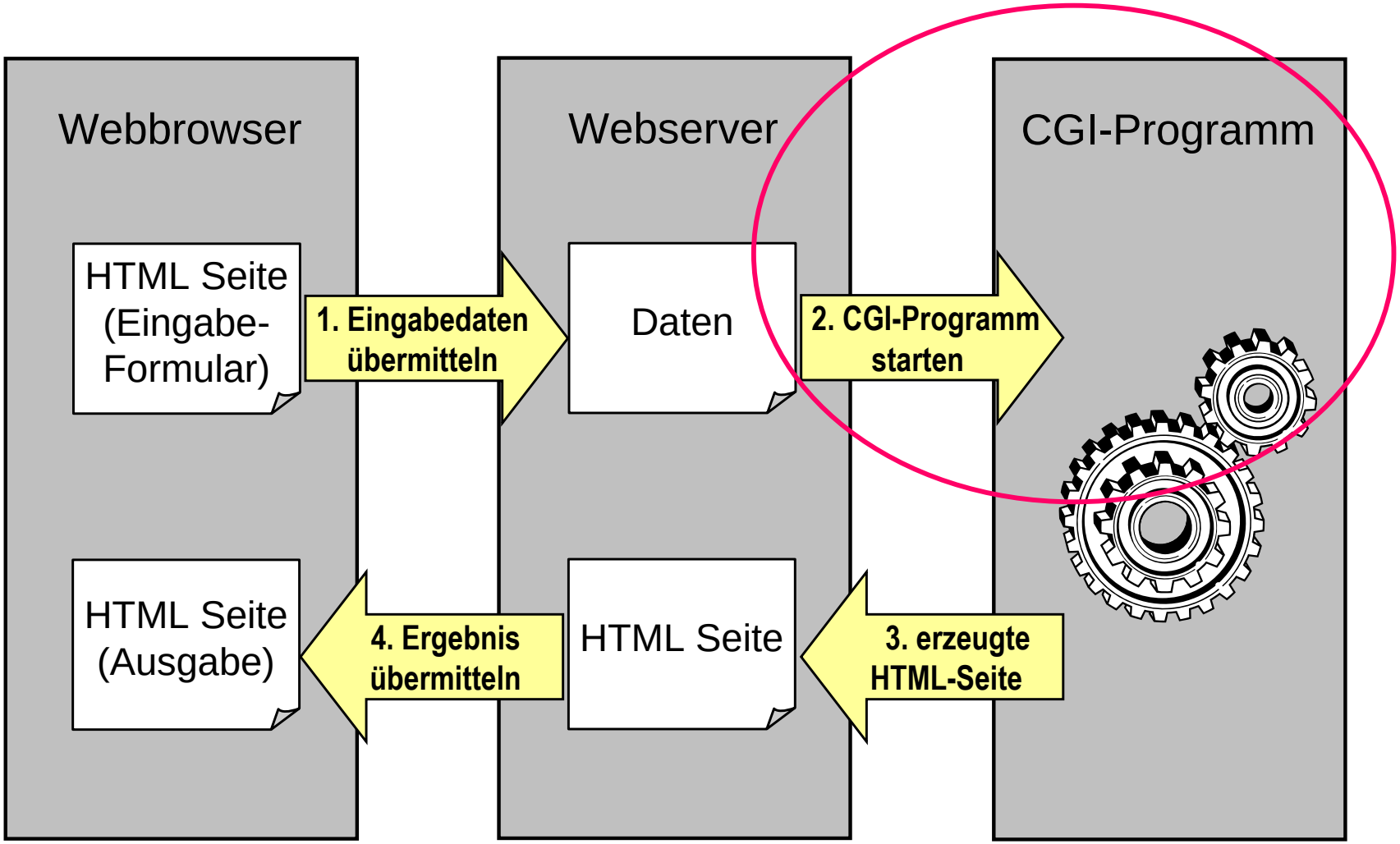
Entwicklung webbasierter Anwendungen

8. Kapitel: CGI

Quellenhinweis:
Viele Folien dieser Vorlesung entstammen der gleichnamigen Vorlesung von Prof. B. Kreling

CGI - Common Gateway Interface



CGI

- Schnittstelle zu Programmen, die per

  HTTP-GET oder -POST aufgerufen werden können
  - ⇨ über Hyperlinks oder durch Absenden von Formularen
  - ⇨ können Daten auf dem Server speichern
  - ⇨ können (datenabhängige) HTML-Datei als Antwort generieren
- Wird vom Webserver angeboten
- Programmiersprache ist nicht vorgeschrieben
  - ⇨ Unix Shell, Perl, PHP, C++, Visual Basic, ...
  - ⇨ Interpretersprachen wie Perl und PHP ersparen Compilation auf Webserver (oft Unix-Maschinen), so dass einfacher Upload genügt
- Anwendungsbeispiele:
  - ⇨ Zugriffszähler, Gästebücher, Statistiken
  - ⇨ Auswahl und Sortierung von Daten, Auskunftssysteme
  - ⇨ Buchungssysteme, Online Shops

Ablauf einer typischen CGI-Kommunikation

- Benutzer füllt HTML-Formular aus und klickt "Submit"
- Browser schickt Formulardaten mit HTTP-POST an ein CGI-Skript

  (Attribut "action" des Formulars)
- Server startet CGI-Skript als selbständiges Programm
  - ⇨ CGI-Skript liest Formulardaten von `cin`
  - ⇨ CGI-Skript liest/speichert Daten aus/in Datenbank oder Datei
  - ⇨ CGI-Skript schreibt HTML-Antwort nach `cout`
  - ⇨ CGI-Skript schreibt Fehlermeldungen nach `cerr`

    (werden vom Server in \logs\error.log geschrieben)
- Server überträgt cout-Strom wie HTML-Datei an Browser

Umgebungsvariablen (1)

Test mit http://localhost/cgi-bin/printenv.pl

Werte werden bei einem CGI-Aufruf vom Webserver gesetzt

- Informationen zum Request

```
REQUEST_URI=/cgi-bin/CgiTest.exe?Name=&Kommentar=
    SCRIPT_NAME=/cgi-bin/CgiTest.exe
HTTP_REFERER=http://localhost/CgiTestFormular.htm
HTTP_COOKIE=Keksname=Kruemel; nochEinKeks=hart
```

sofern gesetzt

- und je nach Methode

```
REQUEST_METHOD=GET
QUERY_STRING=Name=Hugo&Kommentar=BlaBla
```

- oder

```
REQUEST_METHOD=POST
CONTENT_LENGTH=28
CONTENT_TYPE=application/x-www-form-urlencoded
QUERY_STRING=
```

Auf das Ende achten!

Formulardaten hier aus der Standardeingabe (cin in C++)

Umgebungsvariablen (2)

- Informationen über den Client (Browser)

```
HTTP_ACCEPT=image/gif, image/x-xbitmap, image/jpeg, */*
Liste der MIME-Typen, die der Browser darstellen kann
HTTP_ACCEPT_ENCODING=gzip, deflate
HTTP_ACCEPT_LANGUAGE=de
HTTP_CONNECTION=Keep-Alive
HTTP_USER_AGENT=Mozilla/4.0 (compatible; MSIE 5.5)
HTTP_HOST=192.168.0.1
REMOTE_ADDR=192.168.0.11
REMOTE_PORT=1029
```

Umgebungsvariablen (3)

- Informationen über den Server

```
DOCUMENT_ROOT=/apache/htdocs
SCRIPT_FILENAME=/apache/cgi-bin/cgitest.exe
SERVER_ADDR=192.168.0.1
SERVER_ADMIN=admin@xyz.de
SERVER_NAME=localhost
SERVER_PORT=80
SERVER_SIGNATURE=Apache/1.3.14 Server at localhost Port 80
SERVER_SOFTWARE=Apache/1.3.14 (Win32)
GATEWAY_INTERFACE=CGI/1.1
SERVER_PROTOCOL=HTTP/1.1
```

- Informationen über das Server-Betriebssystem

```
COMSPEC=C:\WINDOWS\COMMAND.COM
PATH=C:\WINDOWS;C:\WINDOWS\COMMAND
WINDIR=C:\WINDOWS
```

Codierung von Formulardaten (I)

- Umgebungsvariablen sind grundsätzlich Strings
- Name und Wert eines Formularelements werden durch Gleichheitszeichen = getrennt
- Leerzeichen innerhalb des Werts werden durch Pluszeichen + ersetzt
- die Name/Wert-Paare mehrerer Formularelemente werden durch & getrennt.
- Sonderzeichen, Umlaute etc. werden durch das Prozentzeichen % und 2 Hexadezimal-Ziffern dargestellt
- die hier aufgeführten Sonderzeichen = + & % werden in Namen und Werten ebenfalls hexadezimal codiert

```
QUERY_STRING=Text=Hallo+dies+ist+ein+Test&Zeichen=%21
```

Codierung von Formulardaten (II)

Zeichen	Code	Zeichen	Code	Zeichen	Code	Zeichen	Code
Leerz.	+	!	%21	"	%22	#	%23
$	%24	%	%25	&	%26	'	%27
(	%28	)	%29	+	%2B	,	%2C
/	%2F	:	%3A	;	%3B	<	%3C
=	%3D	>	%3E	?	%3F	[	%5B
\	%5C	]	%5D	^	%5E	"	%60
{	%7B	|	%7C	}	%7D	~	%7E
°	%A7	Ä	%C4	Ö	%D6	Ü	%DC
ß	%DF	ä	%E4	ö	%F6	ü	%FC

Text=Hallo+dies+ist+ein+Test&Zeichen=%21

Prinzipieller Aufbau eines CGI-Skripts

Text=Hallo+dies+ist+ein+Test&Zeichen=%21

1. Requestmethode bestimmen

   REQUEST_METHOD=GET bzw. POST

2. Für POST: Daten von STDIN einlesen

   CONTENT_LENGTH auslesen und beachten!

   Für GET: Daten aus Umg.variable QUERY_STRING

   Es gibt kein Daten-Ende-Zeichen

3. Strings zerlegen und „interessante“ Daten rausfiltern
   1. & - Zeichen trennt "Name=Wert"-Paare
   2. = - Zeichen trennt Name und Werte
   3. Sonderzeichen rekonstruieren + als Leerzeichen, %xx
4. Eigentliche Aufgabe ausführen

   z.B. Datenbankanfrage

5. Zurückliefern des Ergebnisses
   - als Datenstrom im HTML-Format

     HTML-Sonderzeichen müssen codiert werden (z.B. € als €)
   - Als Bilddaten im GIF oder JPEG-Format

Beispiel: CGI-Skript in C++ (CgiTest.cpp)

```
int main(int argc, char* argv[], char* envp[])
    // envp ist ein Array von Zeigern auf nullterminierte Strings. Diese
    // Strings enthalten die "Umgebungsvariablen" im Format "Name=Wert".
    {
          // HTTP-Header für Antwort an Browser (verlangt 2 Zeilenendezeichen):
          cout << "Content-type: text/html" << endl << endl;

          // Anfang der HTML-Datei schreiben:
          cout << "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.01//EN\">" << endl;
          cout << "<html>" << endl;
          cout << "<head>" << endl;
          cout << "  <title>CGI-Test</title>" << endl;
          cout << "</head>" << endl;
          cout << "<body>" << endl;

          try {
           // alle Umgebungsvariablen ausgeben:
           cout << "  <h3>Umgebungsvariable</h3>" << endl;
           cout << "  <p>" << endl;
           int i = 0;
           while (envp[i]!=NULL) {
                      cout << "    " << envp[i] << "<br>" << endl;
                      i++;
           }
           cout << "  </p>" << endl;
```

Beispiel: CGI-Skript in Perl (echo.pl)

```
# Formulardaten in einzelne Parameter zerlegen mit '&' als Trenner:
my @ParameterListe = split(/&/, $ParameterString);

print "  <p><strong>Einzelne Parameter:</strong><br>\n";
my $Parameter;
foreach $Parameter (@ParameterListe)
{
        # Parameter in Name und Wert zerlegen mit '=' als Trenner:
        my $Name;
        my $Wert;
        ($Name, $Wert) = split(/=/, $Parameter);

        # Leerstellen restaurieren ('+' ersetzen durch ' '):
        $Wert =~ tr/+/ /;

        # Hex-Codes %xx umwandeln in Character:
        $Wert =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C" , hex($1))/eg;

        # HTML-Sonderzeichen '&', '<', '>' kodieren:
        $Wert =~ s/&/&amp;/;
        $Wert =~ s/</&lt;/;
        $Wert =~ s/>/&gt;/;

        # Parameter ausgeben in HTML-Datei:
        print "$Name = $Wert <br>\n";
}
```

Syntax gewöhnungs-bedürftig…

…aber sehr mächtig und bequem

Apache für CGI konfigurieren

- Skript-Verzeichnisse definieren
  - ⇨ alle Dateien darin werden ausgeführt, nicht übertragen

```
ScriptAlias /cgi-bin/ "/apache/cgi-bin/"
ScriptAlias /php/ "/apache/php/"
```

- oder Datei-Endungen als ausführbar definieren
  - ⇨ wenn Skripte in beliebigen Verzeichnissen liegen

```
AddHandler cgi-script .cgi
AddHandler cgi-script .pl
```

- und dem MIME-Type eines Skripts den Pfad zum ausführbaren Programm zuordnen
  - ⇨ d.h. "Dateien dieses Typs öffnen mit ..."

```
Action application/x-httpd-php /php/php.exe
```

Typische CGI-Aufrufe

Ein CGI-Script kann aus einer HTML-Datei heraus auf verschiedene Arten aufgerufen werden:

- **Formular:**
  - Beispiel: `<form action="/cgi-bin/myscript.pl" method="get">`
  - Anwendung: Suchdienste, Gästebücher oder elektronische Einkaufskörbe
- **Verweise:**
  - Beispiel: `<a href="/cgi-bin/myscript.pl">Los</a>`
  - Anwendung: Statistik-Abfragen
- **Grafikreferenz:**
  - Beispiel: `<img src="/cgi-bin/counter.pl">`
  - Anwendung: grafische Zugriffszähler
- **Server Side Include:**
  - Beispiel: `<!--#exec cgi="/cgi-bin/counter.pl" -->`
  - Anwendung: textbasierte Zugriffszähler

CGI-Scripts, die keinen Input vom Anwender benötigen

Das CGI-Script muss Daten im GIF- oder JPEG-Format zurücksenden.

Untypische CGI-Aufrufe

- **direkte Eingabe der URL im Browser:**

  Beispiel: http://localhost/cgi-bin/printenv.pl

  Anwendung: Test

- **sofortiger automatischer Aufruf beim Laden einer Seite:**

  Beispiel: <img src="/cgi-bin/script.pl">

  Anwendung: Anzeigen einer Statistik

- **verzögerter automatischer Aufruf beim Laden einer Seite:**

  Beispiel: <meta http-equiv="refresh"
  content="3; URL=/cgi-bin/script.pl">

  Anwendung: nach 3 Sekunden die neue URL aufrufen

Entwicklung webbasierter Anwendungen

9. Kapitel: PHP

Quellenhinweis:
Viele Folien dieser Vorlesung entstammen der gleichnamigen Vorlesung von Prof. B. Kreling

Situation

- HTML unterstützt statische Seiten
- CSS unterstützt Layout
- ECMA-Script bietet zusammen mit DOM mächtige Funktionalität
- Bisherige CGI-Skripte liefern ein universelles Werkzeug
  - ⇨ Anbindung an „normale“ Programmiersprachen: Perl, C++,…
  - ⇨ Datenverarbeitung auf dem Server ist möglich
- Aber:
  - ⇨ Programmierung ist teilweise unbequem
  - ⇨ CGI-Dateien sind vollkommen unabhängig von HTML (nicht integriert)
  - ⇨ Universelle Programmiersprachen wurden nicht für dynamisches HTML entwickelt

Dynamisch erzeugte Webseiten sind so schwer zu erstellen!

Grundidee

Skriptsprache, welche speziell für die Webprogrammierung geeignet ist, und in HTML eingebettet werden kann.

PHP bedeutet "*PHP: Hypertext Preprocessor*" (rekursiv)

- ⇨ Integriere den PHP-Code in die HTML-Datei (ähnlich ECMA-Skipt)
- ⇨ Beim Aufruf durch einen Web-Browser, erkennt der Web-Server, (der die Datei zum Browser übermittelt), dass es sich um eine HTML-Datei mit eingebettetem PHP-Code handelt.
- ⇨ Der server-seitig installierte PHP-Interpreter analysiert die PHP-Code-Passagen, führt den Code aus und erzeugt daraus den endgültigen HTML-Code, der an den Browser gesendet wird.
- ⇨ PHP erweitert Perl um viele aktuelle Belange des Web-Publishings
  - ⇨ z.B. PDF-Dateien dynamisch generieren und an den Browser senden

Mittlerweile ist so viel eingebaut, dass Performance ein Thema ist!

Historie

- 1994: Rasmus Lerdorf (*1968 in Grönland) beginnt mit einem Hack
- 1995: PHP/FI 1.0 PHP - "Personal Home Page Tools", FI - "Form Interface"

  1995: PHP/FI 2.0 noch ohne echten Parser
- 1997: PHP 3.0 "Personal Home Page" oder "PHP HyperText Preprocessor"
  - ⇨ echter Parser
  - ⇨ Interpreter
  - ⇨ grundlegende OO Notation
- 2000: PHP 4.0
  - ⇨ Interpiler wie Perl 5
  - ⇨ Performancegewinn von Faktor 2-5x im Einzelfall 100x.
  - ⇨ neuer, schnellerer Sprachkern "Zend"
  - ⇨ viele, neue Funktionen (mehrere Tausend)
- 2003: PHP 5.0
  - ⇨ diverse Optimierungen
  - ⇨ Objektorientierung
- 2006: PHP 5.1.1 (und PHP 4.4.2) in der XAMPP-Suite enthalten

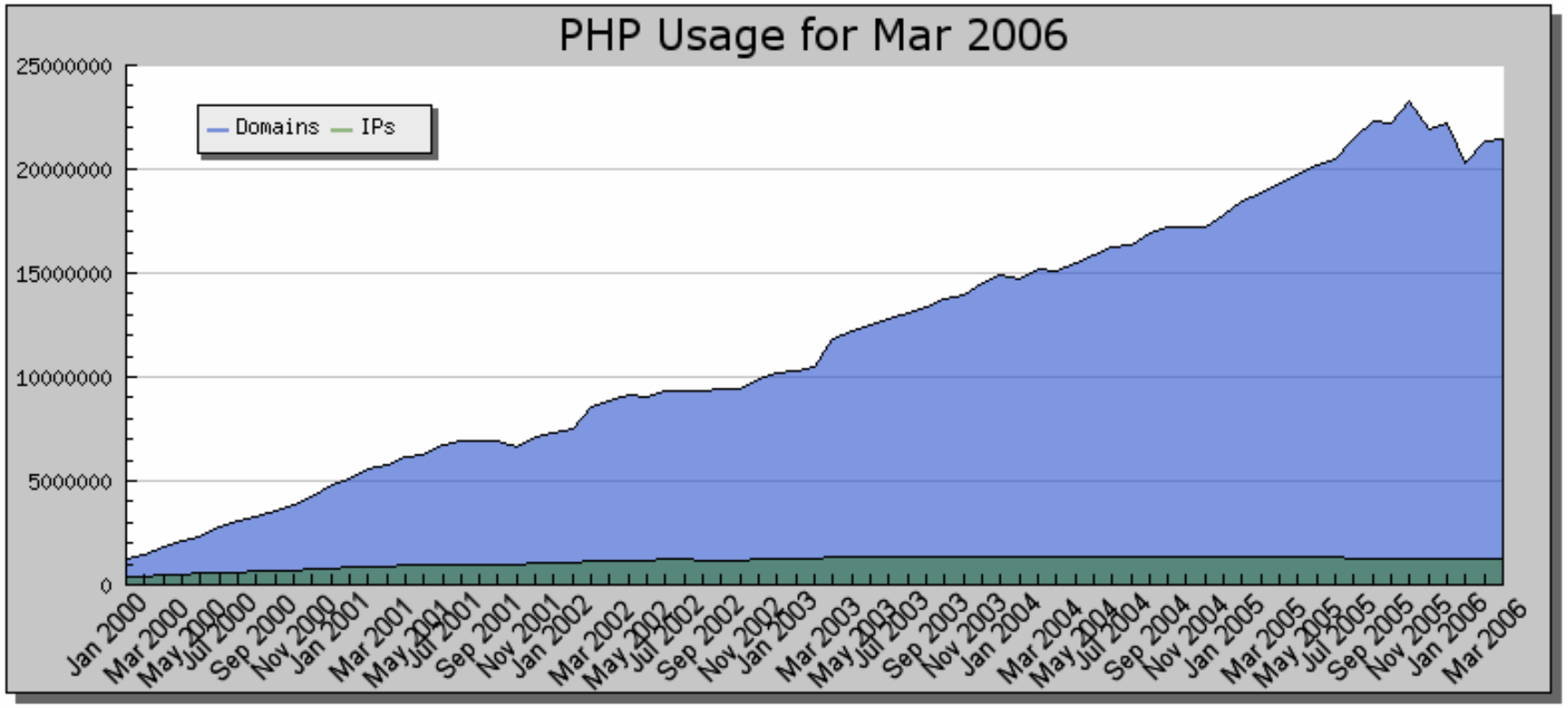
Open Source

Verbreitung

http://www.php.net/usage.php

Usage Stats for March 2006

PHP: 21,439,178 Domains, 1,277,736 IP Addresses
Source: Netcraft



In der Praxis hat sich PHP bewährt und wird von vielen großen Web-Angeboten mit Erfolg eingesetzt.

Primitives Beispiel

```
<?php   header ("Content-type: text/html");   ?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
        content="text/html; charset=iso-8859-1">
  <title>Hello World</title>
</head>
<body>
  <p> Hello
<?php
    echo "World</p>";
?>
</body>
</html>
```

PHP und HTML mischen

- eine PHP-Datei ist ein Programm für den PHP-Interpreter
  - ⇨ und nicht etwa eine HTML-Seite für den Browser
  - ⇨ die Ausgabe des Interpreters ist typischerweise eine HTML-Seite
- Text außerhalb der Klammern `<?php ... ?>` wird direkt in die Ausgabe kopiert
  - ⇨ "reiner HTML-Code" innerhalb einer PHP-Datei ist eine sehr kompakte Ausgabeanweisung (entspricht `echo`)
  - ⇨ auch Leerstellen und Leerzeilen werden in die Ausgabe kopiert
  - ⇨ weitere Schreibweisen für die Klammer

XHTML-konform

SGML-konform

```
<? ... ?>   <?= $Variable ?>   <% ... %>
<script language="php"> ... </script>
```

ECMA-Stil

- 2 Modi: "PHP ausführen" und "HTML kopieren"
  - ⇨ zu Beginn der Datei ist der Interpreter im Modus "HTML kopieren"

PHP Crashkurs

- besondere Stärke in der Kombination mit HTML
- Syntax, Operatoren und Steueranweisungen ähnlich C++
- nicht typisiert ähnlich JavaScript
  - ⇨ Funktionsdeklaration mit function
  - ⇨ keine Variablendeklaration
  - ⇨ float, nicht double
- alle Variablennamen beginnen mit **$**
- Konstantendefinition für einfache Datentypen (ohne **$**)

```
define ("GREETING", "Hello you.");
```

- äußerst reichhaltige (Funktions-) Bibliotheken
  - ⇨ Datenbankzugriffe, Mathematik, Strings, Mail, HTML,…

Strings

können beliebig lang sein

- flexible Schreibweise
  - ⇨ in `"..."` dürfen auch einfache Variable vorkommen

    ```
    echo ("<td>Anzahl: $Anzahl</td>");
    ```
  - ⇨ Sonderzeichen wie in C++: `\n` `\t` `\"` `\\` `\$`
  - ⇨ `'...'` ist ebenfalls möglich als Stringklammer,

    allerdings werden darin keine Variablen ausgewertet
  - ⇨ damit sind "geschachtelte Strings" möglich,

    z.B. HTML-Attribute in PHP-Strings

    ```
    echo ('<p class="Kopf">');
    ```
- Verkettung von Strings mit dem Operator `.`

  ```
  $Begruessung = "Hallo ".$Name;
  ```
- Zugriff auf einzelne Character (beginnt bei 0)

  ```
  $Zeichen = $MeinString{$Zeichenposition};
  ```

Ausgabe

- bei Start von Kommandozeile: Ausgabe auf Konsole

  bei Start vom Webserver: Antwort an Browser

```
echo ( string arg1 [, string argn...] );
print ( string arg );
int printf ( string format [, mixed args] );
```

  - ⇨ echo und print sind gleichwertig
- HTML-Modus außerhalb von <?php ... ?> entspricht echo
- Ausgabepuffer leeren

```
void flush ();
```

  - ⇨ ob der Server das weiterleitet, ist nicht sichergestellt
  - ⇨ normalerweise überflüssig

Assoziative Arrays

- dynamisch (variable Länge), keine Deklaration erforderlich
- wahlweise assoziativ oder indiziert (ähnlich JavaScript)
- Zugriff auf Elemente über Schlüssel (String) oder Index (Integer 0..count-1)

```
$arr[] = 5;              // hängt ans Ende an
$arr[3] = "xyz";         // Zugriff per Index
$map["abc"] = 0.05;      // Zugriff per Schlüssel
$ff = array('Erdbeere' => 'rot' , 'Banane' => 'gelb');
 // Zuweisung als Tupel (Item, Value); $ff['Erdbeere']='rot'
```

  - ⇨ Abarbeitung indiziert:

```
for ($i=0; $i<count($arr); $i++)
  echo ($arr[i]);
```

  - ⇨ Abarbeitung als Kollektion:

```
foreach($_ENV as $Schluessel => $Wert)
  echo (" $Schluessel = $Wert \n");
```

Beispiel: Umgebungsvariablen

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
   "http://www.w3.org/TR/html4/strict.dtd">
  <html>
  <head>
      <meta http-equiv="Content-Type" content="text/html;
    charset=iso-8859-1">
      <title>Umgebungsvariablen</title>
  </head>
  <body>
   <h2>Umgebungsvariablen</h2>
    <pre>
<?php
foreach($_ENV as $key => $value) {
  echo "$key=$value\n";
}
?>
    </pre>
  </body>
  </html>
```

Globale assoziative Arrays

Wirklich ohne _ !

`$GLOBALS`	Liste aller globalen Variablen
`$_COOKIE`	Liste aller Cookies für diesen Server
`$_GET`	alle per GET übermittelten Formulardaten
`$_POST`	alle per POST übermittelten Formulardaten
`$_FILES`	Infos über mit POST hochgeladene Dateien
`$_ENV`	alle Umgebungsvariablen
`$_SERVER`	Umgebungsvariablen vom Server
`$_REQUEST`	$_COOKIE und $_GET und $_POST
	(d.h. potenziell gefährliche Daten vom Client)

Zugriff auf Formulardaten

PHP hat den QUERY_STRING bereits dekodiert

- PHP stellt globale assoziative Arrays bereit
  - ⇨ Name des Formularelements dient als Index

```
if ( isset( $_POST["Elementname"] ))
   echo $_POST["Elementname"];
```

- PHP bildet (je nach Konfiguration) Formularelemente in globale Variable ab
  - ⇨ sehr elegant für Schreibfaule, aber wartungsunfreundlich

```
if ( isset( $Elementname ))
   echo $Elementname;
```

  - ⇨ und gefährlich: Elementname könnte gehackt sein; ein Hacker könnte so eine nicht-initialisierte Variable setzen

- `isset` prüft jeweils, ob die Variable überhaupt existiert

Konfiguration von PHP

- `php.ini` ist zentrale Konfigurationsdatei
- `register_globals = Off` // On ist depreceated!

  Form-Elemente werden nicht mehr in globale Variable abgebildet (Zugriff nur noch über assoziative Arrays)
- `magic_quotes_gpc = Off`

  (sonst wird `"` zu `\"` für $_GET, $_POST, $_COOKIE)
- `phpinfo();`

  generiert eine Übersicht zu PHP als HTML-Seite (mit Pfad zu PHP.ini)

es gibt oft mehrere PHP.ini – Dateien im Filesystem – aber nur eine wird verwendet!

PHP Version 5.0.4 

System	Windows NT RH-FH 5.1 build 2600
Build Date	Mar 31 2005 02:44:34
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--with-gd=shared"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Programme\apache\xampp\apache\bin\php.ini
PHP API	20031224
PHP Extension	20041030
Zend Extension	220040412
Debug Build	no
Thread Safety	enabled
IPv6 Support	enabled
Registered PHP Streams	php, file, http, ftp, compress.zlib, https, ftps
Registered Stream Socket Transports	tcp, udp, ssl, sslv3, sslv2, tls

Beispiel: Formularauswertung

```
<?php
if ($_SERVER["REQUEST_METHOD"]=="GET") {
    $Params = $_GET; echo ("(mit GET übermittelt)\n");
}
else if ($_SERVER["REQUEST_METHOD"]=="POST") {
    $Params = $_POST; echo ("(mit POST übermittelt)\n");
}
if (isset($Params["Anwendername"]))
    echo ("Anwendername=".$Params["Anwendername"]."\n");
if (isset($Params["KommentarText"]))
    echo ("KommentarText=".$Params["KommentarText"]."\n");
echo ("</pre>");
?>
```

Strukturierung und Wiederverwendung

- typische Struktur: viele kleine Programme anstatt eines großen Programms mit vielen Funktionen
  - ⇨ ein Formular benötigt oft 2 PHP-Seiten (Aufbau und Auswertung)
- benötigte Klassen oder Funktionen in eigene PHP-Datei und per include einbinden

```
require("Pfadname.php");
```

  - ⇨ vergleichbar mit #include in C++
  - ⇨ fehlende Datei bewirkt Abbruch bei require, Warnung bei include
  - ⇨ kann in if-Anweisungen stehen und wird dann nur bedingt inkludiert
  - ⇨ Dateiname kann Laufzeitausdruck sein, nicht nur eine Konstante
  - ⇨ innerhalb der require-Datei wird im HTML-Modus begonnen !

kleiner Nachteil:
PHP muss immer
mehrere Dateien öffnen

- Variablen sind "require-übergreifend" sichtbar

Apache für PHP konfigurieren

- PHP unter Verwendung des CGI
  - ⇨ PHP-Installationsverzeichnis zum Skript-Verzeichnis erklären:

    ```
    ScriptAlias /php/ "/apache/php/"
    ```

  - ⇨ PHP-Interpreter für Dateien mit MIME-Type PHP aufrufen:

    ```
    Action application/x-httpd-php /php/php.exe
    ```

  - ⇨ MIME-Type PHP der Datei-Endung .php zuordnen:

    ```
    AddType application/x-httpd-php .php
    ```

  - ⇨ auch index.php als Startseite zulassen:

    ```
    DirectoryIndex index.php
    ```

  - ⇨ nicht nötig für Verzeichnisse mit PHP-Dateien

    ```
    Options ExecCGI
    ```

- effizient: PHP als Apache-Module
  - ⇨ wird eingebunden, wenn Apache compiliert wird
  - ⇨ In der Installation von XAMPP 1.4.13 ist PHP5 integriert

PHP-Dateien können in jedem Dokument-Verzeichnis liegen