PHP

Ludwig-Erhard-Schule

2023-05-27

Dynamisch erzeugte Webseiten – wofür?

- Ausgabe aktueller Daten (z.B. Messwerte)
- Erzeugen von Seiten aus Vorlagen
- Reaktion auf Benutzereingaben/Formulare
- Auszug aus Datenbanken
- Shop-System
- ...

Wie

- Spezieller Server für eine bestimmte Sprache
  - node.js (JavaScript)
  - Apache Tomcat (Java)
- Allgemeiner Server (z.B. Apache)
  - Anbinden beliebiger Programme über CGI
  - Ausführen spezieller Sprachen mit Server-Modulen

CGI: Das Common Gateway Interface

+ Jedes Programm, das Text annehmen und schreiben kann, kann benutzt werden.
– CGI ist langsam.
– Sicherheit ist problematisch.
– In Webhosting-Paketen ist CGI nur gegen Aufpreis erhältlich.
– Das Programm muss auf dem Server lauffähig sein.

Verbreitung

- PHP wurde speziell zur Website-Programmierung entwickelt (domänenspezifische Sprache).
- PHP wird klassischerweise von einem Apache-Modul interpretiert.
- PHP wird von jedem Webhoster standardmäßig angeboten.
- PHP ist die am weitesten verbreitete Sprache zur Webserver-Programmierung.
- Es gibt eine große Community, viel Dokumentation und viele fertige Lösungen.

Geschichte

- PHP entstand aus einem Hobbyprojekt.
- PHP hat viele Eigenschaften von Perl übernommen
  Perl hat viele Eigenheiten aus dem Unix-Umfeld.
- PHP ist inzwischen eine vollwertige Programmiersprache.
  Objektorientierung und funktionale Programmierung sind möglich.
- PHP kann inzwischen auch auf der Kommandozeile ausgeführt werden.
- PHP wird immer noch aktiv weiterentwickelt.

PHP-Seiten

- PHP-Dateinamen müssen mit `.php` enden.
- In PHP-Dateien darf auch HTML verwendet werden.
- PHP-Code wird mit `<?php` eingeleitet.
- Wenn der Code vor dem Ende der Seite endet, muss `?>` folgen.

```
<!DOCTYPE html>
<html lang="de">
<body>
  <?php echo '<h1>Test</h1>'; ?>
</body>
</html>
```

Wichtige Sprachelemente

Formatierung

- PHP ist formatfrei.
- Groß- und Kleinschreibung wird unterschieden.
- Statements enden mit einem Strichpunkt.
- Empfehlungen zum Formatieren in PSR1 und PSR12.

Kommentare

- Einzeilig mit // oder #
- Mehrzeilig mit /* und */

Variablen

- PHP ist eine schwach typisierte Sprache.
- Variablennamen beginnen mit einem $-Zeichen.
- Groß- und Kleinschreibung wird unterschieden.
- Die Gültigkeitsregeln von Variablen sind speziell:
  - Lokale Variablen haben Vorrang vor globalen Variablen.
  - Globale Variablen stehen als Mitglied von `$GLOBALS[]` oder mit Schlüsselwort `global` zur Verfügung.
  - Superglobals stehen immer zur Verfügung.
  - Auf Klassenvariablen muss innerhalb der Klasse mit `$this->...` zugegriffen werden.
  - Statische Variablen behalten ihren Wert über Funktionsaufrufe hinweg.

Variablen prüfen

- `==` vergleicht Werte, `===` vergleicht zusätzlich den Typ.
- `empty()` liefert *falsch* zurück, wenn es eine Variable gibt und sie weder `null` ist noch ein leerer Text.
- `is_null()` liefert *falsch* zurück, wenn es eine Variable gibt und sie nicht den Wert `null` hat.
- `isset()` liefert *wahr* zurück, wenn es eine Variable gibt und sie nicht den Wert `null` hat.
- `array_key_exists()` liefert *wahr* zurück, wenn es im Array diesen Schlüssel gibt, auch wenn der Wert dazu `null` ist.
- `property_exists()` macht das selbe für Eigenschaften von Objekten.

Arrays

- Ein *Array* (Feld) ist eine geordnete Reihe von Werten. Auf Elemente kann man über ihre Nummer zugreifen.
- Ein *assoziatives Array* (Dictionary, Map) ist eine Sammlung von Schlüssel-Wert-Paaren. Über den Schlüssel kann der Wert erreicht werden. Assoziative Arrays sind nicht unbedingt sortiert, der Schlüsselzugriff erfolgt jedoch schnell.
- *PHP-Arrays* sind assoziativ. Wenn kein Schlüssel vergeben wird, werden die Schlüssel durchnummeriert. PHP-Arrays sind nach der Einfügereihenfolge geordnet.

Array-Namen

```
$benutzername = $_GET['benutzername'];
```

- Arrays sind wie Variablen benannt.
- Auf Arrayelemente wird mit dem Variablennamen und dem Schlüssel in eckigen Klammern zugegriffen.
- Wenn der Schlüssel eine Zahl ist, dürfen die Anführungszeichen entfallen.
- Wird kein Schlüssel angegeben, wird automatisch eine (fortlaufende) Nummer verwendet.

Arrays erzeugen

- Erzeugen mit Schlüsseln, lange Form:
  ```
  $geschlechter = array('Bob' => 'm', 'Alice' =>
  'w', 'Kim' => 'd');
  ```
- Erzeugen mit Schlüsseln, kurze Form:
  ```
  $geschlechter = ['Bob => 'm', 'Alice' => 'w',
  'Kim' => 'd'];
  ```
- Erzeugen ohne Schlüssel (lange Form):
  ```
  $primzahlen = array(1, 2, 3, 5, 7, 11);
  ```
- Vorsicht! Sie können nicht gleichzeitig eine Variable und ein Array mit dem selben Namen haben.

Arrays ergänzen

- Schlüssel mit Wert zufügen (keine Fehlermeldung, falls ein existierender Wert dadurch ersetzt wird):
  `$geschlechter['Olaf'] = 'm';`
- Einzelnen Wert hinten anhängen:
  `$primzahlen[] = 23;`
- Werte hinten anhängen:
  `array_push($primzahlen, 13, 17, 19);`
- Arrays können mit `array_merge()` zusammengefügt werden.

Arrays benutzen

- Alle Werte oder Schlüssel-Wert-Paare durchgehen:
  Benutzen Sie eine `foreach()`-Schleife.
- Nach einem Wert suchen:
  `$schluessel = array_search(7, $primzahlen);`
- Nachsehen, ob es einen Schlüssel gibt:
  `if(array_key_exists('Kim', $geschlechter))`
- Ein Schlüssel-Wert-Paar entfernen:
  `unset($geschlechter['Kim']);`
- Elemente zählen:
  `$anzahl = count($geschlechter);`
- Im PHP-Manual finden Sie viele weitere Array-Funktionen.

Wichtige Superglobals

$_GET[ ] hält die Eingaben aus einem mit GET verschickten Formular.

$_POST[ ] hält die Eingaben aus einem mit POST verschickten Formular.

$_SERVER[ ] hält Angaben zu IP-Adressen, zum Skriptnamen und zur Umgebung

$_COOKIE[ ] hält Informationen aus dem aktuellen Cookie.

$_SESSION[ ] hält Angaben zur Sitzung (wenn eine existiert).

Texte

- In ’einfachen’ Anführungszeichen:
  Alles außer \’ wird unverändert übernommen.
- In "doppelten" Anführungszeichen:
  - Steuercodes wie \n können verwendet werden.
  - \, $ und " müssen mit \ maskiert werden.
  - Variablen werden interpoliert. Im Zweifelsfall in geschweifte Klammern packen:

```
echo "<p>Hallo {$benutzer},<br />\n";
```

- Heredoc/Nowdoc:
  Wie doppelte/einfache Anführungszeichen, aber mehrzeilig.

Heredoc und Nowdoc

- Beginn mit <<< und einem frei gewählten Signalwort.
- Bei Nowdoc steht das Signalwort in einfachen Anführungszeichen.
- Ende: Das Signalwort steht alleine in einer Zeile, der Strichpunkt in der nächsten.

```
$gruss = <<<GRUSSENDE
Hallo $benutzer,
schön, dass Sie wieder hier sind.
GRUSSENDE
;
```

Textfunktionen

- Texte ausgeben mit `echo`
- Texte verbinden mit `.`
- HTML-Sequenzen maskieren mit `htmlspecialchars()`
- Länge messen mit `mb_strlen()` oder `strlen()`
- Vergleichen mit `strcmp()` (exakt) oder `strcasecmp()` (Groß- und Kleinschreibung egal)

Anführungszeichen in Texten (I)

Problem

Beim Erzeugen von HTML muss Text mit Anführungszeichen erzeugt werden.

```
<a href="...">
```

Lösung 1

Verwenden Sie einfache Anführungszeichen, setzen Sie den Text zusammen, falls Sie Variable benötigen:

```
$ziel='http://...';
echo '<a href="' . $ziel . '">';
```

Anführungszeichen in Texten (II)

Problem

Beim Erzeugen von HTML muss Text mit Anführungszeichen erzeugt werden.

```
<a href="...">
```

Lösung 2

Verwenden Sie doppelte Anführungszeichen, maskieren Sie die innerhalb des Textes mit einem \:

```
$ziel='http://...';
echo "<a href=\"$ziel\">";
```

Anführungszeichen in Texten (III)

Problem

Beim Erzeugen von HTML muss Text mit Anführungszeichen erzeugt werden.

```
<a href="...">
```

Lösung 3

Verwenden Sie ein Heredoc:

```
$ziel='http://...';
echo <<<LINKENDE
<a href="$ziel">
LINKENDE
;
```

Zeilenenden ausgeben

Problem

In HTML werden Zeilenenden ignoriert. HTML-Quelltext ist aber besser lesbar, wenn er Zeilenenden enthält.

- Erzeugen Sie mehrzeilige Ausgaben mit einem Heredoc.
- Fügen Sie in Text in doppelten Anführungszeichen an strategischen Stellen die Zeichenfolge `\n` ein.

```
echo "...Schluss.</p>\n<p>Neu ...";
```

Wozu Funktionen?

- Code wird wiederverwendbar.
- Code wird in überschaubare Einheiten zerlegt.
- Kleine Einheiten können einfach auf Korrektheit geprüft werden.
- Änderungen, die fachlich *einen* Bereich betreffen, müssen im Code auch nur an *einer* Stelle gemacht werden.

Leitsätze

- Don’t repeat yourself! (DRY)
- Divide et impera (divide and conquer)

Funktionen definieren

Funktionsdefinition

- Schlüsselwort `function`, dann ein frei gewählter Funktionsname und runde Klammern.
- Optional noch ein Doppelpunkt und der Datentyp des Rückgabewertes.
- In den runden Klammern stehen Funktionsargumente.
- Der Code wird in geschweifte Klammern eingeschlossen.

```
function rechneBmi($groesseCm, $gewichtkg) : int
{
...
}
```

Funktionen dokumentieren

- Wählen Sie einen aussagekräftigen Funktionsnamen.
- Verwenden Sie camelCase bei langen Funktionsnamen.
- Verwenden Sie strukturierte Kommentare (DocComments).

```
/**
 * @param int $groesseCm
 * @param int $gewichtkg
 * @return int
 */
function rechneBmi($groesseCm, $gewichtkg) : int
{
  $bmi = $gewichtKg / $groesseCm / $groesseCm * 10000;
  return $bmi;
}
```

Funktionsparameter

- Man kann Datentypen mit angeben:
  `function druckeKopf(string $titel)`
- Man kann Standardwerte vorgeben:
  `function rechneBrutto($netto, $steuersatz=19.0)`
- Man kann variable Parameterzahlen annehmen:
  `function rechneDurchschnitt(...$noten)`

Parameter-Übergabe

Verfahren	*Call by Value*	*Call by Reference*
Verwendung	Standard	Aktivieren mit &
Funktion erhält	Werte der Parameter	Zugriff auf Parameter
Übergebene Variablen	bleiben gleich	ändern sich dauerhaft

```
function demo(&$aendert, $bleibt)
{
  $bleibt++; $aendert++;
}
$a = 1; $b = 1;
demo($a, $b);
// a ist jetzt 2, b ist noch 1
```

Kontrollfluss in Funktionen

- Das Schlüsselwort `return` beendet einen Funktionsaufruf und liefert auf Wunsch einen Rückgabewert zurück.
- Wenn `return` fehlt, endet die Funktion in ihrer letzten Zeile und liefert nichts zurück (`void`).
- In Fehlersituationen kann eine Funktion auch mit `throw` abbrechen.
- Verwenden Sie sinnvolle Rückgabewerte.

Rekursion

- Rekursion ist, wenn eine Funktion sich selbst aufruft.
- Rekursion braucht klare Abbruchbedingungen, damit nicht unendlich viel Speicher und Rechenzeit verbraucht wird.
- Rekursive Algorithmen lasse sich immer auch nicht rekursiv formulieren. Unfaires Beispiel:

```php
function countdown($zahl)
{
  if($zahl > 0) {
    echo "$zahl ";
    countdown($zahl - 1);
  }
  echo "Bum!";
}
```

```php
function countdown($zahl)
{
  while($zahl > 0) {
    echo "$zahl ";
    $zahl = $zahl - 1;
  }
  echo "Bum!";
}
```

PHP-Datumsfunktionen

- PHP kennt klassische prozedurale Datumsfunktionen und moderne Datums-Objekte.
- Nur die modernen Funktionen beherrschen Millisekunden.
- Die modernen Objekte beherrschen Datumswerte bis weit in die Vergangenheit und Zukunft.
- Datumsobjekte ähneln denen in Java und C++.

Beispiel (objektorientiert)

- Datumsobjekt erzeugen:
```
$termin = new DateTime('now');
$silvester = new DateTime('2021-12-31');
```
- Datum ausgeben:
```
echo $termin->format('d.m.Y');
```
- Zeitintervall definieren:
```
$einTag = new DateInterval('P1D');
```
- und verwenden:
```
$termin->add($einTag);
// $termin ist nun morgen um die selbe Zeit
```

Locale

Wie Sie wissen, werden Datumswerte, Zahlenwerte und Geldbeträge abhängig von Sprache und Land formatiert. Es gibt in PHP Formatier-Objekte, die Ihnen die Arbeit weitgehend abnehmen:

- IntlDateFormatter für Datumswerte
- NumberFormatter für Zahlen und Geld

Das Locale kann zum Beispiel aus dem HTTP-Header ausgelesen werden.

NumberFormatter

NumberFormatter erzeugen

- Für Zahlen:
  `$zahlenformat = new NumberFormatter('de_DE', NumberFormatter::DECIMAL);`
- Für Geld:
  Verwenden Sie `NumberFormatter::CURRENCY);`

NumberFormatter nutzen

Ausgabe erzeugen

```
$anzeige = $zahlenformat->format(123.45);
```

Kommazahl einlesen

```
$zahl = $zahlenformat->parse('123,45');
```

Chrome und Firefox ersetzen bei Zahlen aus `<input>`-Feldern vom Typ `number` das Komma durch einen Punkt, damit ist der Aufwand beim Lesen von Zahlen oft nicht nötig.

NumberFormatter tunen

- Einstellungen abfragen mit `getAttribute()`
- Einstellungen setzen mit `setAttribute()`

```
// Immer zwei Nachkommastellen:
$zahlenformat->setAttribute(
  NumberFormatter::FRACTION_DIGITS, 2);
```

Entscheidungen mit `if()`

- Nach dem Schlüsselwort `if` folgt die Bedingung in runden Klammern.
- Der *Dann*-Teil folgt in geschweiften Klammern.
- Wenn es einen *Sonst*-Teil gibt, wird er nach dem Schlüsselwort `else` in geschweiften Klammern angehängt.

```
if(/* Bedingung */) {
  // Code für den
  // Dann-Teil
}
```

```
if(/* Bedingung */) {
  // Code für den
  // Dann-Teil
}else {
  // Code für den
  // Sonst-Teil
}
```

Kombiniertes `if()`

- Verschachteltes `if` ist möglich.
- Kombination von Bedingungen mit `or` oder `and` ist besser, falls möglich.[1]

```
if(/* ist Lehrer */) {
  // Aufzugsschlüssel
}else {
  if(/* gehbehindert */) {
    // Aufzugsschlüssel
  } else {
    // kein Schlüssel
  }
}
```

```
if(/* ist Lehrer */
      or
      /* gehbehindert */)
{
  // Aufzugsschlüssel
}else {
  // kein Schlüssel
}
```

[1] Bei verschachteltem if() muss der Dann- oder Sonst-Teil oftmals wiederholt werden.

Auswahl aus Alternativen

If-Kaskade

```
if($var == 1) {
  // Aktion 1
} else if ($var == 2) {
  // Aktion 2
} else if ($var == 3) {
  // Aktion 3
} else {
  // Sonst-Aktion
}
```

- `switch` ist nur als Vergleich mit festen Werten möglich.

Switch-Statement

```
switch($var) {
case 1:
  // Aktion 1
  break;
case 2:
  // Aktion 2
  break;
case 3:
  // Aktion 3
  break;
default:
  // Sonst-Aktion
break;
}
```

Fehlerbehandlung mit Exceptions

Exceptions (Ausnahmen) eignen sich besonders, um aufeinander aufbauende Operationen im Fehlerfall sauber zu beenden.

- Ein kritischer Programmteil kann mit `throw` einen Fehler »werfen«.
- Dieser Programmteil wird in einem `try`-Block aufgerufen.
- Im Fehlerfall wird der `try`-Block beendet und der `catch`-Block ausgeführt.
- Sie werden Exceptions im zweiten Jahr in Java näher kennenlernen.

Sprünge mit goto

- `goto` stammt aus den Anfangszeiten der Programmierung.
- `goto` macht den Programmfluss schwer nachvollziehbar.
- Moderne Programmiersprachen haben viele Sprachkonstrukte, um auf `goto` zu verzichten:
  - Funktionen mit mehreren »Ausgängen« (`return`).
  - Schleifen mit Steuerung durch `break` und `continue`.
  - Fehlerbehandlung mit `throw` und `try/catch`.

Verwendung

- Definition eines Sprungziels: `ende:`
- Sprung zum Ziel: `goto ende;`
- `goto` kann nicht in Schleifen hineinspringen.

Klassische for-Schleife

- Festlegen eines Startwertes, einer Schleifenbedingung und eines Inkrementes.
- Festlegen einer Variablen.

```
for($i = 0; $i < 10; $i++) {
  // Aktion
}
```

- $i wird zu Anfang auf 0 gesetzt.
- Die Schleife läuft, solange $i kleiner 10 ist.
- Am Ende jedes Schleifendurchganges wird $i um 1 erhöht.

while-Schleife

- Festlegen einer Schleifenbedingung
- Im Rumpf der Schleife muss dafür gesorgt werden, dass die Schleife endet.

```
$fertig = 0;
while(! $fertig) {
  // Aktionen.
  if(/* Abbruchbedingung */) {
    $fertig = 1;
  }
}
```

Schleifen abkürzen

- Schleife komplett abbrechen mit `break;`
- Rest des Schleifendurchlaufs überspringen mit `continue;`

```
while(! $fertig) {
  /* Action */
  if(/* Bedingung 1 */) {
    continue;
  }
  /* Action */
  if(/* Bedingung 2 */) {
    break;
  }
}
```

Ein Array durchlaufen mit klassischem for()

```
$faecher = array('Deutsch', 'Englisch', 'Mathe');
```

Nur Werte

```
$laenge = count($faecher);
for($i = 0; $i < $laenge; $i++) {
  echo $faecher[$i];
}
```

`foreach()` ist in den meisten Fällen besser – siehe nächste Folie.

Ein Array durchlaufen mit foreach()

```
$vornamen = array('Schulze' => 'Fred', 'Meier' =>
'Gerd', 'Merkel' => 'Angela');
```

Nur Werte

```
foreach($vornamen as $name) {
  echo "$name\n";
}
unset($name);
```

Schlüssel und Werte

```
foreach($vornamen as $nn => $vn) {
  echo "$vn heißt mit Nachnamen $nn\n";
}
unset($vn);
```

Klassen und Objekte

- Wenn zusammengehörende Daten zusammen gehalten werden, entstehen bessere Programme.
- Eine *Klasse* bildet den Bauplan für die einzelnen *Objekte*.
- Datenfelder in Objekten heißen *Eigenschaften*.
- Funktionen können als *Methoden* in eine Klasse integriert werden.
- Inhalte einer Klasse können Zugriffsbeschränkungen erhalten.
- Einführungstexte zu Klassen verwenden eher Java als PHP.

Zugriffsrechte vergeben

public	Voller Zugriff
protected	Zugriff nur für Klassenmitglieder und abgeleitete Objekte
private	Zugriff nur für Klassenmitglieder

Für Eigenschaften ohne volle Zugriffsrechte schreibt man bei Bedarf `get()`- und `set()`-Methoden.

Konstanten

- Konstanten sind für eine Klasse einheitlich definiert.
- Sie können auch ohne ein konkretes Objekt verwendet werden.
- Konstantennamen sollten groß geschrieben werden.
- Zugriff erfolgt (ohne $-Zeichen) über ::-Schreibweise.

```
class Waehrungsrechner {
  const EURO_DM = 1.95583;
  ...
  // Zugriff aus der eigenen Klasse
  $euros = $dm / self::EURO_DM;
}
// Zugriff aus anderen Klassen
$euros = $dm / Waehrungsrechner::EURO_DM;
```

Statische Eigenschaften

- Statische Eigenschaften gelten für die ganze Klasse, nicht für ein einzelnes Objekt.
- Beispielsweise kann die Anzahl der Objekt einer Klasse gezählt werden.

```
class Produkt {
  static int $maxId = 123;
  public function neuesProdukt() {
    $self::maxId++;
...
```

Statische Methoden

- statische Methoden können aufgerufen werden, ohne dass ein Objekt erzeugt wurde.

```
class Waehrungsrechner {
  const EURO_DM = 1.95583;
  static function rechneDMinEuro (float $dm) {
    ...
  }
}
// Aufruf von außen
$euros =
Waehrungsrechner::rechneDMinEuro($maerker);
```

Klassen definieren

```
class Kunde
{
  public string $nachname;
  private int $kundennummer;
  public function __construct($name, $nummer)
  {
    $this->nachname = $name;
    $this->kundennummer = $nummer;
  }
  public function getKundennummer()
  {
    return $this->kundennummer;
  }
  ...
```

Objekte erzeugen

- Definieren Sie zuerst eine Klasse.
- Objekte werden mit `new()` erzeugt.
- Wenn ein Objekt initialisiert werden muss, braucht die Klasse eine Methode `__construct()`.
- Objekte können mit `unset()` gelöscht werden.

Klassen benutzen

Benutzung

```
$meinkunde = new Kunde('Maier', 313);
echo 'Kunde Nummer '
  . $meinkunde->getKundennummer();
echo 'heißt ' . $meinkunde->nachname;
$kunden = array();
$kunden[$meinkunde->getKundennummer()] = $meinkunde;
echo 'Kunde 313 heißt ' . $kunden[313]->nachname;
```

Beispiel: Warenkorb

Konventionell

- Definieren von Anzahl, Artikelnummer, Name und Preis als unabhängige Arrays.
- Verbindung über gleichen Array-Index.

Objektorientiert

- Definition einer Bestellposition, die aus Anzahl, Artikelnummer, Name und Preis besteht.
- Definition des Warenkorbs als Array aus Bestellpositionen.

Vererbung

- In der objektorientierten Programmierung wird Vererbung zur Spezialisierung verwendet.
- Vererbung wird mit dem Schlüsselwort `extends` gekennzeichnet.

```
class Schueler{ ...}
class Berufsschueler extends Schueler { ...}
// Berufsschüler haben zusätzlich einen
Ausbildungsbetrieb
```

Traits und Interfaces

- Eine Klasse kann nur von *einer* Elternklasse erben.
- Ein *Interface* benennt einen Satz Methoden, den eine Klasse umsetzen soll.
- Ein *Trait* enthält zusätzlich eine Implementierung der benannten Methoden.
- Interfaces werden mit `implements` akzeptiert, Traits mit `use` importiert.

Sitzungs-Management

Das Problem

Wie kommen Daten von einer Seite auf eine andere? HTTP ist ein zustandsloses Protokoll. Der Webserver weiß nicht,

- wer die Seite abruft und
- welche Seite vorher abgerufen wurde.

Im Grunde muss nur eine Sitzungs-ID übergeben werden, alles weitere kann im Server gespeichert sein.

Lösungsmöglichkeiten

- Formulardaten (GET/POST)
- GET-Parameter in der URL
- Cookies

Cookies

- Cookies sind Daten geringen Umfangs, die von einem Webserver im Browser des Benutzers abgelegt werden.
- Cookies können im Browser nachgesehen und gelöscht werden.
- First-Party-Cookies können nur von der Website, die sie gesetzt hat, ausgelesen werden.
- Third-Party-Cookies dienen dem Site-übergreifenden Tracking.
  Sie werden aus Datenschutzgründen abgeschafft.

Cookie-Arten

Session- Diese Cookies werden gelöscht, wenn das Browserfenster geschlossen wird. Hauptzweck: Sitzungs-Management

dauerhaft Cookies bleiben für eine einstellbare Zeit gesetzt. Hauptzweck: Benutzer wiedererkennen, Website anpassen.

Third Party Hauptzweck: Werbenetzwerke

Cookie-Sicherheit

Cookies liegen auf dem Client. Dadurch besteht prinzipiell ein Sicherheitsrisiko.

- Der Benutzer kann Cookies jederzeit löschen und mit etwas Energie auch ändern.
- Mit Cross-Site-Scripting-Attacken können Cookies gestohlen werden.

Cookies auslesen und setzen

- Cookies der Website liegen im Superglobal-Array `$_COOKIE[]`
- Cookies setzen mit `setcookie(`*`Name`*`, `*`Wert`*`, `*`Ablauf`*`)`.
- *Ablauf* ist ein Unix-Timestamp (Anzahl Sekunden seit 1.1.1970).
- Wenn *Ablauf* nicht angegeben oder auf 0 gesetzt wird, entsteht ein Sitzungscookie.
- Für längere Lebensdauer muss *Ablauf* auf ein Datum in der Zukunft gesetzt werden.

Cookie-Beispiel

```
$datumsobjekt = new DateTime();
$heute = $datumsobjekt->getTimestamp();
$naechstesJahr = $heute + 365 * 24 * 60 * 60;
// Cookies setzen
setcookie('Name', $benutzer, 0); // Session
setcookie('Stil', 'dunkel', $naechstesJahr);
// ...und auslesen,
// z.B. aus einer anderen Seite der selben Site
$benutzer = $_COOKIE['Name'];
echo "Hallo $benutzer";
```

Sessions

- Sessions bieten ein bequemes Sitzungs-Management, man muss sich die Sitzung nicht mit Cookies selbst »basteln«.
- Eine Session setzt ein einziges Cookie.
  Falls Cookies gesperrt sind, wird GET in der URL verwendet (unsicher!).
- Sitzungsdaten liegen auf dem Server, nicht beim Client.
- Session starten mit `session_start();`
- Session beenden mit `session_destroy();`
- Sitzungsdaten abrufen und setzen mit dem Superglobal `$_SESSION[]`

Sessions benutzen

- Konfigurieren Sie Session-Sicherheit einmalig in der php.ini.
- Jede Seite, die eine Session benutzt, muss als erstes `session_start();` verwenden.
- Sitzungsdaten abrufen und setzen mit dem Superglobal `$_SESSION[]`
- Am Schluss Sitzungsdaten löschen mit `$_SESSION[] = array ();`
- Dann die Session beenden mit `session_destroy();`

Session benutzen

Seite 1: Variable belegen

```
session_start();
$_SESSION['name'] = $benutzer;
```

Seite 2: Variable abfragen

```
session_start();
if(!empty($_SESSION['name']) {
    $benutzer = $_SESSION['name'];
    echo "Hallo $benutzer!";
}else {
    echo "Sie sind nicht angemeldet!";
}
```

Session beenden

Seite 3: Session beenden

```
session_start();
$_SESSION = array (); // Zuerst Daten löschen
// Dann den Cookie
$params = session_get_cookie_params();
setcookie(session_name(), null, time() - 84000,
  $params['path'], $params['domain'],
  $params['secure'], $params['httponly']);
// Und aufräumen.
session_destroy();
echo '<p>Sie sind ausgeloggt.</p>';
```

Daten speichern

Wenn Benutzerdaten auf dem Server gespeichert werden sollen, kommt folgendes in Frage:

Dateien für große Objekte wie z.B. Bilder

Datenbank für strukturierte Daten

PHP und Datenbanken

- PHP arbeitet mit vielen Datenbanken zusammen.
- Es gibt universelle Datenbankschnittstellen und Schnittstellen für einzelne Datenbanken.
- Wir verwenden im Unterricht die Schnittstelle mysqli und die Datenbank MariaDB.

Datenbank-Sitzungen

Datenbankzugriffe in PHP laufen immer so ab:

- Verbindung zur Datenbank aufbauen.
- SQL-Befehle absetzen.
- Wenn Transaktionen verwendet werden: Commit oder Rollback nicht vergessen.
- Verbindung trennen.

Datenbank-Sitzung aufbauen

Sitzung aufbauen

```
$verbindung = new mysqli('localhost', $benutzer,
$passwort, $datenbank);
if ($verbindung->connect_errno) {
  // Datenbankverbindung ist gescheitert
  // Skript abbrechen
}
```

In der Entwicklungsphase sollten Sie vor dem Verbinden Fehlerbehandlung aktivieren:

```
mysqli_report(MYSQLI_REPORT_ERROR
  | MYSQLI_REPORT_STRICT);
```

Zugangsdaten: Problem

- Zugangsdaten dürfen niemals unterhalb des `htdocs`-Ordners liegen: Sie könnten ausgelesen werden.
- Zugangsdaten sollen nicht in der `php.ini` liegen: Jeder PHP-Benutzer kann sie auslesen.
- Datenbank-Zugangsdaten kann man meistens nicht benutzerbezogen vergeben: Datenbankbenutzer und Website-Benutzer sind zwei getrennte Konzepte.

Zugangsdaten: Lösungsvorschlag

- Speichern Sie Zugangsdaten außerhalb von `htdocs` in einer Datei.
- Verwenden Sie ein bequem handhabbares Dateiformat.
- Setzen Sie restriktive Zugriffsrechte auf diese Datei.
- Verschlüsseln Sie den Inhalt, falls möglich.

Zugangsdaten: Beispiel

Konfigurationsdatei zugang.json in /xampp/bkwi

```
{
    "Computer": "localhost",
    "Datenbank": "pizzabase",
    "Benutzer": "pizzaservice",
    "Passwort": "pizza99"
}
```

Zugriff

```
$datei = file_get_contents('/xampp/bkwi/zugang.json');
$konfiguration = json_decode($datei);
$dbVerbindung = new mysqli(
    $konfiguration->Computer, $konfiguration->Benutzer,
    $konfiguration->Passwort, $konfiguration->Datenbank);
```

Datenbank-Sitzung beenden

Sitzung beenden

```
myscli::close();
```

Falls Sie Transaktionen verwenden, sollten Sie vorher `commit` machen.

SQL-Statements

Arten von Statements

Abfragen liefern eine Ergebnismenge
SELECT

Befehle liefern Erfolg oder Scheitern
UPDATE, INSERT, DELETE, CREATE ...

Parameter einbinden

Abfragen müssen typischerweise konkretisiert werden.

- Text zusammensetzen mit . und Variablen-Interpolation
```
$befehl = "SELECT Password FROM Users
  WHERE Name = '$name'";
```
- Verwenden von *Prepared Statements* mit Platzhalter ? und separater Parameter-Übergabe.
```
$befehl = "SELECT Password FROM Users
  WHERE Name = ?";
```

Wenn Benutzereingaben direkt in ein Statement eingebaut werden, ergibt sich ein großes Sicherheitsrisiko (SQL-Injection).

Prepared Statements

- An der Stelle, die angepasst werden muss, steht ein Fragezeichen (ohne Anführungszeichen).
- Das Fragezeichen ist nur Platzhalter für Werte, nicht für Datenbankobjekte oder Schlüsselwörter.
- Die Zuweisung von Daten erfolgt mit `bind_param()`.
- Datenbanken sind streng typisiert, Datentypen müssen angegeben werden.
- Zuerst wird `prepare()` aufgerufen, dann `bind_param()` und schließlich `execute()`.
- Man kann den gebundenen Variablen mehrmals Werte zuweisen und dann mehrmals `execute()` aufrufen.

Prepared Insert

```
$befehl = 'INSERT INTO Benutzer (ID, Name) '
  . ' VALUES (?, ?)';
$statement = $verbindung->prepare($befehl);
$nutzerid = 23; $nutzername='Bernd';
// i steht für Integer, s für String
$statement->bind_param('is', $nutzerid, $nutzername);
// Bernd anlegen
$statement->execute();
$nutzerid = 24; $nutzername='Ulrike';
// Ulrike anlegen
$statement->execute();
// vielleicht noch $verbindung->commit();
```

Abfragen absetzen

Direkte Abfrage

```
$ergebnis = $verbindung->query('SELECT ID, Name '
  . "FROM Benutzer WHERE Name LIKE 'B%'");
```

Prepared Query

```
$statement = $verbindung->prepare('SELECT ID, Name '
  . ' FROM Benutzer WHERE Name like ?');
$muster = 'B%';
$statement->bind_param('s', $muster);
$statement->execute();
$ergebnis = $statement->get_result();
```

Ergebnisse holen: Methoden

Ergebnisse können auf mehrere Arten geholt werden:

- Als numerisches Array
- Als assoziatives Array
- Als Objekt
- In vorher festgelegte Variablen

In jedem Fall erhalten Sie genau eine Ergebniszeile.
Weitere Zeilen werden mit einer Schleife geholt.
Wenn Sie genug haben, müssen Sie das Ergebnis schließen.

Ergebnisse holen: Methode wählen

- Wenn die Ergebnisfelder beim Programmieren bereits feststehen:
  Holen Sie Daten in vorher festgelegte Variablen.
- Wenn Sie die Feldnamen nicht benötigen, verwenden Sie ein numerisches Array.
- Sonst verwenden Sie eine der zwei anderen Methoden nach Ihren persönlichen Vorlieben.

Daten holen: Pseudocode

1. $statement = $verbindung->prepare(*SQL-Befehl*);
2. Falls ? im Befehl: $statement->bind_param(*Parameter*);
3. $statement->execute();
4. $ergebnis = $statement->get_result();
5. Schleife: while($ergebnis->*Daten holen*) {
6. in der Schleife: *Daten anzeigen oder bearbeiten*
7. }
8. $ergebnis->close();

Ergebnisse holen: Einfaches Array

Die Abfrage war 'SELECT ID, NAME FROM Benutzer'.

Vorab bekannte Spalten

```
while($benutzerdaten = $ergebnis->fetch_row()) {
  echo 'Benutzer mit ID ' . $benutzerdaten[0]
  . ' heißt ' . $benutzerdaten[1] . "\n";
}
$ergebnis->close();
$statement->close();
```

Ergebnisse holen: Assoziatives Array

Unbekannte Spalten

```
while($benutzerdaten = $ergebnis->fetch_assoc())
{
  foreach($benutzerdaten as $spalte => $wert) {
    echo "Feld $spalte hat Inhalt $wert.\n";
  }
}
$ergebnis->close();
$statement->close();
```

Ergebnisse holen: Benannte Variablen

Die Abfrage war 'SELECT ID, NAME FROM Benutzer'.

Vorab bekannte Spalten

```
$stmt = $verbindung->prepare(
  'SELECT ID, NAME FROM Benutzer');
$stmt->execute();
$stmt->bind_result($id, $name);
while($stmt->fetch()) {
  echo "Benutzer mit ID $id heißt $name\n";
}
$stmt->close();
```

mysqli: Fehler behandeln (I)

- Alle Methoden liefern null zurück, wenn sie scheitern.
- Ausführliche Fehlerabfrage:

```
$stment = $mysqli->prepare(...);
if($stment != null) {
  // statement ist gültig
  ...
}else {
  // Fehlerbehandlung
}
```

mysqli: Fehler behandeln (II)

- Halbkurzfassung: Zuweisung und Vergleich mit null werden in einer Zeile durchgeführt.
```
if(($stment = $mysqli->prepare(...)) != null)
{
```
- Kurzfassung: Wie Halbkurzfassung, aber der Vergleich mit null ist implizit.
```
if($stment = $mysqli->prepare(...)) {
```

Die Kurzfassung sehen Sie am häufigsten. Das Gleichheitszeichen ist kein Vergleich, sondern eine Zuweisung. Das Ergebnis der Zuweisung wird dann mit null verglichen.

mysqli: Fehler behandeln (III)

Mehrfache Fehlerabfragen ergeben unschöne Verschachtelungen.

```
if($stment = $mysqli->prepare(...)) {
  if($stment->execute()) {
    if($stment->bind_result(...)) {
      if($stment->fetch()) {
        // Daten verarbeiten
      } else { // Fetch-Fehler
      }
    } else { // BindResult-Fehler
    }
  } else { // execute-Fehler
  }
} else { // prepare-Fehler
}
```

mysqli: Fehler behandeln (IV)

Alternativ können die Statements mit `and` in einem einzigen `if` verbunden werden.

```
if($stment = $mysqli->prepare(...)
    and $stment->execute()
    and $stment->bind_result(...)) {
  while ($stment->fetch()) {
    // Daten bearbeiten
  }
} else {
  // ungenaue Fehlermeldung ausgeben.
}
```

mysqli: Fehler behandeln (V)

- Die eleganteste Möglichkeit ist die Verwendung von `try`, `catch` und `throw`.
- Dieses Verfahren wird auch in anderen Sprachen, z.B. Java und C++, verwendet.
- Wir haben leider nicht genug Zeit, um uns das anzusehen.