrücke, auch keine Funktionsnamen

Sprachreferenz: Funktionen und Prozeduren

- Funktionen ohne Rückgabewert (Prozeduren)

```
function QuadratAusgabe($zahl) {
        $zahl = $zahl * $zahl;
        echo „Das Quadrat ist “ . $zahl;
}
```

- Funktionen mit Rückgabewert

```
function Mult($zahl, $zahl1 = 5) {
        $zahl = $zahl * $zahl1;
        return $zahl;
}
```

Angabe von Default-Werten für optionale Parameter immer am Ende der Liste

- Aufruf von Funktionen
  - Funktion ohne Rückgabewert:
  - Funktion mit Rückgabewert:

```
QuadratAusgabe(10);
$a = Mult(10);
```

- Variablen als Funktionsnamen

```
$test = 'Mult';
$test(10); // ruft Funktion Mult auf
```

Sprachreferenz: Funktionen und Prozeduren

- Einsatz des strengen Typisierungsmodus (ab PHP7)

  Funktionen mit Rückgabewert

```php
<?php
declare(strict_types=1);

function sum($a, $b): int {
    return $a + $b;
}

var_dump(sum(1, 2));
var_dump(sum(1, 2.5));
```

Ausgabe:

```
int(3)
Fatal error: Uncaught TypeError: Return value of sum() must be of the type integer, float returned in - on line 5 in -:5 Stack trace: #0 -(9): sum(1, 2.5) #1 {main} thrown in - on line 5
```

Sprachreferenz: Kontrollstrukturen (1)

- Bedingungen if ()
  Trinär-Operator:

$wert = $frage ? $wahr : $falsch;

```
if ($a > $b) {  print "a ist > b"; }
else      { print "a NICHT > b !";  }
```

- switch (wie in C)

```
switch ($i) {
case 0: echo "i ist 0"; break;
case 1: echo "i ist gleich 1";
        break;    // ...
}
```

- While (Bedingung) Anw.
  Do ... While (Bedingung)

```
$i = 1; while ($i <= 10) {
   echo $i++;  }
```

```
$i = 1; do { echo $i++;  } while ($i
<= 10)
```

- Zählschleife (analog C)

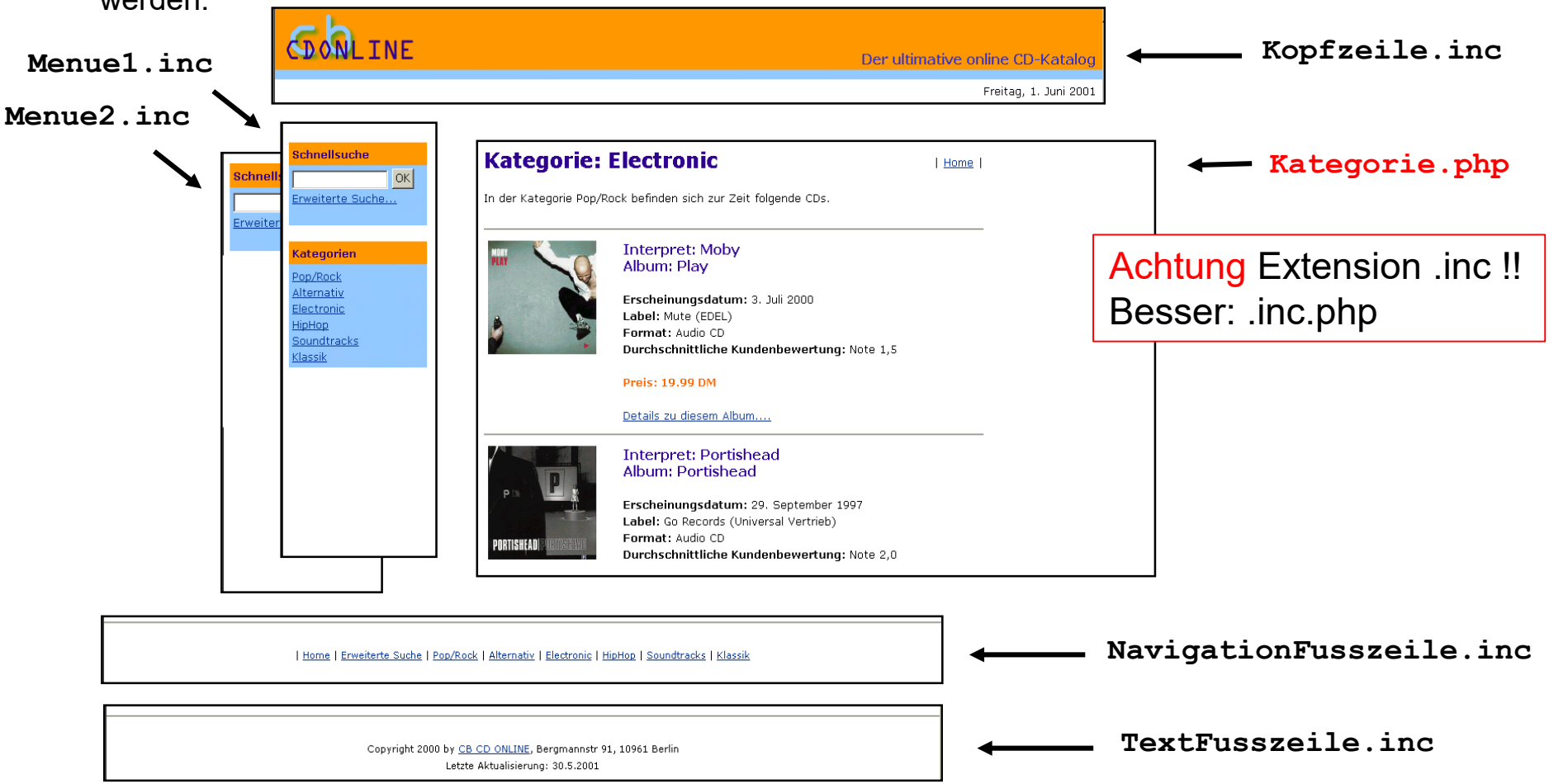
```
for ($i = 1; $i <= 10; $i++)
 { echo $i;  }
```

- For each – Schleife zum Durchlaufen von Arrays

```
$arr = array("eins","zwei","drei");
foreach ($arr as $value) {
    echo "Wert:  $value<br />\n";
```

Kontrollstrukturen (2): Seitengenerierung mittels Require/Include-Befehlen

Die Wartung von Webapplikationen lässt sich erleichtern, indem wiederkehrende Layout-Elemente zentral in eigenen Dateien gespeichert werden, die anschließend in eine Vielzahl von Seiten integriert werden.



Kontrollstrukturen (2): Seitengenerierung mittels Require-Befehlen

Die Layout-Elemente werden anschließend mittels Require-Befehlen wieder zusammengesetzt (siehe auch require_once und include/Include_once).

index.php

```
<?
// Variablen, die in den Includes verwendet werden festlegen
$Seitentitel = "CB CD-Shop - Kategorie: Electronic";

// Einbinden zentraler Funktionsbibliotheken (z.B. Formularprüffunktionen)
require(".\_includes\PhpScript\SeitenFunktionen.inc");

// Einbinden der Layoutbausteine Kopfzeile und Menue links
require(".\_includes\PageElements\Header.inc");
require(".\_includes\PageElements\KopfZeile.inc");
require(".\_includes\PageElements\MenuSucheKategorien.inc");

?>

       <h1>Kategorie: Electronic</h1>
        <p>Hier steht der eigentliche Inhalt der Seite.</p>

<?
// Einbinden der Fusszeilen
require(".\_includes\PageElements\NavigationFusszeile.inc");
require(".\_includes\PageElements\TextFusszeile.inc");

?>
```

Übergabe von Formular-Parameter an Skripte

Parameterübergabe zwischen Skripten

- Für viele Anwendungen ist es nötig, Skripten Parameter zu übergeben
  - z.B. Kategorienseite des CD-Shops: Welche Kategorie?
  - z.B. CD-Anzeigeseite: Welche CD soll angezeigt werden?
- Parameter werden an URLs angehängt

  `http://localhost/kategorie.php?name=Electronic`
- Wenn PHP einen Parameter empfängt wird automatisch eine Variable mit dem Namen und dem Wert des Parameters erzeugt.
  Zugriff nur mittels Superglobalen!
- Syntax der angehängten Parameter
  - Parameter haben die Form: Parametername=Wert
  - ? trennt URL und Parameter
  - & verbindet mehrere Parameter
- Sonder- und Leerzeichen in Parametern
  - Sonder- und Leerzeichen in URLs sind durch den Hexadezimalcode des entsprechenden ASCII-Zeichens zu ersetzen. Schreibweise: %Hexcode
  - PHP-Funktion urlencode: `$parameter=urlencode($parameter)`

Codebeispiel: Parameterübergabe

- An die HTML-Links wird ein Parameter angehängt.

index.htm

```
...
<a href="kategorie.php?kategorie=PopRock">PopRock</a><br>
<a
href="kategorie.php?kategorie=Alternativ">Alternativ</a><br>
<a
href="kategorie.php?kategorie=Electronic">Electronic</a><br>
...
```

- In PHP wird automatisch eine entsprechende Variable erzeugt, die anschließend ausgegeben werden kann.

kategorie.php

```
...
<H1>Kategorie: <? echo $_GET[`kategorie`] ?></H1>
...
```

Einfache Auswertung von Formularen I

- Im HTML-Code des Formulars werden alle Felder eindeutig über name-Attribute bezeichnet.
- Das HTML-Formular ruft ein PHP-Skript auf.

Formular.html

```
<form name="Test" method="get" action="Auswertung.php">
  <input type="text" name="vorname"><br>
  <input type="text" name="name"><br>
  <input type="submit" name="Abschicken" value="Abschicken">
</form>
```

- In PHP werden automatisch Variablen mit den Namen der Formularfelder erzeugt.

Auswertung.php

```
<P>Sie haben folgende Werte eingegeben:</P>
<?
 print "Name:".$_GET['name']."<br>
                      Vorname:".$_GET['vorname'];
?>
<P>Bis bald!</P>
```

- Zugriff auf Post und Get-Daten via assoziativer Arrays

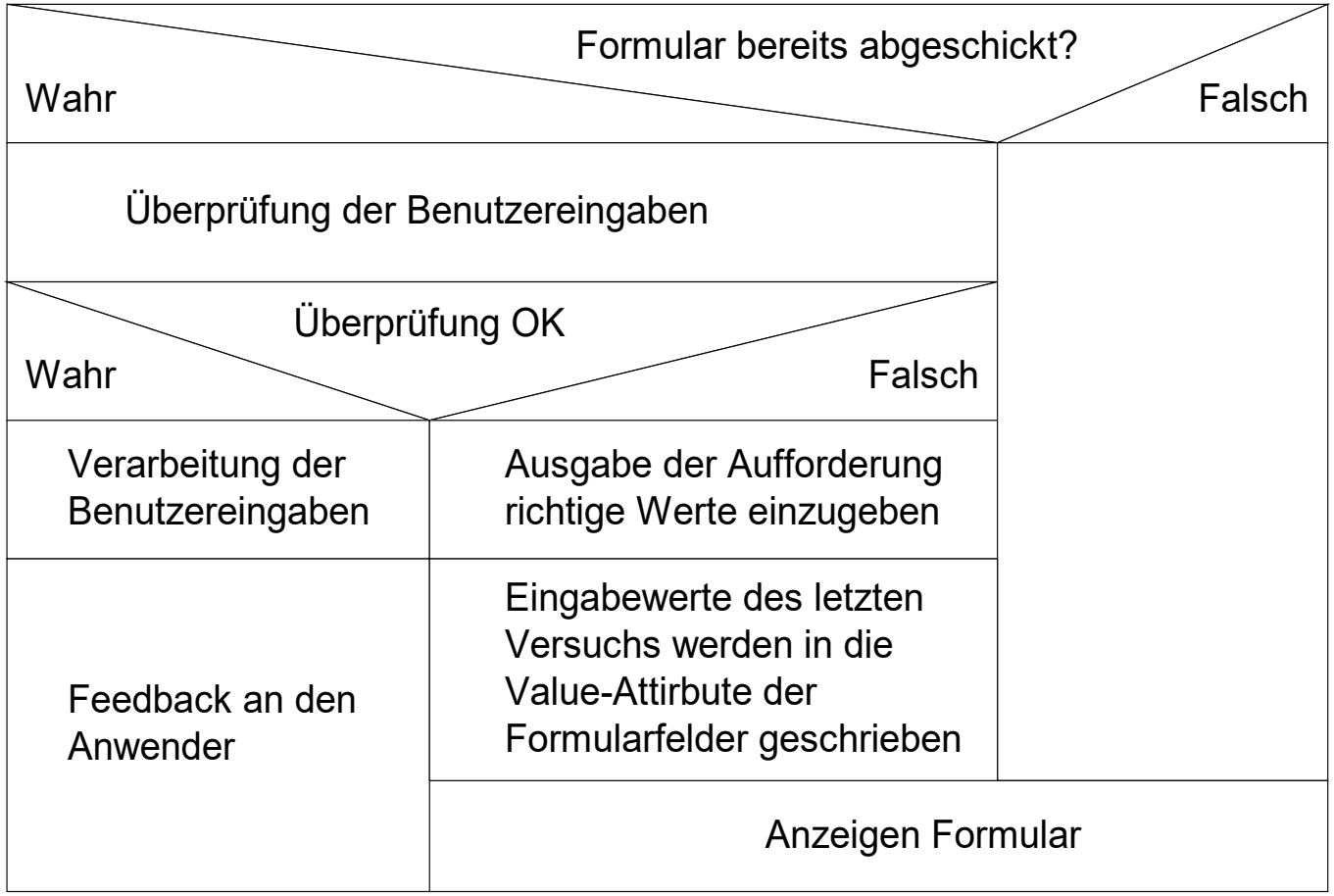
Anforderungen an die Formularauswertung

1. Vollständigkeit
   - Alle notwendigen Formularfelder sollten ausgefüllt werden.
   - Falls nicht
     - sollte der Benutzer einen Hinweis auf fehlende Felder erhalten und
     - es sollte keine Neueingabe bereits eingegebener Daten erforderlich sein.
2. Plausibilität
   - Alle Daten sind auf Plausibilität zu prüfen (eMail-Adresse? Kreditkartennr.?).

- Exkurs: Sicherheit
  - Eine client-seitige Prüfung reicht nicht aus, da JavaScript deaktiviert werden kann.
  - Es sind Angriffe auf den Server mittels Formularinhalten möglich:
    - Über Zeichen wie “ oder ; kann versucht werden Befehle beispielsweise in der Datenbank auszuführen.
    - Der Server bzw. die Datenbank kann mit Megabyte großen Formularinhalten überschwemmt werden. (Denial-of-Service-Angriff)
  - Fazit: Bei professionellen Anwendungen sind Eingaben zusätzlich auf Größe und sicherheitsrelevante Zeichen zu prüfen.

Professionelle Auswertung von Formularen

- Das Formular-Skript ruft sich selber auf (Affenformular-Technik).
- Darstellung der Anwendungslogik als Struktorgramm:



Code FormAuswert.php als Affenformular (Anfang)

```
<html><head><title>Formulardemo</title></head><body>
<?php  // Prüfen ob Formular abgeschickt wurde oder nicht
 if ($_GET) {
    // Validierung: Prüfen ob das Namens-Feld ausgefüllt wurde
    $fehlermeldung = "";
    if ($_GET['name'] == "") { $fehlermeldung = $fehlermeldung . "Name"; }
    // Pruefen ob ein Fehler gefunden wurde
    if ($fehlermeldung == "") {
        // Verarbeiten der Formulareingaben z.B. in Datenbank schreiben
       // Feedback an den Benutzer  ?>
        <p>Sie haben folgende Werte eingegeben:</p>
        <?php   print "Vorname: " . $_GET['vorname']. "<br/>";
                print "Nachname: " . $_GET['name']. "<br/>"; ?>
        <p>Bis bald!</p>
      <?php
    } else {// Ausgabe der Fehlermeldung
      ?>
      <p>Bitte füllen Sie folgende Felder aus: <? echo $fehlermeldung ?></p>
      <?php
    } // Ende: Fehler oder nicht
} // Ende: Abgeschickt oder nicht
```

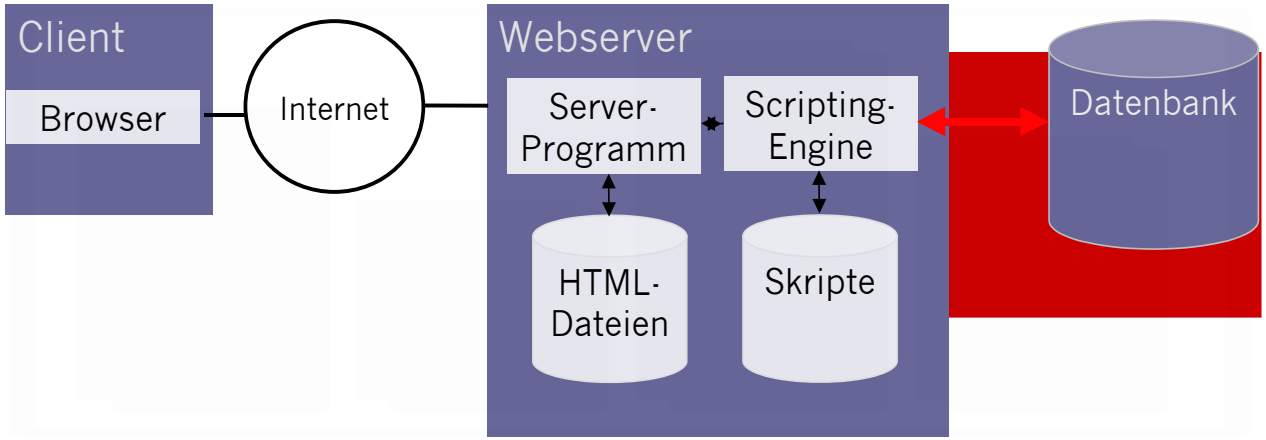
Code: Affenformular (Fortsetzung)

```
// Formular anzeigen, falls :
// 1. Der Benutzer zum ersten mal auf die Seite kommt oder
// 2. das Formular schon abgeschickt wurde und die Prüfung einen Fehler ergab

if (! $_GET) or ($fehlermeldung !="")) {

?>

    $vorname = $_GET? $_GET[‚vorname‘]:“”;

    <form name="Test" method="get" action=„FormAuswert.php">
      <input type="text" name="name"><br>
      <input type="text" name=„vorname" value='<?php echo $vorname>'><br>
      <input type="submit" name="Abschicken" value="Abschicken">
    </form>


<?php } ?>
</body>
</html>
```

Als Grundlage dient das Beispiel von Seite 27:
→ Statt Formular.html und Auswertung.php gibt es jetzt nur noch das Php-Script FormAuswert.php

Datenbankanbindung mit PHP

Datenbankanbindung mit PHP



- PHP verfügt über datenbankspezifische Schnittstellen zu:
  - MySQL, MS SQL-Server, Oracle, Sybase, dBase, Informix
  - InterBase, mSQL, PostgreSQL, DBM
- Datenbankzugriffe in PHP erfolgen über:
  - datenbankspezifische Schnittstellen (hohe Performance)
  - Connectivity-Middleware wie ODBC (geringere Performance)

Beispiel 1: Staus aus der Datenbank holen

- Relationstyp stau_meldungen in der Datenbank staudb.sql:

  stau_meldungen (id, datum, aktiv, autobahn, info)

```
// MySQL-Verbindung zur Datenbank aufbauen
$Connection_ID = mysqli_connect(„server", „user", „pwd") or
       die(mysqli_error());

mysqli_select_db(„staudb",$Connection_ID ) or
       die(mysql_error());

// SQL Query-String definieren
$Query_String = "SELECT * FROM stau_meldungen
                 WHERE aktiv = '1'
                 ORDER BY autobahn, datum desc;";

// Abfrage ausfuehren
$Result = mysqli_query($Query_String);
```

Statt `die` besser PHP error reporting verwenden (Development und Test):
mysqli_report(MYSQLI_REPORT_ERROR | MYSQLI_REPORT_STRICT);

Beispiel 1: Ausgabe der Informationen

- Über eine While-Schleife werden alle Tupel des Abfrageergebnisses durchlaufen und
- die Inhalte in den HTML-Code der Seite geschrieben.

```
<?
// Schleife über alle Tupel
while($Row = mysqli_fetch_assoc($Result)){?>
   <p>
   Autobahn: <? echo $Row[autobahn] ?><br>
   Meldung: <? echo $Row[info] ?><br>
   Datum: <? echo $Row[datum] ?>
   </p>
   <hr size="2">
<? } ?>
```

Beispiel 2: Autobahn-Auswahl

```
// Informationen zu den Autobahnen aus DB holen
$Query_String = "SELECT DISTINCT autobahn
                 FROM stau_meldungen
                 WHERE aktiv = '1'
                 ORDER BY autobahn;";
$Result_Menue = mysqli_query($Query_String);


// Ausgabe der Autobahninformationen
while(($Row = mysqli_fetch_row($Result_Menue )) {
     echo "<p><a href='index.php?autobahn="
          . $Row[3]
          . "'>"
          . $Row[3]
           . "</a></p>"; };
```

- Ergebnis:

```
<p><a href='index.php?autobahn=A01'>A01</a></p>
<p><a href='index.php?autobahn=A07'>A07</a></p>
....
```

Situationsspezifische SQL-Befehle

- Die meisten Datenbankabfragen in Webapplikationen sind situationsspezifisch.
- Beispiel: Nur die Staus einer vom Nutzer gewählten Autobahn aus der Datenbank holen
- Vorgehen:
  - Dem Skript wird über einen Parameter das Auswahlkriterium übergeben
  - Ein passendes SQL-Kommando wird zusammengebaut und ausgeführt

```
// SQL-Query-String situationsspezifisch zusammenbauen
$Query_String = "SELECT * FROM stau_meldungen
                 WHERE aktiv = '1'
                 AND autobahn ='" . $_GET[`autobahn`] . "'
                 ORDER BY datum;";
// SQL-Kommando ausführen
$Result_ID = mysql_query($Query_String);
```

MySQL - Injection

- Eine SQL Injection nennt man das Einfügen von SQL Kommandos, die an dieser Stelle nichts zu suchen haben:

```
// SQL-Query-String situationsspezifisch zusammenbauen
$Query_String = "SELECT * FROM stau_meldungen
                 WHERE aktiv = '1'
                 AND autobahn ='".$_GET['autobahn'."'
                 ORDER BY datum;";


/* was passiert wenn man z.B im Eingabefeld nun: A13'; DROP
staudb;-- angibt. Es wird nun diese Query erzeugt:*/


SELECT * FROM stau_meldungen WHERE aktiv = '1'
                 AND autobahn ='A13'; DROP DATABASE staudb;--';
```

- Injection vermeiden:

  if (!get_magic_quotes_gpc()) {
  $autobahn = addslashes($autobahn); } // auch *mysql_real_escape_string()*.

Formulareingaben in die Datenbank schreiben

- Nach ihrer Validierung werden die Formulareingaben über ein „INSERT INTO“ Kommando in die Datenbank geschrieben:

```
// SQL-Kommando zur Schreiben der Daten zusammenbauen
$Query_String = "INSERT INTO stau_meldungen
                      (autobahn, info, aktiv, datum)
                 VALUES
                      ('" . $autobahn . "','" . $meldung .
                       "','1','" . $datum . "');";

// SQL-Kommando ausfuehren
$Result_ID = mysqli_query($Query_String);
```

- Hinweis:

  Beachten: Anführungszeichen im SQL-Kommando sind durch Hochkommata ‘ zu ersetzen.

Auch möglich: OOP Zugriff auf eine SQL-DB

```php
<?php
// Verbindungs-Objekt samt Zugangsdaten festlegen
@$db = new mysqli("localhost", "root", "", "testbank");

// Verbindung überprüfen
if (mysqli_connect_errno()) {
   printf("Verbindung fehlgeschlagen: %s\n", mysqli_connect_error());
   exit();
}

// SQL-Befehl ausführen
$befehl = $db->query("SELECT version() AS version");

// Antwort der Datenbank in ein Objekt übergeben
$resultat = $befehl->fetch_object();

// MySQL-Version aus dem Resultat-Objekt auslesen
echo "Wir arbeiten mit MySQL-Version {$resultat->version}";

// Verbindung zum Datenbankserver beenden
$db->Close();
?>
```

Cookies

- Setzen von Cookies zur Identifikation wiederkehrender Benutzer:
  übermitteln der Namen-Werte Paare via HTTP-Header an Browser:

```php
<?php
setcookie("name","value",time()+$int);
/*name is your cookie's name, value is cookie's value
$int is time of cookie expires*/
?>
```

http-Response Header (zum Client)

```
Set-Cookie: <name>=<value>[; <name>=<value>]...
[; expires=<date>][; domain=<domain_name>]
[; path=<some_path>][; secure][; httponly]
```

- Cookies auslesen:

```php
$cookie = $_COOKIE["name"];
echo "Der Inhalt des Cookies: $cookie";
```

- Löschen von Cookies:

```php
unset($_COOKIE["name"]); //oder
setcookie("name","",time() - 3600);
```

Sessions

- Im Gegensatz zu Cookies bestehen Sitzungsdaten nur solange, wie Verbindung zum Server existiert.
- Üblicherweise schließt Client Sitzung (Tabs schließen oder Browser), aber auch Server z.B. bei max. Sitzungszeit oder im Fehlerfall
- Sitzungsdaten werden meist vom Webserver verwaltet und über eine eindeutige Session-ID hergestellt mit Client (meist mit Hilfe über Cookies, aber auch über GET-Parameter).

```php
<?php
session_start();

if (!isset($_SESSION["aufrufe"])) {
 $_SESSION["aufrufe"] = 1; // erster Aufruf: Initialisierung
 } else {
 $_SESSION["aufrufe"]++; // weitere Aufrude: hochzählen ...
 }
echo "Sitzungszaehler: " . $_SESSION["aufrufe"];
echo "<br/>Sitzung: Name = ".session_name().", ID = ".session_id();
?>
```

Sessions ohne Cookies

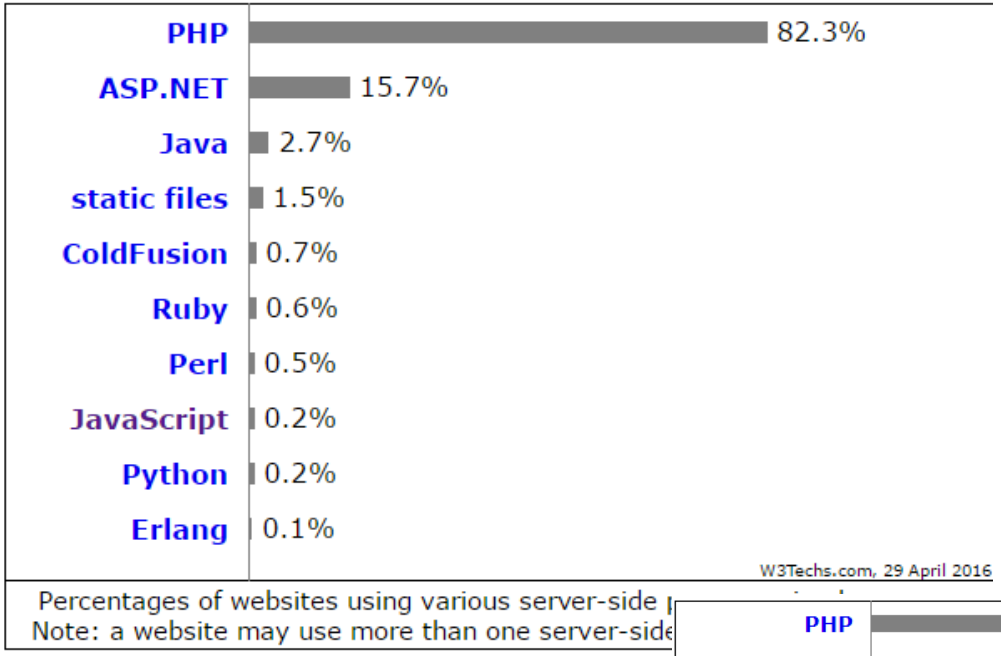
```php
<?php
//Eine Sitzung ohne Cookies mit erneuerter Session-ID
ini_set("session.use_cookies", false);
ini_set("session.use_only_cookies", false);
session_start();
session_regenerate_id(true);

if (!isset($_SESSION["aufrufe"])) {
 $_SESSION["aufrufe"] = 1;
} else {
 $_SESSION["aufrufe"]++;
}
echo "Anzahl Aufrufe dieser Sitzung: " . $_SESSION["aufrufe"] . "<br/>";
echo "<a href='$_SERVER[PHP_SELF]?".session_name()."=".session_id().
"'>Sitzung aufrechterhalten</a>";
?>
```

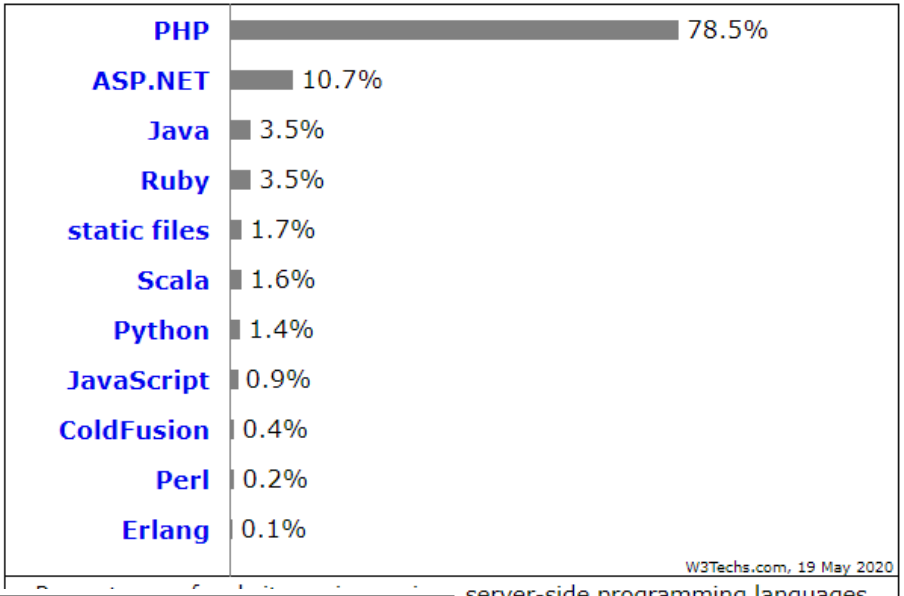
Vor- und Nachteile von PHP

- Vorteile
  - Leicht zu erlernen
  - Vielseitig einsetzbar
  - Code kann nicht gestohlen werden
  - Open Source
  - Für alle gängigen System verfügbar
  - Sehr gut dokumentiert, weit verbreitet

- Nachteile
  - Sicherheitslücke bei den übergebenen Variablen
    http://example.com?variable1="DROP TABLE"
  - Langsamer, da Skriptsprache – Verbesserung JIT
  - Jede Interaktion des Users muss erst vom Server berechnet werden

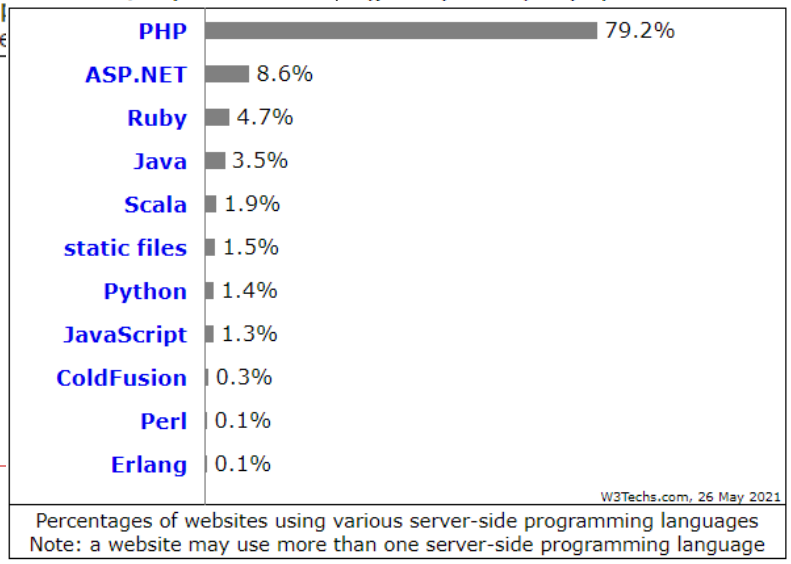
Weitere serverseitige Techniken





Percentages of websites using various server-side p
Note: a website may use more than one server-side

server-side programming languages
e server-side programming language



Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language