# Hochschule Darmstadt
# Fachbereich Informatik

## Entwicklung webbasierter Anwendungen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Hochschule Darmstadt
# Fachbereich Informatik

## 1. Einleitung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Zielsetzung

- aus der Modulbeschreibung:
  - ⇨ Die Studierenden sollen
    - Aktuelle Auszeichnungssprachen kennen und anwenden
    - Skriptsprachen für client- und serverseitige Webprogrammierung anwenden
    - ein Dokument Objekt Modell verstehen
    - die Architektur webbasierter Client/Server-Anwendungen mit Datenbankanbindung verstehen
    - Methoden und Techniken zur Entwicklung webbasierter Anwendung
    - Sicherheitsaspekte im Kontext von Webanwendungen verstehen
  - ⇨ Konkret: Nach der Veranstaltung...
    - kennen Sie den Sinn, Zweck und die Grenzen der verschiedenen Techniken
    - verstehen Sie das Zusammenspiel der verschiedenen Techniken
    - kennen Sie die wesentlichen Standards
    - sind Sie in der Lage, komplexe und standardkonforme Webseiten zu erstellen
    - haben Sie die Grundlagen, um sich in diverse andere Web-Techniken einzuarbeiten

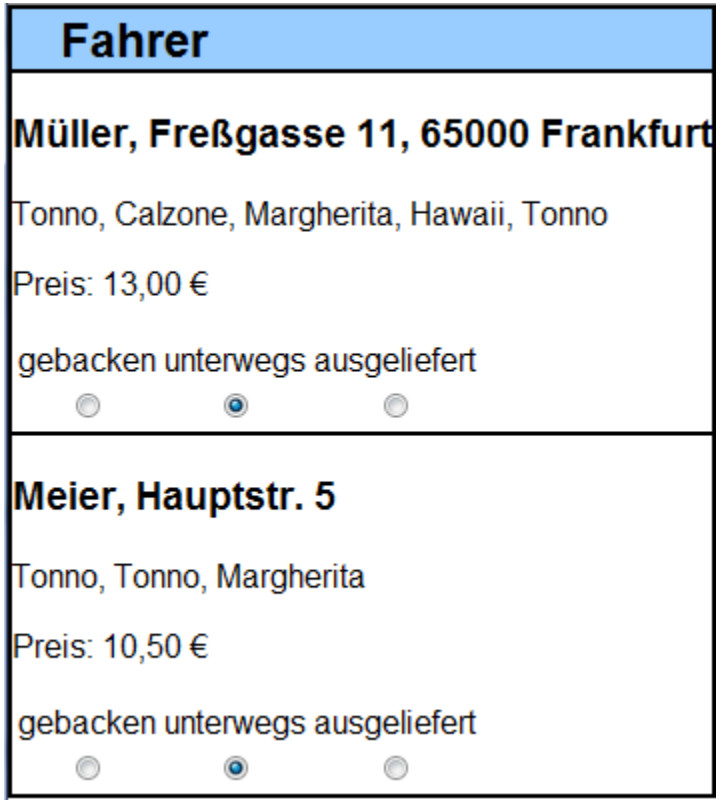
# Konkrete Inhalte des Veranstaltung

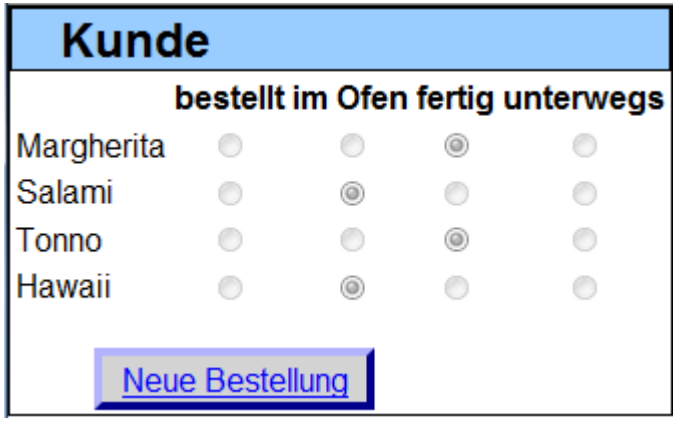
- Entwurf
- HTML Grundlagen
- Formulare und Layout
- CSS und dynamisches Layout
- ECMAScript, DOM, AJAX
- Webserver Konfiguration (Apache), CGI
- Objektorientiertes PHP, MVC Framework
- PHP mit Datenbankanbindung (MySQLi)
- HTTP
- Sicherheit
- Professionelle Webentwicklung
  (Entwicklung, Test, Web-Projektverwaltung uvm.)

Die verschiedenen Themen werden nicht vollständig behandelt – es geht in EWA "nur" um die Grundlagen !

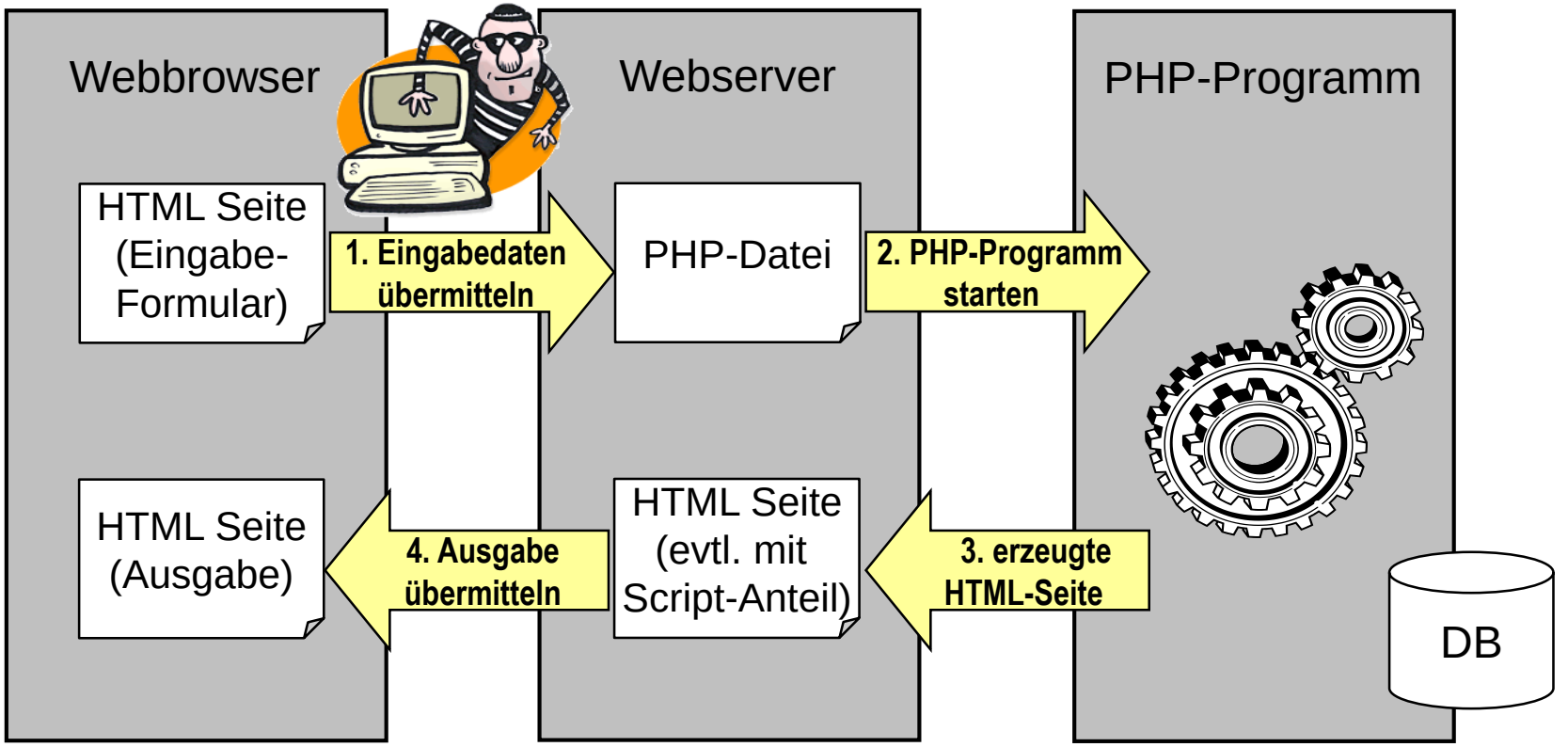
# Aufgabe im Praktikum: Pizzaservice





# Einsatz der Technologien im Zusammenhang




- HTML
- CSS
- ECMA-Script
- DOM
- AJAX
- HTTP
- Server-Konfiguration
- CGI
- PHP
- MySQL

# Webquellen und Software

- Webquellen
  - ⇨ HTML/CSS:
    - webkompetenz.wikidot.com/docs:html-handbuch
    - www.w3schools.com
  - ⇨ PHP
    - www.selfphp.info
    - www.tutorialspoint.com/php7
  - ⇨ JavaScript
    - developer.mozilla.org/en/JavaScript/Guide
  - ⇨ Online-Bücher www.oreilly.de/openbooks.php
    - K. Janowicz: "Sicherheit im Internet"
    - S. Kersken: "Praktischer Einstieg in MySQL mit PHP"
    - uvm.
- Standards
  - ⇨ HTML-, CSS-, DOM-Standard und HTML/CSS-Validator
    - w3.org/
    - validator.w3.org/
  - ⇨ ECMAScript (ECMA-262)
    - www.ecma-international.org/
- Freie Software, Dokus, Tutorials
  - ⇨ Tutorials und Referenzen
    - w3schools.com
  - ⇨ HTML Editor + Validator
    - http://www.phase5.info/
    - http://notepad-plus-plus.org/
  - ⇨ XAMPP (Webserver, MySQL, PHP)
    - apachefriends.org/de/xampp.html

# Hochschule Darmstadt
# Fachbereich Informatik

## 1.1 Softwaretechnik für webbasierte Anwendungen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Motivation

Das Thema kommt im Kapitel "Professionelle Webentwicklung" später noch mal ausführlicher!

- auch webbasierte Anwendungen sind Softwaresysteme !
  - ⇨ es gilt weiterhin alles, was man über Softwaretechnik, Software Ergonomie und GUIs gelernt hat
  - ⇨ nicht vor lauter
    *Design* ⇨ Grafik, Animation, Gimmicks
    den
    Entwurf ⇨ Analyse, Architektur, Datenorganisation
    vergessen !
- Die Programmiersprachen, -umgebungen und die Aufgaben verleiten oft zum Hacken !
- Ein Konzept hat aber noch nie geschadet..... und ein bewusstes Vorgehen auch nicht !

# Anforderungsanalyse: Funktionale Anforderungen

genau wie in der Vorlesung *"Nutzerzentrierte Softwareentwicklung"*

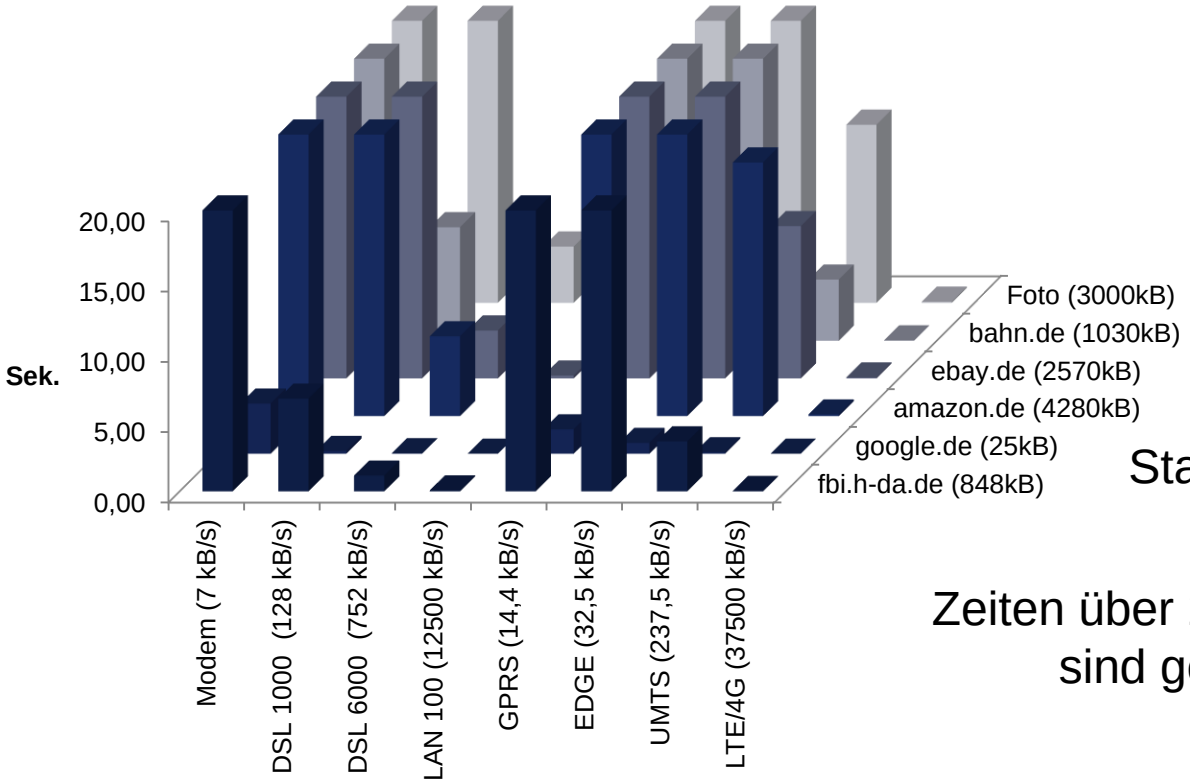
- Zweck des Produkts bestimmen
  - ⇨ Was wollen Benutzer mit der Anwendung erreichen? "sich informieren" ist zu wenig!
  - ⇨ Produktkatalog, Selbstlernmedium, Spiel, Werbung,...?
  - ⇨ Fülle von Informationen darstellen und dennoch leichte Orientierung?
- Ermittlung der Zielgruppe
  - ⇨ Alter, Geschlecht, Sprache, Ausstattung, Ausbildung, PC-Erfahrung, Internet-Zugang, Benutzungsfrequenz
- Verkaufsargument
  - ⇨ Nutzen, Mehrwert ggü. Konkurrenz (z.B. Printmedien)
- Erscheinungsbild
  - ⇨ reine Information oder auch Darstellung?
- Zielmedium
  - ⇨ CD, Internet, Intranet, Smartphone, Tablet, Fernseher (Set-Top-Box)

# Anforderungsanalyse: nicht-funktionale Anforderungen

- Es gibt im Internet ein riesiges Angebot
  - meine Anwendung ist nur eine unter vielen
  - Benutzer wechseln häufig die Anwendungen / Sites
- Benutzer scannen statt zu lesen
  - 79% überfliegen die Seiten nur
  - Schulung darf absolut nicht erforderlich sein
  - Hilfesystem muss überflüssig sein
- Viele unerfahrene Benutzer
  - Kinder, Senioren, Couch Potatoes
- Unterschiedliche Systeme der Benutzer
  - Browser, Plugins, CPU, Bildschirme, Datenverbindung
- Performanz
  - Support für verschiedene Browser, Ausgabegeräte, Transferraten,
  - Anzahl der Benutzer, Häufigkeit des Datenaustauschs,…
- Darstellung
  - Stil, Corporate Identity, Farbschema

# Nicht-funktionale Anforderungen: Problem 1: Transferraten

Auswirkung der Bandbreite auf Ladezeiten




Stand 03/2017

Zeiten über 20 Sekunden sind gekappt!

⇨ Man muss (besonders für mobile Anwendungen) immer noch mit Datenvolumen geizen!

# Nicht-funktionale Anforderungen: Problem 2: Plattformabhängigkeit

- "Plattform" traditionell: Betriebssystem
  - ⇨ Unix-Varianten (Workstation), Windows (PC), MacOS (Mac), MVS (Mainframe) etc.
  - ⇨ oft nur für bestimmte Hardware verfügbar
- "Plattform" im Web: Browser + Version
  - ⇨ Internet Explorer, Firefox, Safari, Chrome etc.
  - ⇨ Erhebliche Unterschiede durch
    - verzögerte oder spezielle Umsetzung der Standards
    - Plug-Ins (z.B. Flash, nicht für alle Systeme verfügbar)
    - Auflösung und spezielle Bedienelemente (z.B. Smartphone mit Touchscreen)

lediglich eine Verlagerung der Abhängigkeit

⇨ Es ist schwierig, eine Webseite für die verschiedenen Zielsysteme zu entwickeln!

# Nicht-funktionale Anf.: Problem 3: what you see is NOT what you get !

- Darstellung erfolgt über HTML
  - ⇨ HTML ist eine Auszeichnungssprache (Markup Language)
  - ⇨ WYSIWYG ist per Prinzip nicht möglich
  - ⇨ HTML Editor zeigt allenfalls ungefähr das Ergebnis
  - ⇨ sorgfältige Vorschau notwendig mit verschiedenen Browsern / Bildschirmauflösungen / Fenstergrößen

wieso hat man das Problem z.B. in PowerPoint nicht ?

- Surfer-freundliche Darstellung
  - ⇨ nicht unterstützte HTML-Anweisungen werden nicht gemeldet, sondern ignoriert
  - ⇨ Darstellung so gut es eben geht
  - ⇨ unangenehm für Webdesigner: visuelle Kontrolle erforderlich
  - ⇨ unbedingt Tool (z.B. HTML Validator) verwenden

⇨ Das Aussehen einer Webseite muss auf vielen Zielsystemen geprüft werden!

# Nicht-funktionale Anforderungen: Problem 4: Auslieferbarkeit

- Im Internet ist es üblich, dass neue Anwendungen einfach als neue Webseite "ausgeliefert" werden
  - ⇨ es gibt keine Ankündigung und keine Installationsprozedur
  - ⇨ die Anwendung muss mit diversen Browsern, Versionen und Endgeräten laufen
  - ⇨ es darf zu keinen Inkompatibilitäten mit Daten von vorhergehenden Versionen kommen
    - in den Browsern gespeicherte Daten (Cookies)
    - auf dem Webserver gespeicherte Daten (z.B. Einträge in einer Datenbank)
  - ⇨ Daten aus alten Versionen sollten aber auch nicht verloren gehen

⇨ Das Ausliefern einer Webseite ist ein kontinuierlicher Prozess, der viel Disziplin erfordert!

# Design

- Festlegungen
  - ⇨ Laufzeitumgebung
    (z.B. Datenbank, Webserver, Browser, optimale Auflösung, ...)
  - ⇨ Entwicklungsumgebung
    (z.B. Programmiersprache mit Version, Frameworks, Testtools,... )
  - ⇨ Verwendung von Standards
    (z.B. HTML5, PHP 7, …)
  - ⇨ Referenztests
    (z.B. HTML-Validator bei W3C)
  - ⇨ Konventionen für die Fehlersuche (Logging, Debugging)
  - ⇨ usw.
- Klassisches Design
  - ⇨ Definition von Klassen mit Verantwortlichkeiten
  - ⇨ Interaktion der Klassen
  - ⇨ Verteilung der Anwendung auf Webbrowser, Webserver

# Test

- Eigentlich ganz normale Tests !
  - ⇨ Funktionalität
  - ⇨ nicht-funktionale Anforderungen (Unbedingt die 4 Probleme testen!)
  - ⇨ Ausnahmefälle
  - ⇨ möglichst viel automatisiert

- Besonders wichtig
  - ⇨ Verständlichkeit und Bedienbarkeit
    - mit dem Auftraggeber bzw. künftigen Benutzern
  - ⇨ Portabilität
    - Browser, Plugins, CPU, Bildschirme, Datenverbindung
  - ⇨ Performanz
    - verschiedene Browser, Ein- und Ausgabegeräte
    - geringe Transferrate
    - hohe Benutzerlast

# Sinnvolle Arbeitsergebnisse (Meilensteine)

- Visionsdokument mit Sinn und Zweck der Website
- Übersicht der Funktionalität (z.B. Use Case Diagramm)
- Beschreibung der nicht-funktionalen Anforderungen
- Navigationsübersicht (Zustandsdiagramm)
- Screen-Diagramme (Skizzen der Seiten)
- Modellierung der SW-Struktur
  - ⇨ Aktivitätsdiagramm
  - ⇨ Klassendiagramm
  - ⇨ Sequenzdiagramm
    - Zusammenspiel von Client (HTML-Seiten) und Server (z.B. PHP)
  - ⇨ Komponentendiagramm
    - Verteilung auf Webserver, Client etc.
- Testplan

vgl. "Nutzerzentrierte Softwareentwicklung" im 3. Semester

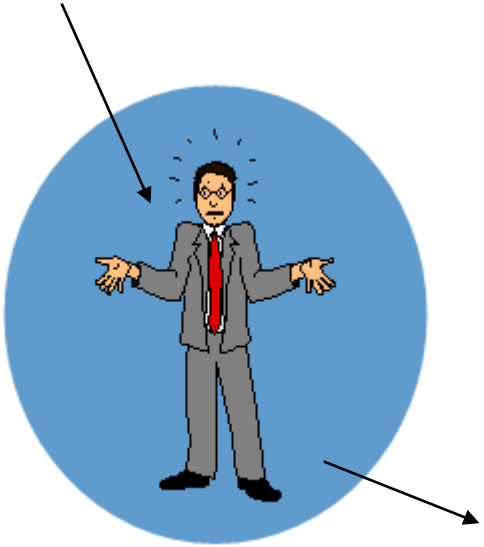
# Hochschule Darmstadt
# Fachbereich Informatik

## 1.2 Ergonomie für webbasierte Anwendungen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Mensch-Maschine-Schnittstelle

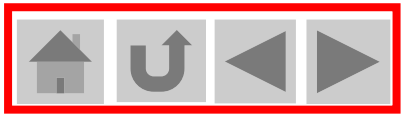
- Der Systemzustand muss auf einen Blick erkennbar sein (ohne ihn dabei zu verändern!)
  - ⇨ Grund-Forderung aus der Software-Ergonomie
- Wo bin ich ? — Ort
  - ⇨ momentaner Aufenthaltsort im System
- Was kann ich hier tun ? — Modus
  - ⇨ zur Verfügung stehende Operationen
- Wie kam ich hierher ? — Weg
  - ⇨ Vorgeschichte, Kontext
- Wohin kann ich gehen ? — Weg
  - ⇨ Ziel eines Verweises soll erkennbar sein



Es folgen Überlegungen, wie diese Fragen durch Layout- und Gestaltungselemente beantwortet werden können.

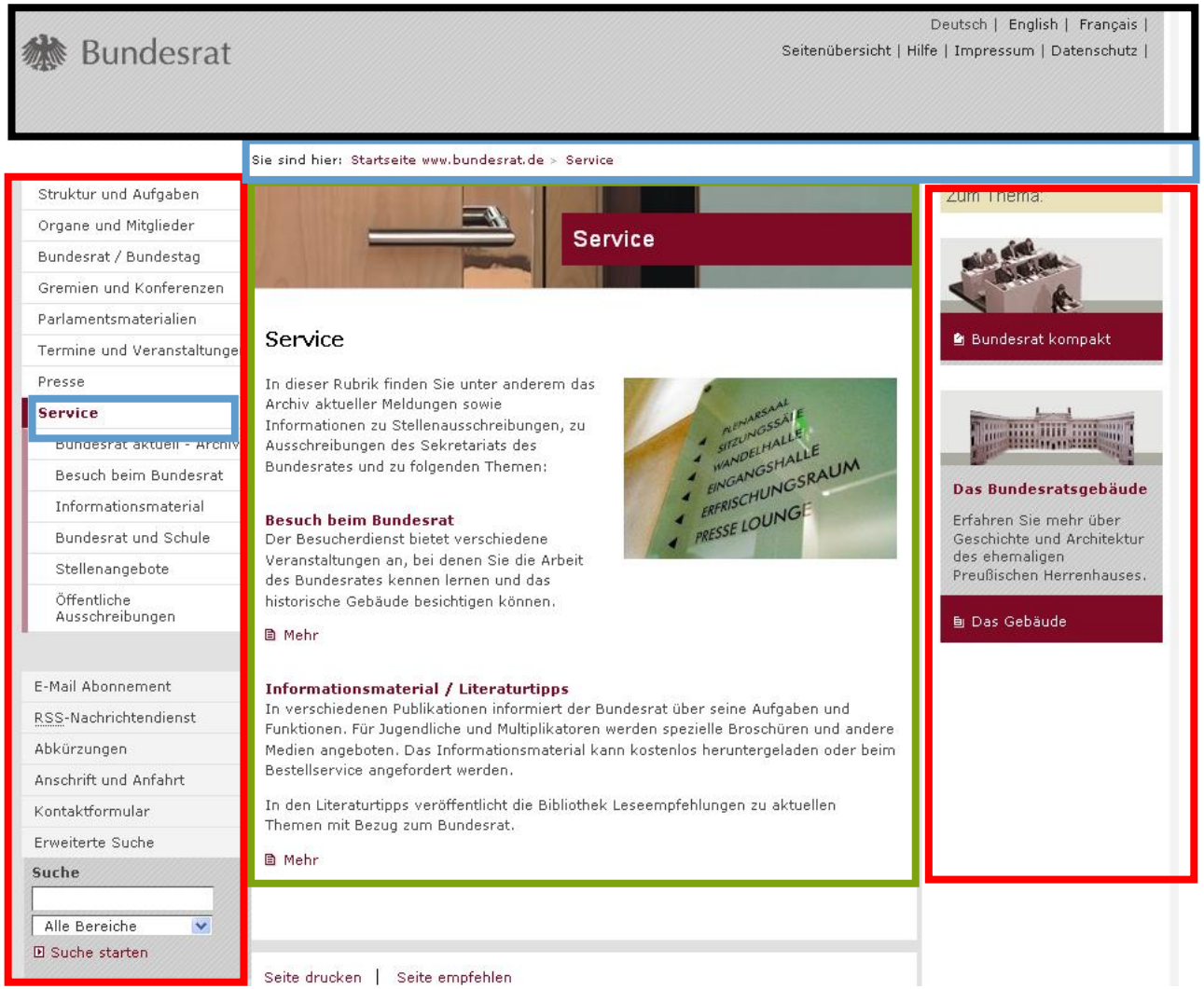
# Layout: Komponenten einer Bildschirmseite

- **Orientierungskomponente** — wo bin ich?
  - ⇨ dient der Orientierung des Benutzers
- **Präsentationskomponente** — was kann ich hier tun?
  - ⇨ Darstellung der Informationsgehalts
  - ⇨ Bühne für Animationen, Simulationen, Videos
- **Navigationskomponente (Interaktions-)** — wohin kann ich gehen?
  - ⇨ dient der Steuerung durch den Benutzer
- **Hintergrund**
  - ⇨ passives Designelement
  - ⇨ unverändert über mehrere Bildschirmseiten



# Layoutbeispiel: Bundesrat

Top-Ergebnis beim BITV Test




**Hintergrund**

**Orientierung**

**Präsentation**

**Navigation**

# Barrierefreie Webseiten

In Deutschland
gilt die BITV
(Barrierefreie Informationstechnik-Verordnung)

- Webseiten sollen auch für Menschen mit einer Behinderung zugänglich sein
  - ⇨ Sehschwächen (z.B. Rot-Grün-Blindheit) aber auch Hör-, Lern-, Lese-, motorische Schwächen  uvm.
  - ⇨ insbesondere soll eine Webseite vorgelesen werden können
    - Screenreader lesen die Seite mit Text2Speech vor
    - Braille-Zeilen geben Blindenschrift zeilenweise aus
    - Tabellen werden von links nach rechts und von oben nach unten gelesen
    - Bilder und Videos sind als solche nicht darstellbar
    - Der Inhalt muss logisch gruppiert und angeordnet sein – und nicht nach der Anordnung auf dem Bildschirm
  - ⇨ Die Bedienung muss mit vereinfachten Tastaturen möglich sein

Textausgabe mit Braille-Zeile
Bild: SBV

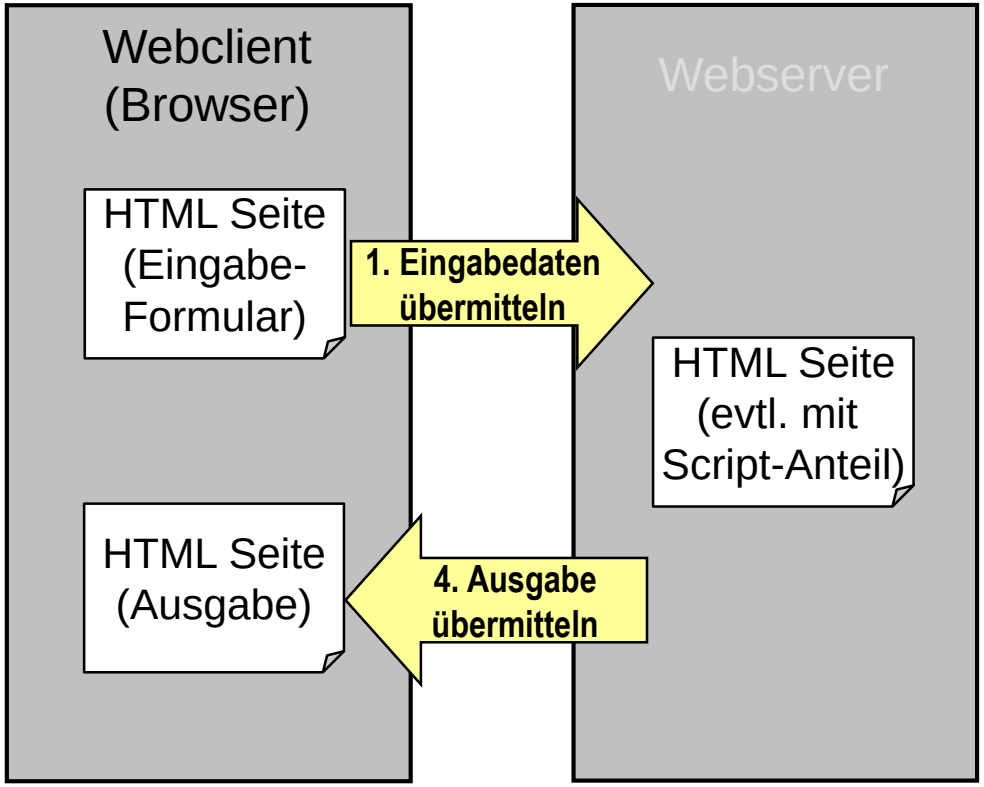
www.computer-fuer-behinderte.de

# Hochschule Darmstadt
# Fachbereich Informatik

## 2. Webclient

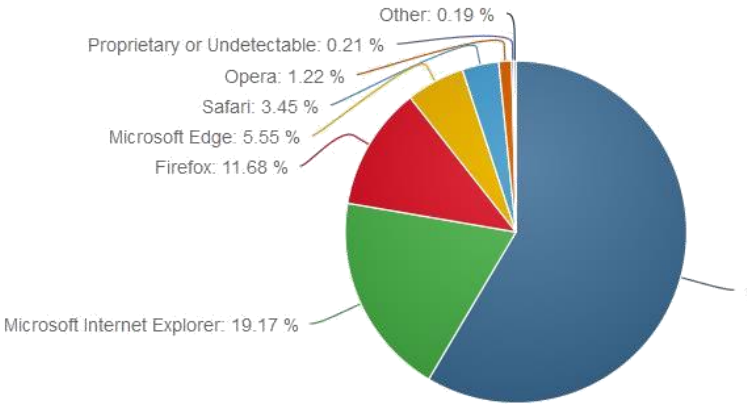
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

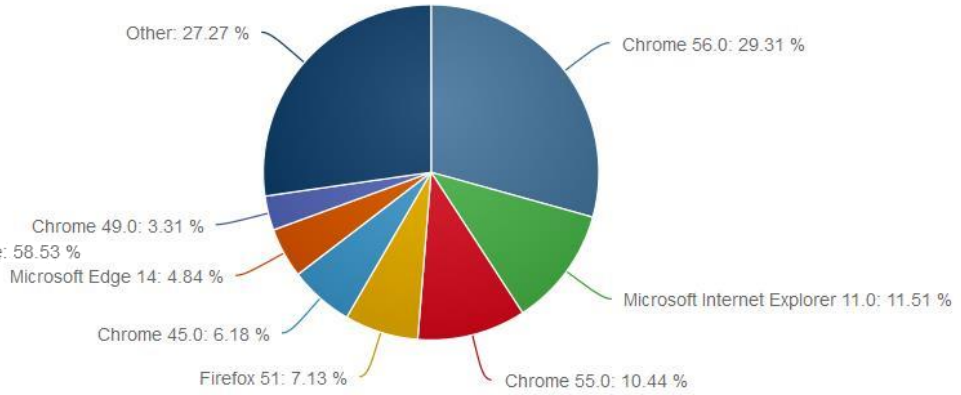
# Der Webclient

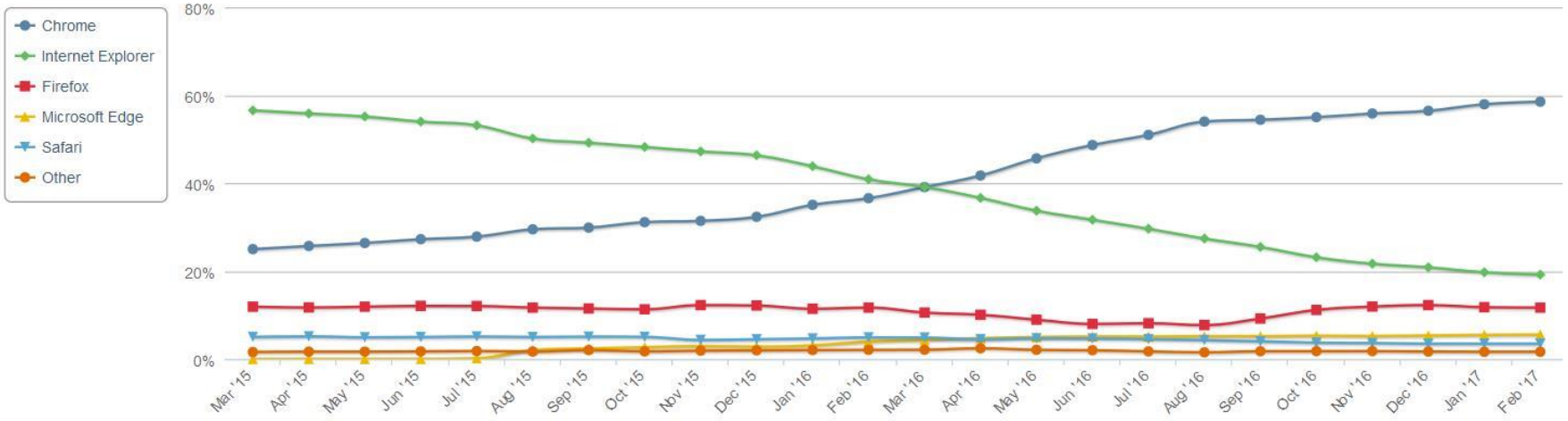



- HTML
- CSS
- ECMA-Script
- DOM
- AJAX

# Marktanteile und Trend der Browser für Desktops

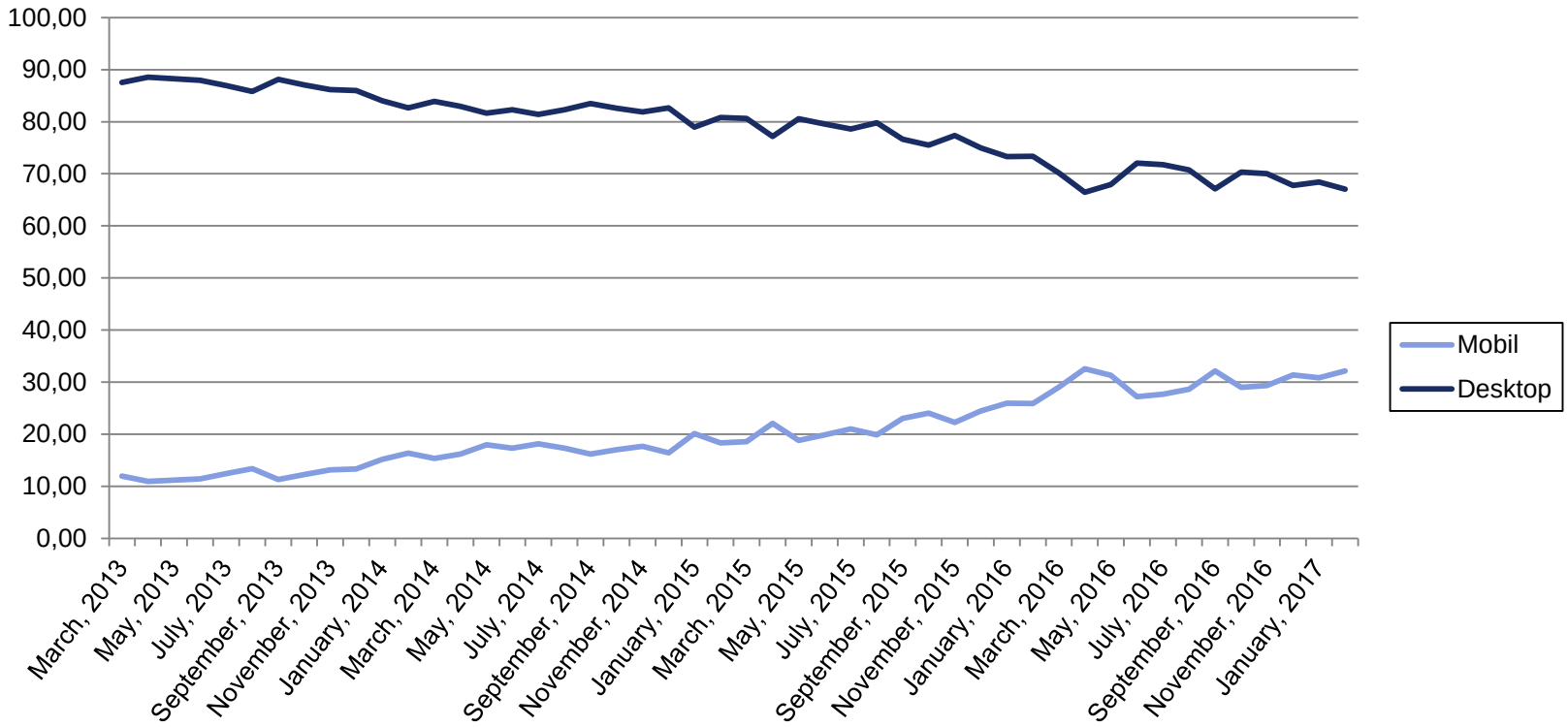



Quelle: Net Applications, http://marketshare.hitslink.com, (Stand 03/2017)

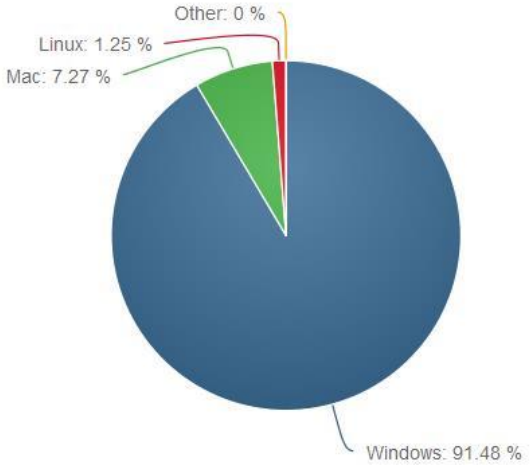
# Browseranteil nach Endgerätetyp

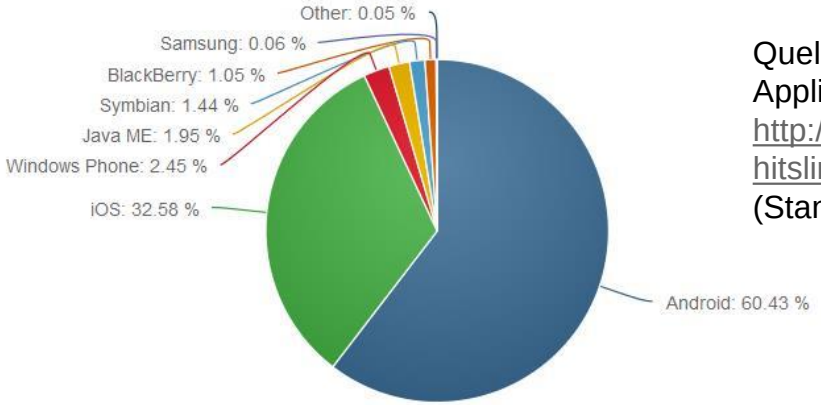



Quelle: Net Applications, http://marketshare.hitslink.com, (Stand 03/2017)

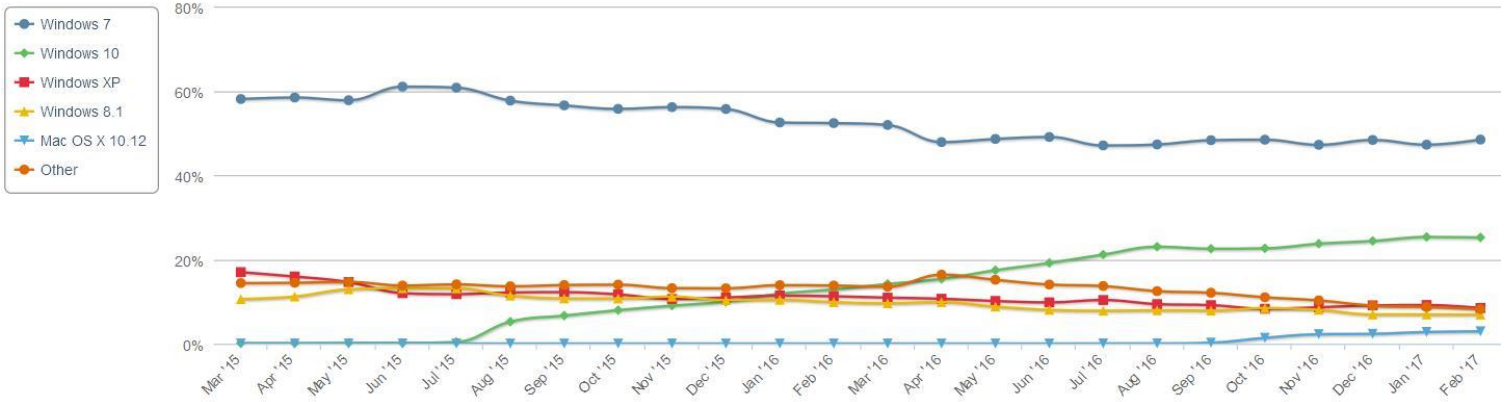
Ca. 35% der Internetzugriffe kommen 2017 von mobilen Geräten.
Die Tendenz ist stark steigend. 2011 waren es nur 4%.

# Marktanteile und Trend der Betriebssysteme von Webclients




Quelle: Net Applications, http://marketshare.hitslink.com, (Stand 03/2017)




Fazit: Windows und der Internet Explorer sind für Webanwendungen Pflicht!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.1 HTML

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Was ist HTML ?

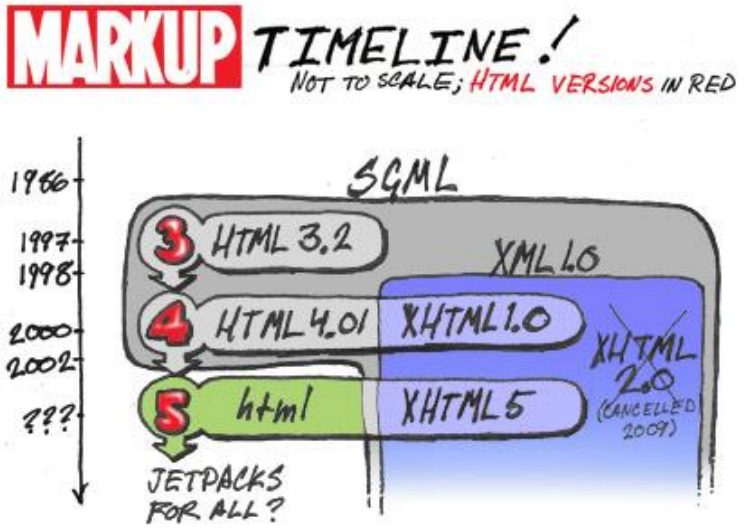
<title> Text des Titels </title>

- Markup Language
  - ⇨ Markup Language: Auszeichnungssprache
  - ⇨ markiert und attributiert Bestandteile eines Fließtexts
  - ⇨ Layout bewusst nicht definiert
  - ⇨ Browser setzen Auszeichnung in visuelle Darstellung um
- HyperText: Verweise auf andere Dokumente
  - ⇨ komfortable Querverweise zu anderen Stellen im eigenen Projekt oder zu beliebigen anderen Dokumenten im Web
  - ⇨ URL (Uniform Resource Locator)
- Text-Dateien (kein Binärformat)
  - ⇨ mit jedem Texteditor zu bearbeiten
  - ⇨ leicht zu generieren und zu debuggen

HTML heißt HyperText Markup Language

# SGML – XML – HTML - XHTML

- SGML (Standard Generalized Markup Language)
  - ⇨ definiert seit 1986 eine Sprachfamilie zur Auszeichnung von Text (ISO-Norm 8879)
  - ⇨ verwendet die Metasprache DTD (Document Type Definition) zur Definition konkreter Sprachen
- XML (Extensible Markup Language) ist eine Untermenge von SGML
  - ⇨ z.B. mit spezieller Klammerstruktur




http://jasonpriem.org/2010/04/markup-languages-whos-who

- HTML und XHTML sind konkrete Sprachen
  - ⇨ HTML und XHTML definieren ausgehend von SGML bzw. XML eine Sprache (Dokumenttyp) mit Tags, Attributen und einer Grammatik
  - ⇨ XHTML beinhaltet strenge Regeln und behandelt Verstöße als Fehler
  - ⇨ HTML lässt viele Freiheiten (ist weniger eindeutig) und interpretiert tolerant
- HTML5 wurde von SGML losgelöst und als eigenständige Sprache definiert
  - ⇨ für HTML5 gibt es eine HTML-und eine XHTML-Syntax ("XHTML5")

# Bemerkungen zum Werdegang

- Die Entwicklung von HTML war ein Trauerspiel
  - ⇨ Anstatt ein ausgereiftes Dokument-Format um URLs zu erweitern wurde HTML seit 1992 jahrelang "neu" erfunden
  - ⇨ Der Kampf um Browser-Marktanteile führte zu proprietären HTML-Varianten
  - ⇨ der Standardisierungsprozess hinkte immer hinter dem Entwicklungsstand der Browser her
  - ⇨ Erst mit HTML4 gab es 1998 einen angemessenen Standard
  - ⇨ HTML5 wurde 10 (!) Jahre später (2009) als Working Draft veröffentlicht
  - ⇨ Die Browser halten bis heute keinen Standard vollständig ein und beinhalten Bugs – oder interpretieren den Standard speziell
- Leidtragende sind die Nutzer
  - ⇨ Webentwickler können ihre Ziele nur schwer realisieren
  - ⇨ Surfer leiden unbewusst (wissen nicht, wie es gemeint war)

# Was ist eigentlich HTML5?


- Alle reden von HTML5 – und alle meinen unterschiedliche Dinge
  - ⇨ Das WWW-Konsortium "W3C" (http://www.w3.org) definiert traditionell den Standard und achtet auf eine saubere Darstellung
    - Üblicherweise wird der Standard in einem langen Prozess versioniert
    - Der Standard liegt weit hinter der Realität im Web zurück
  - ⇨ "WHATWG" (Web Hypertext Application Technology Working Group) beschreibt als "Living Standard" die neuesten Entwicklungen
    - Änderungen sind in der Webwelt normal und sollen in einem schnellen Prozess in den Living Standard von HTML aufgenommen werden
    - Die Änderungen sind fester Bestandteil von HTML
    - HTML5 ist keine Version, sondern ein eigenständiger Name (deshalb fehlt auch das Leerzeichen vor der 5)
  - ⇨ In der Umgangssprache steht HTML5 für das interaktive Internet mit ansprechendem Design (Web 2.0)
    - Das beinhaltet nicht nur HTML5, sondern eine Mischung diverser Webtechniken wie CSS, JavaScript, Frameworks, APIs

# Was bringen HTML5 & Co.?


- HTML5 & Co. bieten unter anderem
  - ⇨ Ausnutzung von leistungsstarken Clients
    - Aufwändige Grafiken (Canvas)
    - Unterstützung von Threads (Workers in Javascript)
    - Datenspeicherung (MicroData)
  - ⇨ Unterstützung von unterschiedlichen Clients
    - Erkennung der Auflösung (CSS3 Media Queries)
    - Offlinezugriffe und Lokalisierung
  - ⇨ Modernisierung des alten Standards
    - Eingabeelemente mit Überprüfungen (z.B. Email-Felder)
    - Audio und Video sind integriert
    - Neue Tags (z.B. Header, Footer, Navigationsbereich)
    - Schickeres Design (z.B. Schatten und runde Ecken in CSS3)

HTML5 ist das, was man im Web heute braucht!

# HTML5: Pro und Contra


- Contra
  - ⇨ W3C und WHATWG propagieren unterschiedliche Standards
  - ⇨ Neue Tags und Attribute werden von vielen Browsern nicht erkannt (das ist nicht wirklich neu)
- Pro
  - ⇨ HTML5 wurde (bzw. wird) abwärtskompatibel entwickelt und selbst alte Browser reagieren gnädig auf unbekannte Inhalte
  - ⇨ Die neue Funktionalität mit HTML4 zu implementieren ist kaum machbar
  - ⇨ Es gibt diverse Webseiten, die zeigen wie weit die neuen Features in den verschiedenen Browsern umgesetzt sind (z.B. http://canluse.com)
  - ⇨ Es gibt diverse JavaScript-Frameworks, die dabei helfen alte Browser zu unterstützen (z.B. http://modernizr.com)

Wir verwenden HTML5 !

# HTML5 und ordentlicher Code

- HTML5 wurde abwärtskompatibel und browserfreundlich definiert
  - ⇨ für viele Sprachkonstrukte, die unter HTML4 einen Fehler erzeugten, ist jetzt das Browserverhalten definiert – und es sind keine Fehler mehr
    - viele fehlende Tags werden automatisch erkannt
    - schließende Tags können oft weggelassen werden
    - Groß/Kleinschreibung der Tags ist nicht festgelegt usw.
  - ⇨ Praktische Konsequenzen
    - Code, der gegen viele Regeln eines sauberen Entwurfs verstößt, kann trotzdem HTML5-konform sein
    - um HTML5-Code auf einen ordentlichen Stil zu überprüfen, reicht der normale Konformitätscheck (HTML-Validator) nicht mehr aus
  - ⇨ Lösung
    - HTML5 sollte in professionellen Projekten zusätzlich mit einem statischen Code-Analyse-Tool auf die Einhaltung von Implementierungsrichtlinien geprüft werden (vgl. Lint für C++)

# Implementierungsrichtlinien in dieser Veranstaltung

- Implementierungsrichtlinien für EWA
  - ⇨ Kein Layout / Design in HTML sondern konsequenter Einsatz von CSS
  - ⇨ Einrückungen wie in normalen Programmiersprachen
  - ⇨ Weitere Regeln, die HTML5 einschränken, werden auf den folgenden Folien mit diesem Symbol markiert


- Die Einschränkungen sind gering und fordern keine große Umstellung
  - ⇨ Die Syntax für ein paar HTML-Konstrukte ist etwas strenger
  - ⇨ Der HTML-Code wird aber definitiv besser lesbar

Ordentlicher HTML5-Code ist die Zielsetzung in dieser Veranstaltung!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.1.1 HTML Grundlagen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

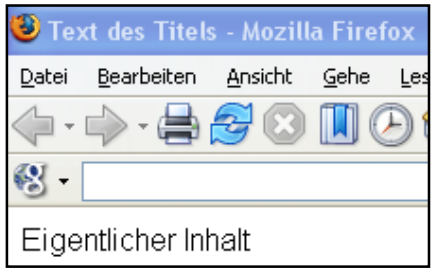
# Grundgerüst einer (ordentlichen) HTML5-Datei

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Text des Titels</title>
  </head>
  <body>
    <p>Eigentlicher Inhalt</p>
  </body>
</html>
```

Dokumenttyp

Zeichensatz

Titel für Browserfenster



# Schreibregeln

- Leerzeichen, Tabulator und Zeilenvorschub sind Trenner
  - ⇨ Anzahl spielt keine Rolle, außer in Attributwerten
  - ⇨ Ausnahme: in `<pre>` Abschnitten (=preformatted)
- Einrückung dient nur der Lesbarkeit
  - ⇨ wird vom Browser ignoriert
  - ⇨ wird von manchen WYSIWYG Tools ruiniert
- Kommentare

```
<!-- das ist ein Kommentar -->
```

- Sonderzeichen und Umlaute können kodiert werden

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| < | &lt; | > | &gt; | & | &amp; | " | &quot; |
| ä | &auml; | Ä | &Auml; | ö | &ouml; | Ö | &Ouml; |
| ü | &uuml; | Ü | &Uuml; | ß | &szlig; | € | &euro; |

# Tags (Marken) und Attribute

- Tags (Marken) markieren Abschnitte im Text
  - ⇨ Tag-Name steht in spitzen Klammern
  - ⇨ gleicher Name für öffnendes und schließendes Tag
  - ⇨ schließendes Tag kenntlich an zusätzlichem /
  - ⇨ Der Name des Tags wird kleingeschrieben 

  - ⇨ `<h2>willkommen in unserem Hotel</h2>`
- öffnende Tags können zusätzliche Attribute enthalten
  - ⇨ Attribute haben einen Namen und in der Regel einen Wert
  - ⇨ Attributwerte werden in Anführungszeichen geschrieben 


```
<h2 id="hallo">willkommen in unserem Hotel</h2>
```

- mit Tags markierte Abschnitte können verschachtelt sein

```
<h2><em>willkommen</em> in unserem Hotel</h2>
```

# Sonderfall Standalone-Tags

- es gibt einige wenige "Standalone-Tags"
  - ⇨ leere Elemente = Abschnitte ohne Inhalt
  - ⇨ werden durch einen / vor der schließenden Klammer des öffnenden Tags hervorgehoben


```
willkommen <br />
auf unserer Homepage
```

oder

```
<meta charset="UTF-8" />
```

- alternative Schreibweisen (die wir aber nicht verwenden)

`<br>` oder `<br></br>`

# Strukturierung von Text

alle außer  und <br /> erzeugen einen Block

- Überschriften
  - `<h1>` Überschrift der höchsten Gliederungsebene
  - `<h6>` Überschrift der niedrigsten Gliederungsebene

heading1 … heading6

- Abschnitte
  - `<p>` Textabsatz
  - `<div>` allgemeiner Block

div = division = Bereich

  - `<span>` Inline-Element

kein Block

"Aufhänger" für CSS

- Aufzählungen (nummeriert oder auch nicht)
  - `<ol>, <ul>, <li>`

ordered list, unordered list, list item

- Zeilenumbruch erzwingen und verhindern
  - `<br />` expliziter Zeilenumbruch (standalone tag)

kein Block

  - ` ` geschütztes Leerzeichen – verhindert Zeilenumbruch
  - `&shy;` soft hyphen – Bindestrich bei Bedarf

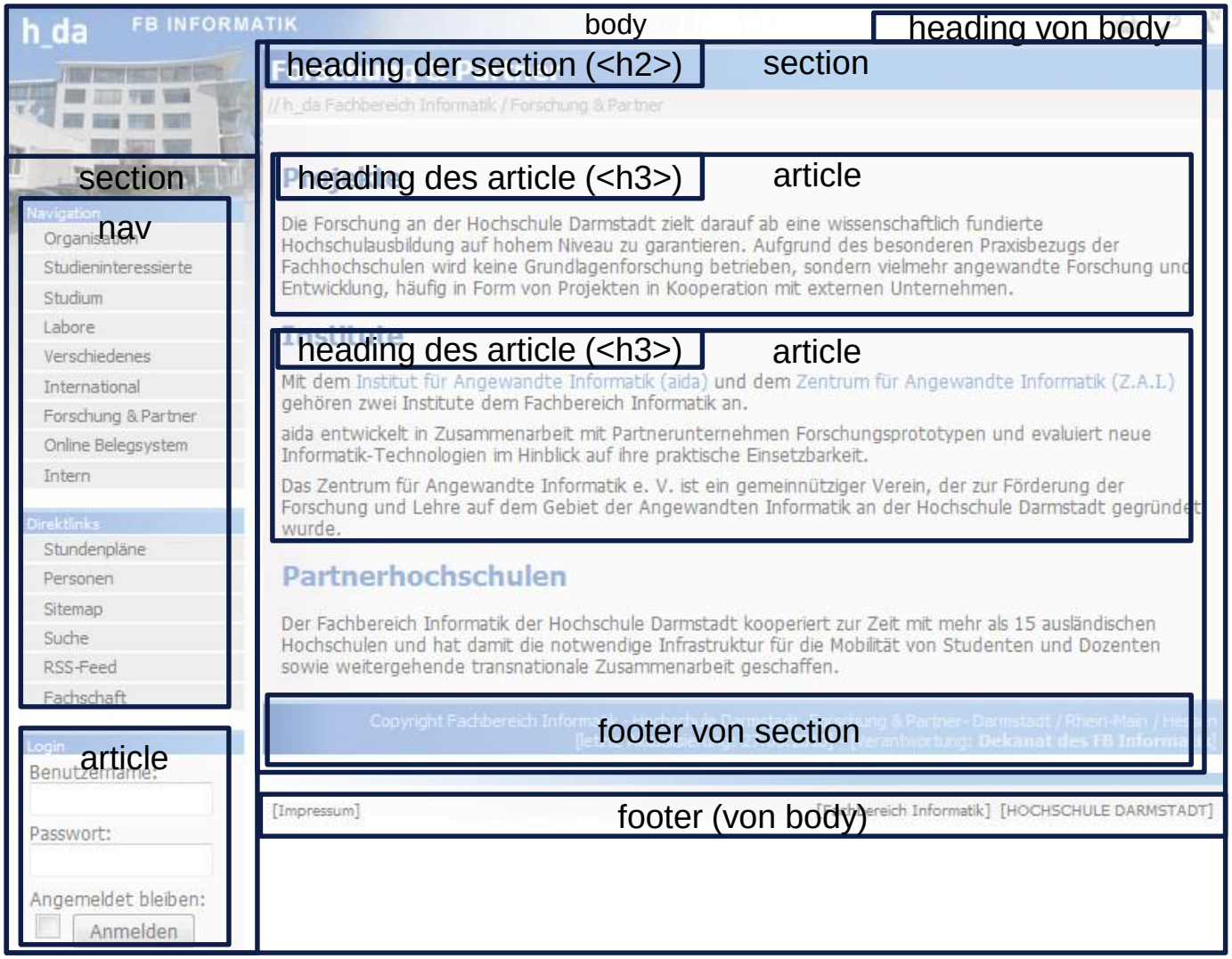
z.B. 3. Kapitel

# Strukturierung von Webseiten


- Der Text innerhalb des <body>-Tags kann gegliedert werden
  - ⇨ <section> Abschnitt - ein logischer Bereich einer Webseite (z.B. der News Bereich)
  - ⇨ <article> Artikel
    ein Textabschnitt, der eigenständig einen Inhalt abdeckt (z.B. eine einzelne News)
  - ⇨ <body> kann mehrere <section>s und <article>s enthalten – auch verschachtelt
  - ⇨ <nav> Navigationsbereich – enthält Verknüpfungen zur Navigation
  - ⇨ <header> und <footer> Kopf- / Fußzeilenbereich
    ein Bereich mit Überschriften bzw. Logos, Datum usw. für das Gesamtdokument (im <body>) oder <section>s und <article>s
  - ⇨ <aside> Blöcke an der Seite
    z.B. für Seitenleisten (außer Navigation), Zitate, Anmerkungen
  - ⇨ Erläuterung: http://ie.microsoft.com/testdrive/HTML5/SemanticNotepad/

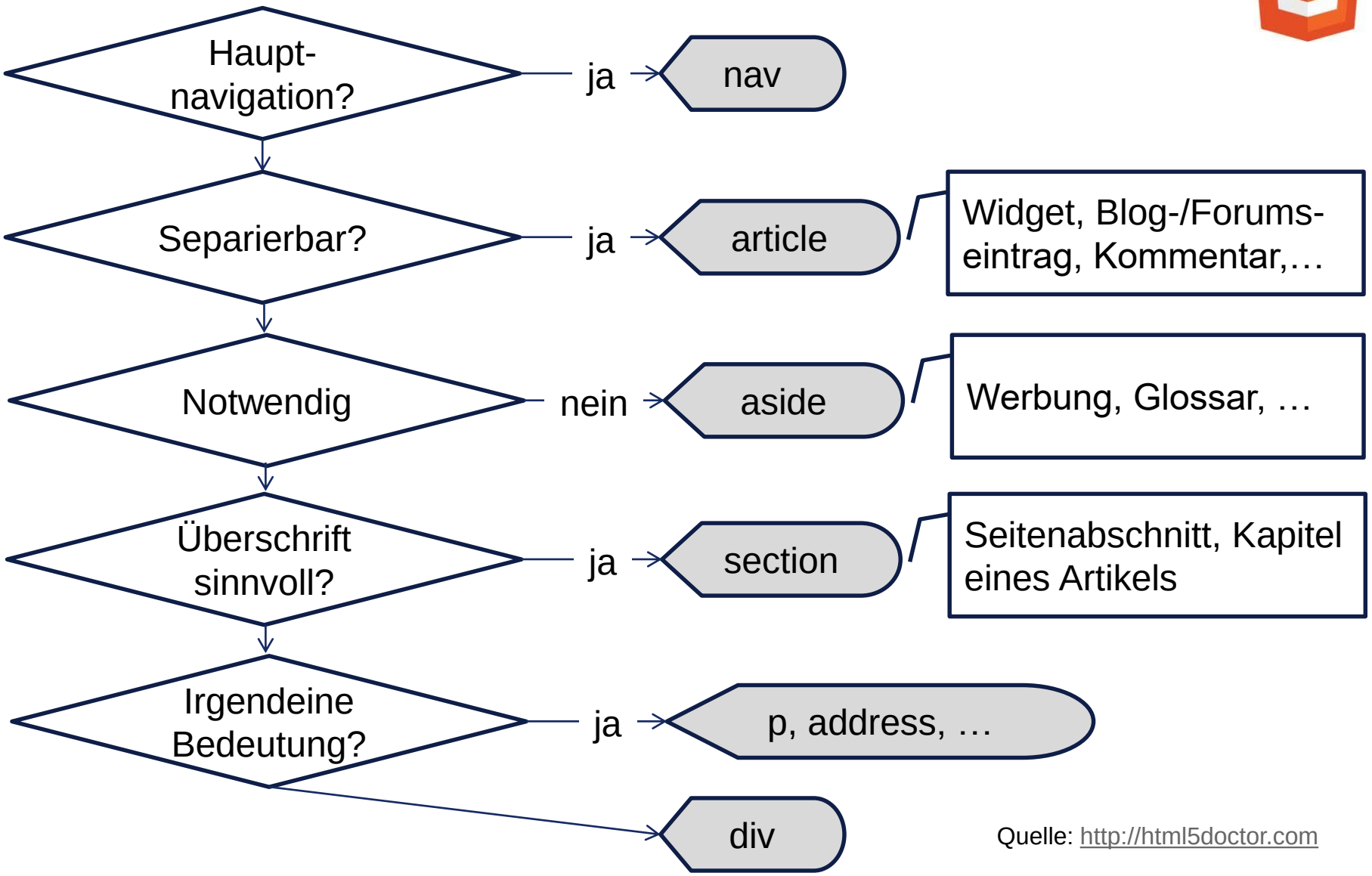
<section>s und <article>s haben eine Überschrift (<h2>...<h6>)!

# Strukturierung von Webseiten am Beispiel




IE8 kennt das nicht!

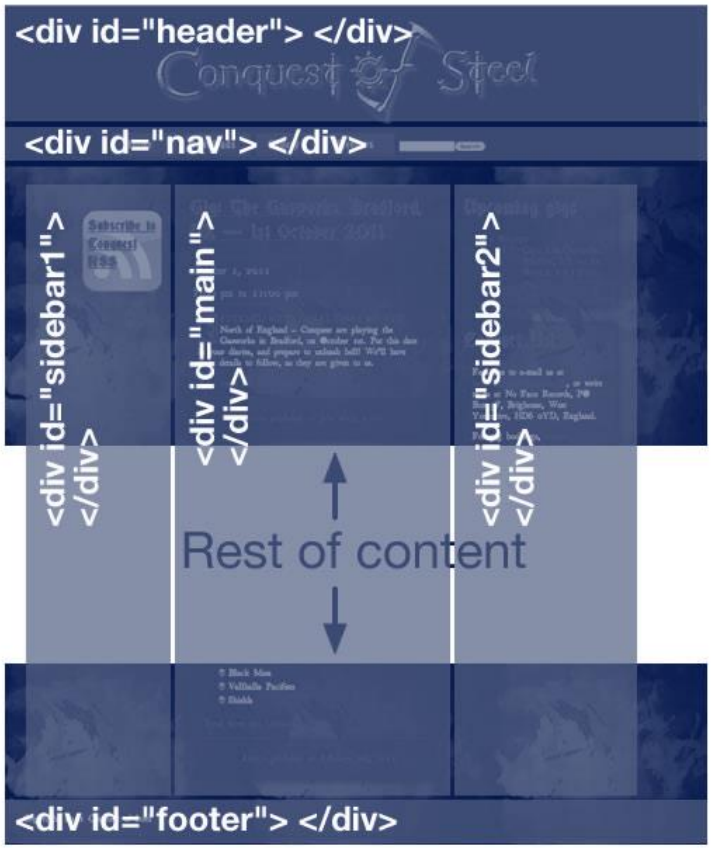
# Welches Strukturelement ist das Richtige?



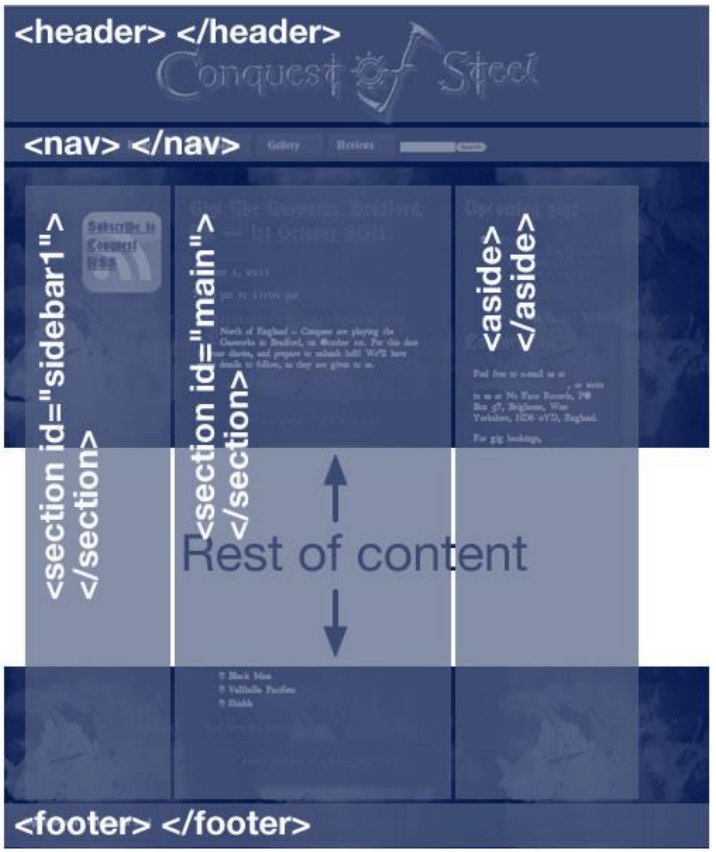

Quelle: http://html5doctor.com

# Vergleich HTML4 und HTML5

- HTML4




- HTML5




Quelle: http://www.w3.org/wiki/HTML_structural_elements

# Universalattribute

- **können zu jedem Tag hinzugefügt werden**
  - ⇨ `id` dateiweit eindeutiger Bezeichner für Scripte
  - ⇨ `class` Name der zugehörigen Style Sheet Klasse
  - ⇨ `title` Erläuterung zum Element, erscheint als Tooltip
  - ⇨ `style` eingebettete Style Sheet Attribute (siehe CSS)
  - ⇨ `lang, dir` Landessprache und Textlaufrichtung
  - ⇨ …

```
<h2
      id="JB007" class="mycssstyleclass" title="mytooltip"
      style="color:red" lang="de" dir="ltr">
Hallo</h2>
```

# Entwicklerdefinierte Datenattribute


- Zu jedem Tag können Datenattribute hinzugefügt werden
  - ⇨ mit dem Prefix "data-" zur Kennzeichnung
  - ⇨ mit einem Namen ohne Großbuchstaben
  - ⇨ Bsp.: data-preis, data-key, data-xxx
  - ⇨ In HTML oder auch über das DOM
- Beispiel Pizza-Service:

```
<ol>
 <li data-preis="4.99">Pizza Salami</li>
 ...
</ol>
```

Zur eigenen Verarbeitung mit JavaScript / DOM

# Logische Formatierung

- markiert Bedeutung von Textabschnitten
- macht keine Aussage über visuelles Erscheinungsbild
  - ⇨ das wird erst per CSS definiert
  - ⇨ für Sprachausgabe muss stattdessen das akustische Erscheinungsbild definiert werden...
- ein paar Beispiele:

| | |
|---|---|
| `<em>` | emphatisch (betont) |
| `<strong>` | stark betont |
| `<samp>` | Beispiel |
| `<dfn>` | Definition |
| `<cite>` | inline-Zitat (z.B. für Eigennamen; oft kursiv) |
| `<q cite="URL">` | Zitat mit Quellenangabe |
| `<blockquote>` | Zitat als Block gesetzt |

Block

# Grenzfall: physische Formatierung

- definiert das visuelle Erscheinungsbild
- nicht verpönt, aber besser zu vermeiden – es gibt oft andere Tags mit einer echten Bedeutung!

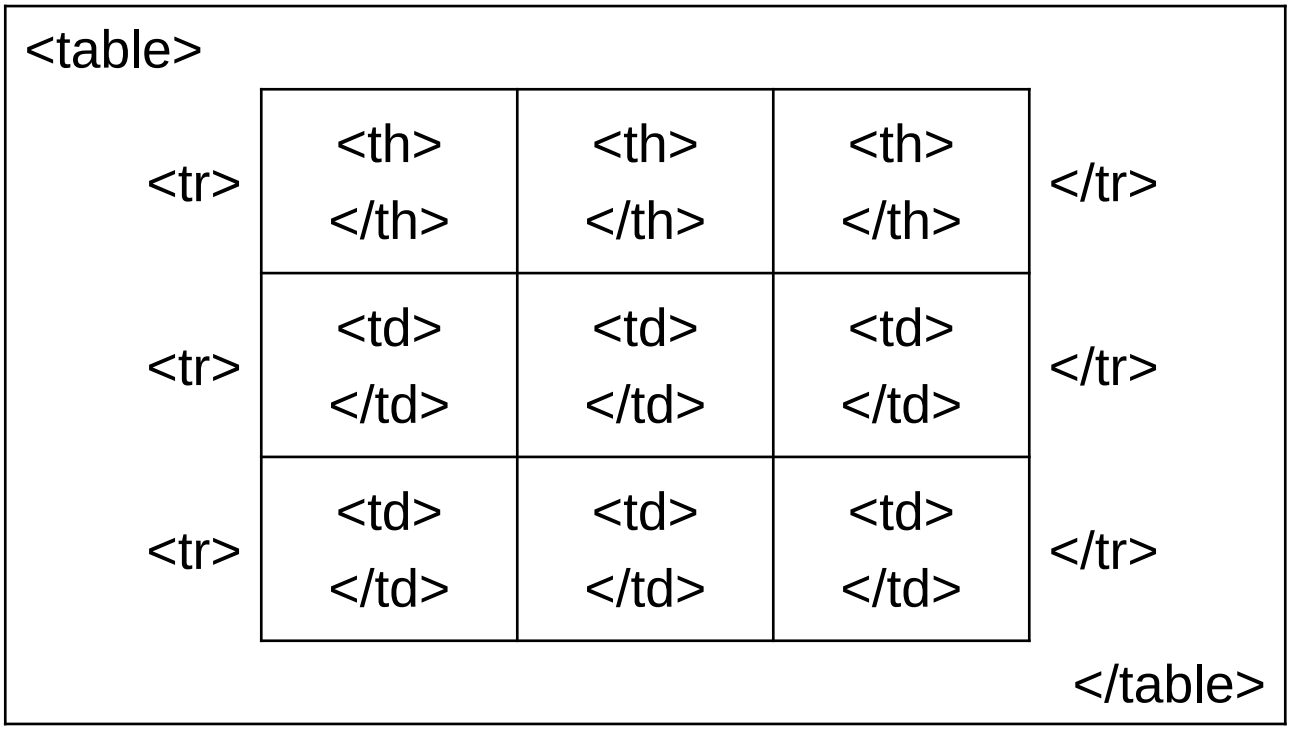
- ein paar Beispiele:

| | |
|---|---|
| `<b>` | **fette Schrift (bold)** |
| `<i>` | ***kursive* Schrift (italic)** |
| `<sup>` | **Schrift** $^{\text{hochgestellt}}$ |
| `<sub>` | **Schrift** $_{\text{tiefgestellt}}$ |
| `<pre>` | `vorformatierter Text (nur für Quellcode)` |

Block

# Struktur einer Tabelle (1)

Grundidee:




*nach S. Münz: SELFHTML*

# Struktur einer Tabelle (2)

```
<table>
   <caption>Meine Tabelle</caption>
    <tr>
       <th>Kopfzelle: 1. Zeile, 1. Spalte</th>
       <th>Kopfzelle: 1. Zeile, 2. Spalte</th>
    </tr>
    <tr>
       <td>Datenzelle: 2. Zeile, 1. Spalte</td>
       <td>Datenzelle: 2. Zeile, 2. Spalte</td>
    </tr>
    <tr>
       <td>Datenzelle: 3. Zeile, 1. Spalte</td>
       <td>Datenzelle: 3. Zeile, 2. Spalte</td>
    </tr>
</table>
```

nur zur Darstellung tabellarischer Daten

Tabellen-Überschrift

zeilenweise (tr = table row)

beliebig viele Zeilen und Spalten

# Struktur einer Tabelle (3)

- Spaltenbreite definieren
  - ⇨ falls der Browser die Spaltenbreite nicht automatisch bestimmen soll, kann über CSS die Breite festgelegt werden
  - ⇨ die Spalten werden vor dem ersten Tabelleneintrag "benannt"

```
<colgroup>
   <col id="col1" />
   <col id="col2" />
</colgroup>
```

Für einen schnellen Aufbau großer Tabellen:
Die Breite aller Spalte festlegen!
Eine id macht die Spalte für CSS adressierbar:

```
#col1 {width:80%;}
```

- Zellen verbinden
  - ⇨ Spalten verbinden `<td colspan="3">...</td>`
  - ⇨ Zeilen verbinden `<td rowspan="2">...</td>`

auch kombinierbar

- Tabellenüberschrift
  - ⇨ unmittelbar nach dem `<table>`-Tag `<caption>...</caption>`
  - ⇨ Unterschrift mit `<caption style="caption-side:bottom">`

# Grafiken einbinden

- Das `<img>`-Tag erlaubt das Einbinden von Bildern

standalone tag

  - ⇨ für Bilddateien der Formate GIF, PNG, JPG und seit HTML5 SVG
  - ⇨ notwendige Attribute:

| | |
|---|---|
| `src` | Quelldatei |
| `alt` | Ersatztext, der ausgegeben wird, wenn die Grafik nicht angezeigt werden kann (Screenreader, Ladeprobleme)<br>Dient die Grafik nur zur Veranschaulichung (z.B. Bild einer Pizza neben dem Namen der Pizza) wird `alt=""` gesetzt |

  - ⇨ optionale Attribute

| | |
|---|---|
| `width, height` | Breite und Höhe in Pixeln |
| `title` | Text, der als Tooltip angezeigt wird |

besser in CSS setzen

  - ⇨ Beispiel

```
<img src="bilder/PizzaSalami.gif" width="250" height="250"
  alt="" title="Salami Pizza" />
```

# Bilddateien

Per JavaScript gezeichnete Grafiken mit <canvas>

- GIF oder PNG für Grafiken (z.B. Screenshots, ClipArts)
  - ⇨ GIF: 256 Farben Palette, komprimiert
  - ⇨ Nachfolger: PNG mit Palette und True Color, komprimiert, verlustfrei
  - ⇨ Freistellen (transparenter Hintergrund) möglich
- JPEG für Fotos
  - ⇨ Echtfarben, komprimiert, verlustbehaftet
  - ⇨ Freistellen nur bei PNG
- SVG (Scalable Vector Graphics) für technische Zeichnungen
  - ⇨ XML-basiertes Format für 2D-Vektorgrafiken
  - ⇨ ohne Qualitätsverlust skalierbar
- Möglichkeiten zur Speicherplatzersparnis
  - ⇨ Hintergrund mit kleiner Grafik "kacheln"
  - ⇨ Größe und Farbtiefe reduzieren
  - ⇨ Beschränkung auf wenige Grafiken


Hauptproblem Übertragungszeit!

# Audio und Video

- Audio und Video gehören erst seit HTML5 zum Standard
  - ⇨ Vor HTML5 waren Multimediadateien nur über Plugins (z.B. FlashPlayer) abspielbar (und die Plugins waren nicht betriebssystemübergreifend verfügbar)
  - ⇨ Seit HTML5 gibt es "nur noch" Uneinigkeit über die vom Browser zu unterstützenden Formate (wegen Lizenzrechten)
  - ⇨ Damit die AV-Datei von allen Browsern abgespielt wird, können verschiedene Formate bereitgestellt werden
    - Mittlerweile unterstützen aber alle Browser mp3 und mp4
    - mp3 und ogg für Audio bzw. ogv und mp4 für Video

```
<audio autoplay controls>
        <source src="myAudio.mp3" />
        <source src="myAudio.ogg" />
</audio>
<video width="320" height="240" poster="bild.jpg" controls autoplay>
        <source src="myVideo.mp4" />
        <source src="myVideo.ogv" />
        Ihr Browser unterstützt leider kein Video-Tag
</video>
```

Der Browser wählt ein Format, das er unterstützt!

  - ⇨ Abspielen erfolgt im "streaming mode" wegen der Übertragungszeit

# Meta-Angaben

- Anweisungen für WWW-Server, WWW-Browser und automatische Suchprogramme ("Robots")
- eine kleine Auswahl von Meta-Angaben:

```
<meta name="description" content="Autovermietung" />
<meta name="author" content="B. Kreling" />
<meta name="keywords" content="Hotel,Urlaub,Meer" />
<meta name="robots" content="noindex" />
<meta name="date" content="2001-02-06" />
<meta name="language" content="de" />
<meta http-equiv="refresh" content="5" />
```

lädt die aktuelle Seite nach 5 Sekunden erneut

- Setzen des verwendeten Zeichensatzes

```
<meta charset="UTF-8" />
```

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.1.2 Hyperlinks

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Anwendungsfälle für Hyperlinks

- Beispiele für Einsatzmöglichkeiten
  - ⇨ Querverweis (vgl. Lexikon, Literaturstelle)
  - ⇨ Blättern (nächste Seite / vorige Seite)
  - ⇨ Inhaltsverzeichnis (Unterkapitel / Oberthema)
  - ⇨ Stichwortverzeichnis
  - ⇨ freie Navigation, neue Dokumentstrukturen ⇨ Hypermedia
  - ⇨ Download einer Datei
  - ⇨ sonstiger Dienst
- "Hyperlink" ist lediglich eine technische Realisierung !
- Einsatzgebiet klären und gestalterisch unterscheiden

# Gestaltungstipps für Verweise

- ein Verweis ist ein Blickfang
  - ⇨ nur bedeutungstragende Begriffe mit Hyperlink hinterlegen
- Verweistext soll das Verweisziel deutlich machen
  - ⇨ vorzugsweise immer derselbe Text für dasselbe Ziel
  - ⇨ nicht zu viele Verweise auf dieselbe Stelle innerhalb einer Seite
- Verweis sollte unmittelbar erkennbar sein
  - ⇨ nicht erst nach "Abtasten" mit der Maus
- alle Seiten vollständig verlinken
  - ⇨ "Zurück"-Button des Browsers sollte innerhalb einer Website überflüssig sein - aber er sollte möglichst auch funktionieren

# Ziele von Verweisen

- eine Datei, die der Browser als Seite darstellen kann
  - ⇨ meistens HTML, aber auch anderes möglich
  - ⇨ im Internet oder lokal
- bestimmte Position ("Anker") innerhalb einer darstellbaren Datei
- eine Datei, die der Browser selbst nicht darstellen kann
  - ⇨ diese wird zum Download angeboten oder mit einer Hilfsanwendung geöffnet
- andere Dienste neben WWW
  - ⇨ mailto, gopher, ftp, telnet, news

```
<a href="mailto:j.bond@fbi.h-da.de">J. Bond</a>
<a href="ftp://www.xyz.de/setup.zip">Download</a>
<a href="file:///c:/lokal.htm">lokale Datei</a>
```

# Verweise

Der Verweistext sollte eine klare Information über das Ziel des Verweises geben !

- Allgemeine Form

```
<a href="Dienst://Server:Port/Verz/Datei#Anker">
  Text</a>
```

  Teile davon können weggelassen werden

- Datei im selben / unter- / übergeordneten Verzeichnis

```
<a href="start.htm">Text</a>
<a href="sub/Datei.html">Text</a>
<a href="../inhalt.htm">Text</a>
```

relativ

- Datei auf anderem Server

auch: localhost

```
<a href="http://www.xyz.de/datei.htm">Text</a>
```

absolut

- Groß-/Kleinschreibung beachten

beliebter Fehler unter Windows

  ⇨ Server laufen meist unter Unix und Unix ist case sensitive bezüglich Datei- und Verzeichnisnamen

# Absolute und relative Verweise

ohne Angabe von Server und Verzeichnispfad

- relative Verweise innerhalb der eigenen Website (projekt-intern) sind vorteilhaft für
  - ⇨ Migration auf anderen Server oder in anderes Verzeichnis
  - ⇨ Entwicklung auf lokaler Festplatte mit späterem Upload
  - ⇨ Download als ZIP und lokale Installation (vgl. SELFHTML)

- absolute Verweise sind vorteilhaft für
  - ⇨ Versenden von Seiten per eMail (z.B. Werbung, Stundenplan; sofern der Leser online ist wird er direkt auf den Webserver weitergeleitet)
  - ⇨ Verweise auf fremde Websites (projekt-extern)

# Verweise innerhalb einer Datei ("Anker")

- wird häufig eingesetzt für "Inhaltsverzeichnis" am Anfang einer Datei
  - ⇨ z.B. bei FAQ
- Verweisziel definieren per id in beliebigem Tag

```
<h2 id="Erl">Erläuterung</h2>
```

veraltet:
<a name="Erl">...</a>

- Verweis definieren

```
siehe die <a href="#Erl">Erläuterung</a> unten
```

- der Verweis kann auch zu einer bestimmten Position in einer anderen Datei zeigen

```
<a href="datei.htm#Erl">Erläuterung</a>
<a href="http://www.xyz.de/datei.htm#Anker">...</a>
```

- der Browser scrollt die Seite so, dass der Anker an der Oberkante des Fensters erscheint

# Zusammenfassung

- Grundgerüst: DOCTYPE, <html>, <head>, <body>, <title>, charset...
- Schreibregeln: Zeilenumbruch, Kommentare und Sonderzeichen
- Tags und Attribute
- Tabellen
- Logische Formatierung und verpönte Formatierung
- Einbinden von Grafiken, Audio, Video...
- Meta-Angaben
- Verwendung von Hyperlinks <a href="..." >
- Verweise innerhalb einer Seite (Anker)

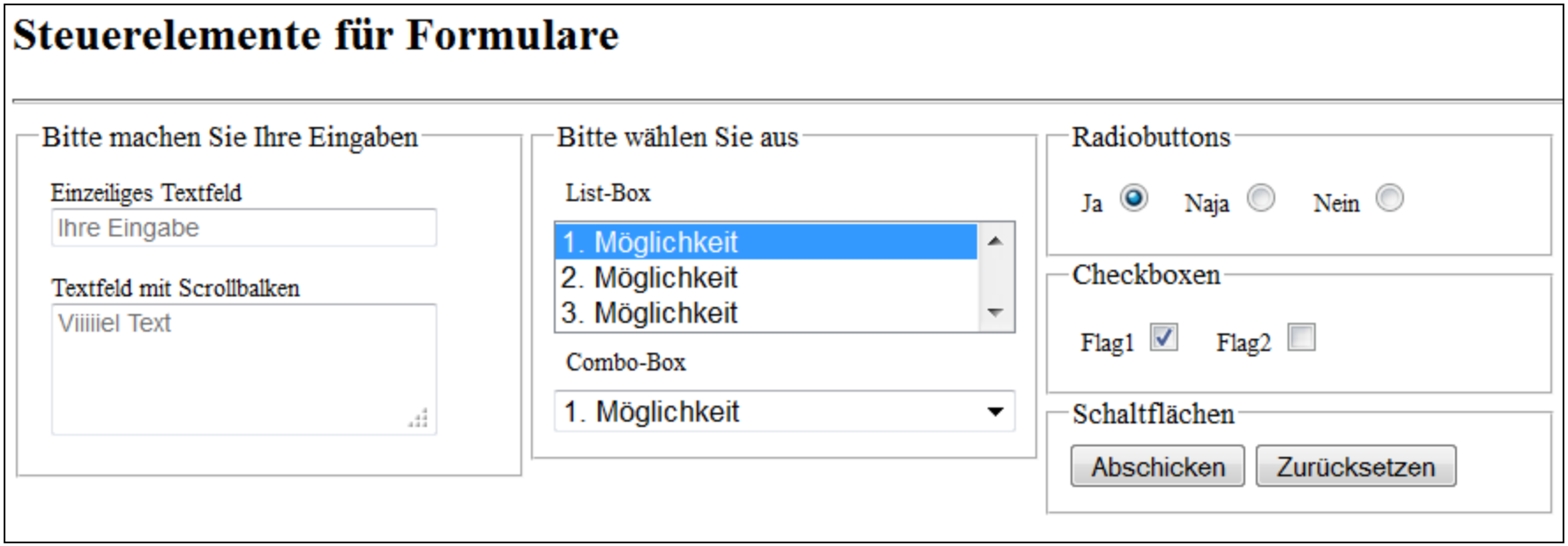
Jetzt können Sie eine einfache HTML-Seite schreiben!

# Hochschule Darmstadt
# Fachbereich Informatik

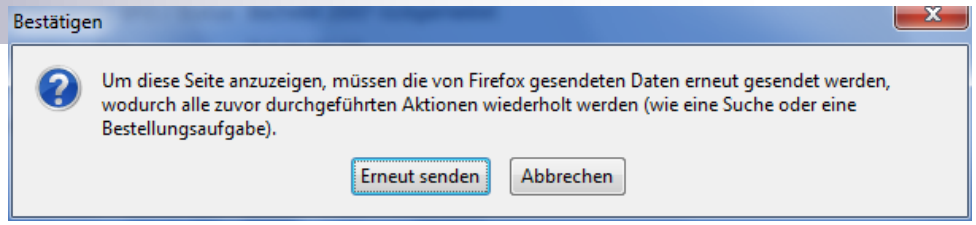
## 2.1.3 HTML Formulare

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Formular - Beispiel



# Funktion von Formularen




- Formulare dienen der Eingabe von Daten
  - ⇨ eingegebene Daten werden an Server übermittelt und dort ausgewertet
  - ⇨ es gibt 2 Möglichkeiten der Datenübertragung
    - `get` übermittelt Parameter für Abfrage (z.B. Suchmaschine)
    - `post` übermittelt Daten zwecks Speicherung (z.B. Bestellung)

vgl. Reload im Browser

- Bereich mit Eingabeelementen im HTML-Body markieren

```
<form action="/cgi-bin/Echo.pl" id="form1"
    accept-charset="UTF-8" method="get">
      Steuerelemente (Eingabefelder, Auswahllisten,
      Buttons...) und sonstige HTML-Tags und CSS-Formatierung
</form>
```

hier: Übergabe der Daten an Perl-Skript

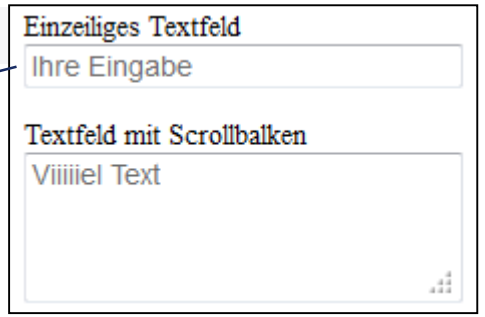
  - ⇨ accept-charset zur Sicherheit gegen willkürliche Benutzereinstellung
  - ⇨ falls das Steuerelement außerhalb des Formulars liegt, kann der Bezug über `form="form1"` hergestellt werden

aber nicht mit Internet Explorer

- Alternative Aktion: Formulardaten per eMail verschicken
  - ⇨ `action="mailto:Meier@xyz.de"`
  - ⇨ unsicher, weil von der Installation beim Surfer abhängig

# Eingabefelder

placeholder erscheint nur ohne value




- einzeilige Textbox

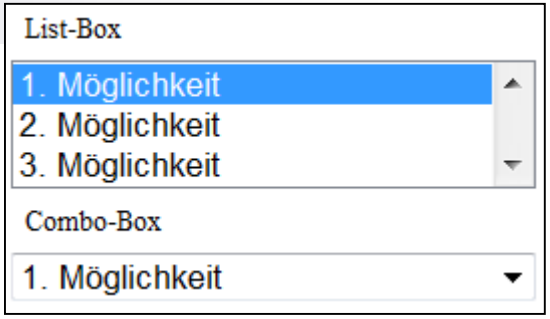
```
<input type="text" name="zuname" value="Muster"
  size="30 maxlength="40" placeholder="Ihre Eingabe"
  readonly />
```

standalone tag

  - ⇨ `name` und `value` wird an Server übermittelt
  - ⇨ `value` kann vorbelegt sein
  - ⇨ `size` und `maxlength` für Anzeigelänge und Maximalgröße
  - ⇨ `placeholder` wird angezeigt, bevor man eine Eingabe macht
  - ⇨ mit `readonly` reine Anzeige (ausgegraut)
- Variante: Passwortfeld mit *-Anzeige
  - ⇨ wie oben, jedoch `type="password"`
  - ⇨ keine verschlüsselte Übertragung!
- mehrzeiliges Textfeld (bei Bedarf mit Scrollbalken)

```
<textarea name="feedback" cols="50" rows="10"
  placeholder="Viiiiiel Text"></textarea>
```

# Auswahllisten

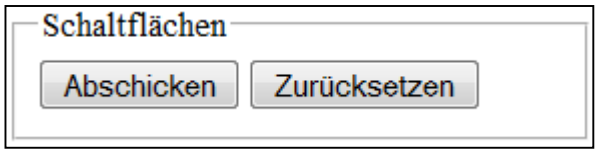



- Listbox

```
<select name="top4[]" size="3" multiple>
  <option selected>     1. Möglichkeit</option>
  <option>              2. Möglichkeit</option>
  <option value="3">    3. Möglichkeit</option>
  <option>              4. Möglichkeit</option>
  </select>
```

  - ⇨ `size` bestimmt die Höhe in Zeilen
  - ⇨ Vorauswahl ggfs. mit `<option selected>`
  - ⇨ angezeigter Text wird als ausgewählter Wert übertragen, sofern kein `<option value="xyz">` definiert ist
- Combobox
  - ⇨ Eine Listbox mit `size="1"` liefert eine aufklappende Liste mit Optionen
- Mehrfachauswahl mit zusätzlichem Attribut `multiple`
  - ⇨ Bei erlaubter Mehrfachauswahl eckige Klammern an den Namen hängen (z.B. `name="top4[]"`)! PHP macht dann daraus ein Array!

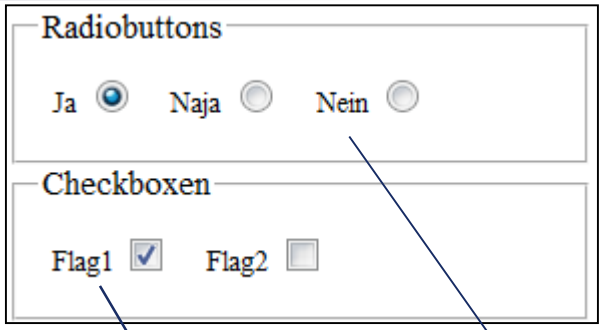
# Schaltflächen und verborgene Felder




- allgemeine Schaltflächen für JavaScript-Ereignisse
  ```
  <input type="button" name="Start" value="Startseite"
  onclick="self.location.href='http://www.xyz.de/'" />
  ```
- Schaltfläche zum Absenden der Formulardaten
  - ⇨ wie oben, jedoch `type="submit"` ; `onclick` nicht nötig
- Schaltfläche zum Löschen der Formulardaten
  - ⇨ wie oben, jedoch `type="reset"` ; `onclick` nicht nötig
- verborgenes Datenfeld (z.B. für Sessionverwaltung)
  - ⇨ `<input type="hidden" name="sessionID" value="4711" />`
- in HTML5 gibt es noch diverse andere Typen für das Input-Tag
  - ⇨ `<input type="email"... />` oder auch `date`, `number`, `color` uvm.
  - ⇨ diese Typen überprüfen automatisch die Eingabe


Daten werden nur dann übertragen, wenn die Felder ein name-Attribut haben!

# Radiobuttons und Checkboxen




- Radiobuttons als Gruppe von Knöpfen, die sich gegenseitig auslösen (Auswahl 1 aus n)
  - ⇨ Gruppierung erfolgt durch identischen `name`
  - ⇨ der `value` wird als Wert der Gruppe übertragen

```
<input type="radio" name="OK" value="1" />
<input type="radio" name="OK" value="2" />
<input type="radio" name="OK" value="3" checked />
```

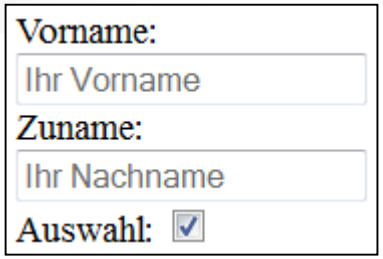
  - ⇨ Vorauswahl durch Attribut `checked`
  - ⇨ Zur Beschriftung ist `<label>` geeignet (nächste Folie)
- Checkboxen für Boole'sche Eingabe

"on" wenn value fehlt

```
<input type="checkbox" name="zutat" value="salami" />
```

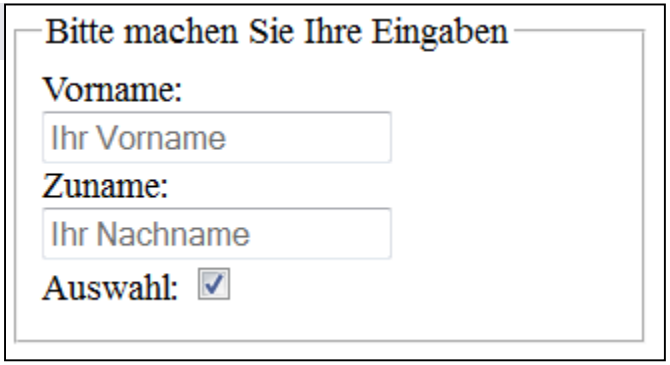
  - ⇨ übermittelt wird der `value` nur für angekreuzte Checkboxen
  - ⇨ Vorauswahl durch Attribut `checked`
  - ⇨ Beschriftung erfolgt mit Labels

# Beschriftung von Formularelementen




- Formularelemente haben kein Attribut für Text
  - ⇨ Der Text "Vorname" und das Eingabefeld "Ihr Vorname" im Beispiel haben (für den Browser) keinen Zusammenhang
- Mit Hilfe von "Labels" wird ein logischer Bezug zwischen Formularelement und Beschriftungstext hergestellt
  - ⇨
```
<label>Zuname:
    <input type="text" id="nachname" name="Zuname"/>
</label>
```
  - ⇨ Das `<label>` umschließt "sein" Formularelement
  - ⇨ Alternativ wird der Bezug über eine `id` und das Attribut `for` hergestellt: `<label for="nachname"></label>`
  - ⇨ anwendbar für `<input>`, `<select>` und `<textarea>`
- Vorteile bei der Verwendung
  - ⇨ Beim Klicken auf den (zugeordneten) Text wird das Eingabefeld selektiert bzw. die Checkbox selektiert
  - ⇨ Unterstützung von Screenreadern

# Gruppierung von Formularelementen




- Größere Formulare bestehen häufig aus *Gruppen* von Elementen. Ein typisches Bestellformular besteht beispielsweise aus Elementgruppen wie "Absender", "bestellte Produkte" und "Formular absenden"
  - ⇨ Eine zusammengehörige Gruppe von Formularelementen wird durch `<fieldset>...</fieldset>` umrahmt
  - ⇨ Dazwischen können Sie beliebige Teile Ihres Formulars definieren.
- Unterhalb des einleitenden `<fieldset>`-Tags und vor den ersten Formularinhalten der Gruppe sollte eine Gruppenüberschrift (z.B. Formular) für die Elementgruppe vergeben werden.
  - ⇨ Schließen Sie den Gruppenüberschriftentext in die Tags `<legend>...</legend>` ein
- Vorteil bei der Verwendung
  - ⇨ Formatierung nach Wunsch über HTML/CSS
  - ⇨ Web-Browser kann Elementgruppen durch Linien oder ähnliche Effekte optisch sichtbar machen

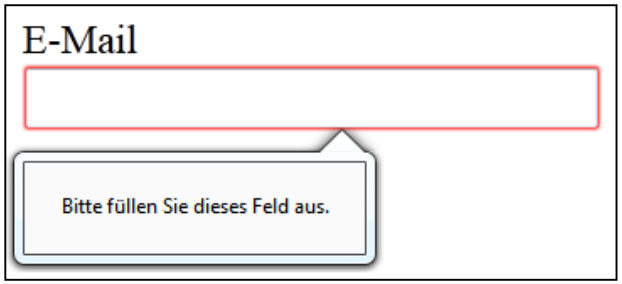
# Attribute für Eingabefelder

- Hinweistexte
  - ⇨ werden bei Eingabefeldern angezeigt, bevor man eine Eingabe macht
  - ⇨ `placeholder="Ihr Nachname"`
- Tabulatorreihenfolge
  - ⇨ normalerweise entsprechend der Reihenfolge in der HTML-Datei
  - ⇨ oder explizit setzen mit Attribut `tabindex="1"` usw.
- Tastaturkürzel definieren
  - ⇨ `accesskey="x"` springt mit alt+x sofort in das entsprechende Eingabefeld
  - ⇨ Dies ist im Browser nicht erkennbar und muss beschriftet werden !

# Attribute zur Validierung von Eingabefelder

- required

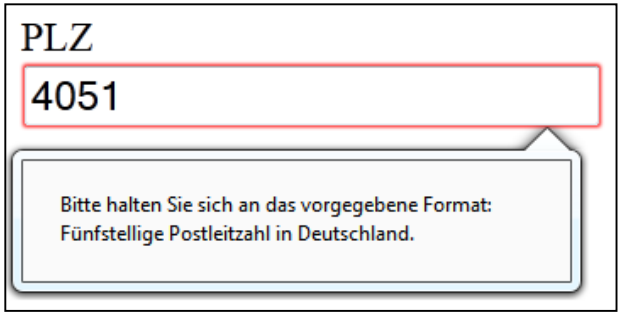
```
<input type="email"
  required />
```




- pattern

```
<input pattern="[0-9]{5}" name="plz"
  title="Fünfstellige Postleitzahl in Deutschland."/>
```

Ohne required, darf das Feld trotz pattern auch leer bleiben!




- min..max

```
<input name="bday" type="date" max="1994-12-31" />
```

# Zusammenfassung

Jetzt wissen Sie alles um eine (statische) HTML-Seite zu entwickeln!

- Grundidee Formulare (Übertragung von Daten an den Web Server)
- Aufbau von Formularen
  - ⇨ 1- und mehrzeiliges Textfeld (`<input type="text"... />` bzw. `<textarea> ... </textarea>`
  - ⇨ Listbox und Combobox (`<select...><option>`...)
  - ⇨ Radiobuttons und Checkboxen `<input type="radio" name="x".../>` bzw. `<input type="checkbox".../>`)
  - ⇨ Schaltflächen und verborgene Felder `<input type="button" ... onclick.../>` bzw. `<input type="hidden".../>`
  - ⇨ Abschicken von Formularen `<input type="submit".../>`
  - ⇨ Beschriftung von Formularelementen `<label>`, `<fieldset>` und `<legend>`

Daten werden nur für Formularelemente übertragen, die innerhalb eines <form>s liegen und ein name-Attribut haben!
Wohin die Übertragung geht, legt das action-Attribut des <form>s fest.

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.1.4 HTML Werkzeuge

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

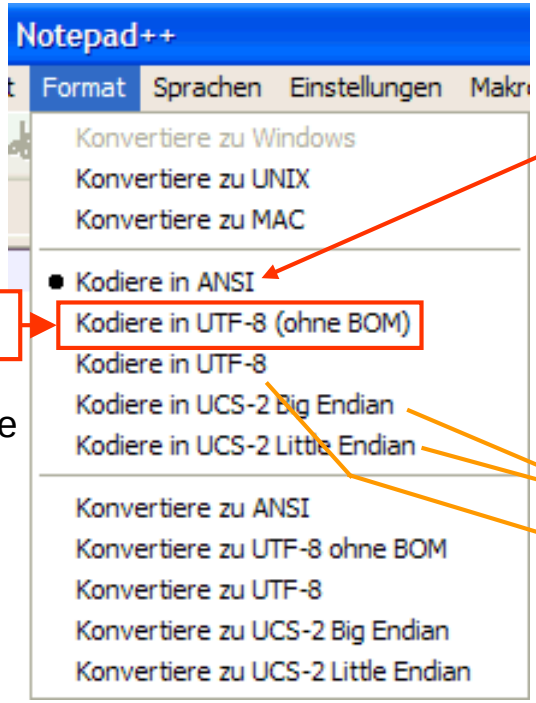
# Zeichenkodierung und Fonts

http://de.selfhtml.org/html/allgemein/zeichen.htm
http://de.selfhtml.org/inter/zeichenkodierungen.htm
http://www.unicode.org/charts/

- Zeichenkodierungen definieren die Abbildung Code → Zeichen
  - ⇨ Codepage 850 — 8E → Ä — *MS DOS*
  - ⇨ Codepage 1252 (Latin I) — C4 → Ä — *MS Windows (alt)*
  - ⇨ ISO-8859-1 — C4 → Ä — ähnlich ⇗
  - ⇨ Unicode — 00C4 → Ä — *MS Windows (neu)*
    - UTF-8 ist eine kompakte Unicode-Kodierung mit 1..4 Byte pro Zeichen
  - ⇨ benannte Zeichen in HTML — Ä → Ä — *reines ASCII*
  - ⇨ die Zeichenkodierung ist eine Eigenschaft der Datei und wird in XML und HTML im Header dokumentiert
- Fonts definieren die Abbildung Code → Aussehen
  - ⇨ OpenType und TrueType basieren auf Unicode
  - ⇨ Lucida Calligraphy — 00C4 → 𝒜̈
  - ⇨ Times New Roman — 00C4 → Ä
  - ⇨ verschiedene Textabschnitte können mit verschiedenen Fonts dargestellt werden; dies wird per CSS festgelegt

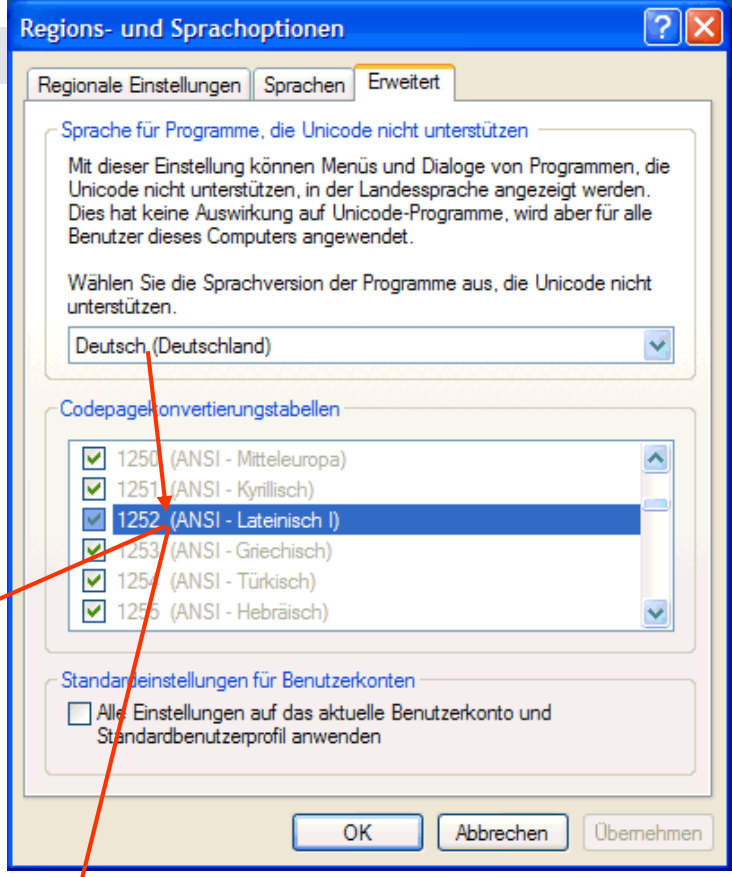
# Zeichenkodierung im Editor

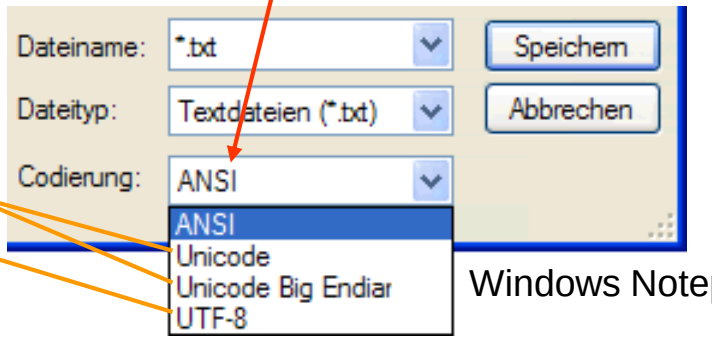
- die Datei muss auch wirklich mit der angegebenen Zeichencodierung erstellt sein
  - ⇨ Einstellung des Editors
  - ⇨ Default des Betriebssystems




BOM stört PHP

UCS-2 = Unicode




Windows Notepad

mit BOM (= Byte Order Mark) am Dateianfang
(optional bei UTF-8; zur Unterscheidung von ISO)

# Zeichenkodierung systemweit einheitlich

- vorzugsweise UTF-8 systemweit als Zeichenkodierung einsetzen
  - ⇨ Projekt von vorneherein mit UTF-8 aufsetzen
  - ⇨ nachträgliche Umstellung ist mühsam
  - ⇨ uneinheitliche Kodierung würde explizite Konvertierungen erfordern
- PHP-Dateien in UTF-8 ohne BOM (Byte Order Mark) kodieren
  - ⇨ BOM besteht aus Bytesequenz EF BB BF am Dateianfang
  - ⇨ BOM würde von PHP sofort ausgegeben, vor evtl. HTTP Header
  - ⇨ Achtung: eine UTF-8 Datei ohne BOM kann der Editor von einer ISO Datei nur unterscheiden, wenn sie auch tatsächlich Umlaute enthält (ggf. hilfsweise als Kommentar einfügen)
- Zeichenkodierung und Sortierreihenfolge gleich beim Anlegen der Datenbank festlegen
  - ⇨
```
CREATE DATABASE `db`
DEFAULT CHARACTER SET utf8
COLLATE utf8_unicode_ci;
```
  - ⇨ vorzugsweise einheitlich für alle Tabellen und Felder
- Zeichensatz für die Kommunikation zwischen PHP und Datenbank definieren
  - ⇨
```
$mysqli->set_charset("utf8");
```

# HTML Browser




- Es gibt eine Vielzahl verschiedener Browser in verschiedenen Versionen
  - ⇨ Die Unterstützung von "neueren" Features ist nicht sicher
    - Im Web gibt es diverse Seiten, welche die Umsetzung verfolgen z.B. http://caniuse.com oder http://html5readiness.com/
    - Für ältere Browser muss oft eine Rückfalllösung entwickelt werden
  - ⇨ Webseiten unbedingt für verschieden Browser testen
    - z.B. bei http://browsershots.org

# Audio element - Working Draft

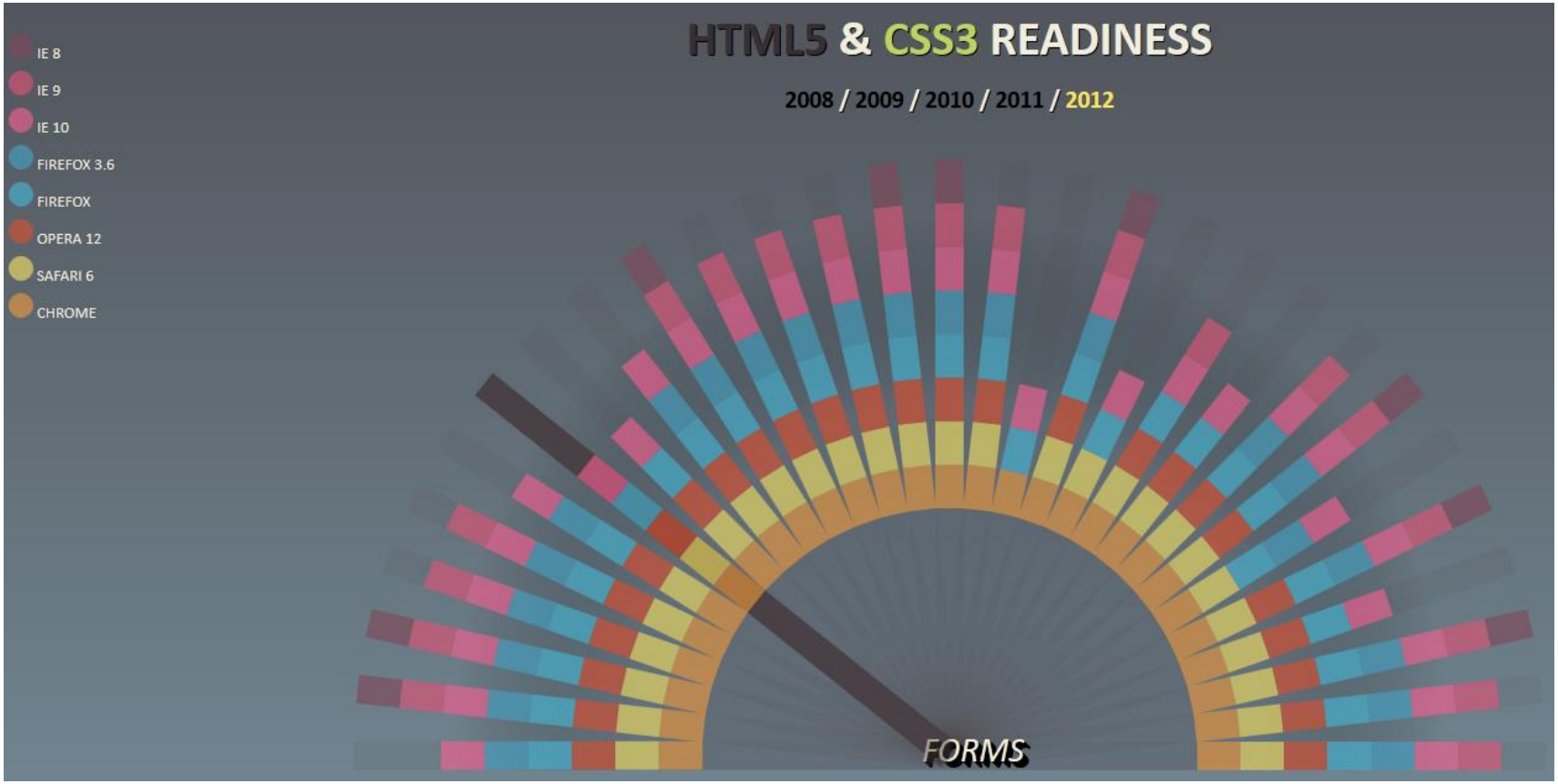
*Method of playing sound on webpages (without requiring a plug-in)*

| *Usage stats: | Global |
|---|---|
| Support: | 83.73% |
| Partial support: | 0.02% |
| Total: | 83.75% |

| Show all versions | IE | Firefox | Chrome | Safari | Opera | iOS Safari | Opera Mini | Android Browser | Blackberry Browser | IE Mobile |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | 2.1 | | |
| | | | | | | | | 2.2 | | |
| | | | | | | 3.2 | | 2.3 | | |
| | | | | | | 4.0-4.1 | | 3.0 | | |
| | 8.0 | | | | | 4.2-4.3 | | 4.0 | | |
| | 9.0 | | 31.0 | | | 5.0-5.1 | | 4.1 | | |
| | 10.0 | 26.0 | 32.0 | | | 6.0-6.1 | | 4.2-4.3 | 7.0 | |
| Current | 11.0 | 27.0 | 33.0 | 7.0 | 19.0 | 7.0 | 5.0-7.0 | 4.4 | 10.0 | 10.0 |
| Near future | | 28.0 | 34.0 | | 20.0 | | | | | |
| Farther future | | 29.0 | 35.0 | | 21.0 | | | | | |
| 3 versions ahead | | 30.0 | 36.0 | | | | | | | |

http://caniuse.com

# Stand der Umsetzung




http://html5readiness.com/

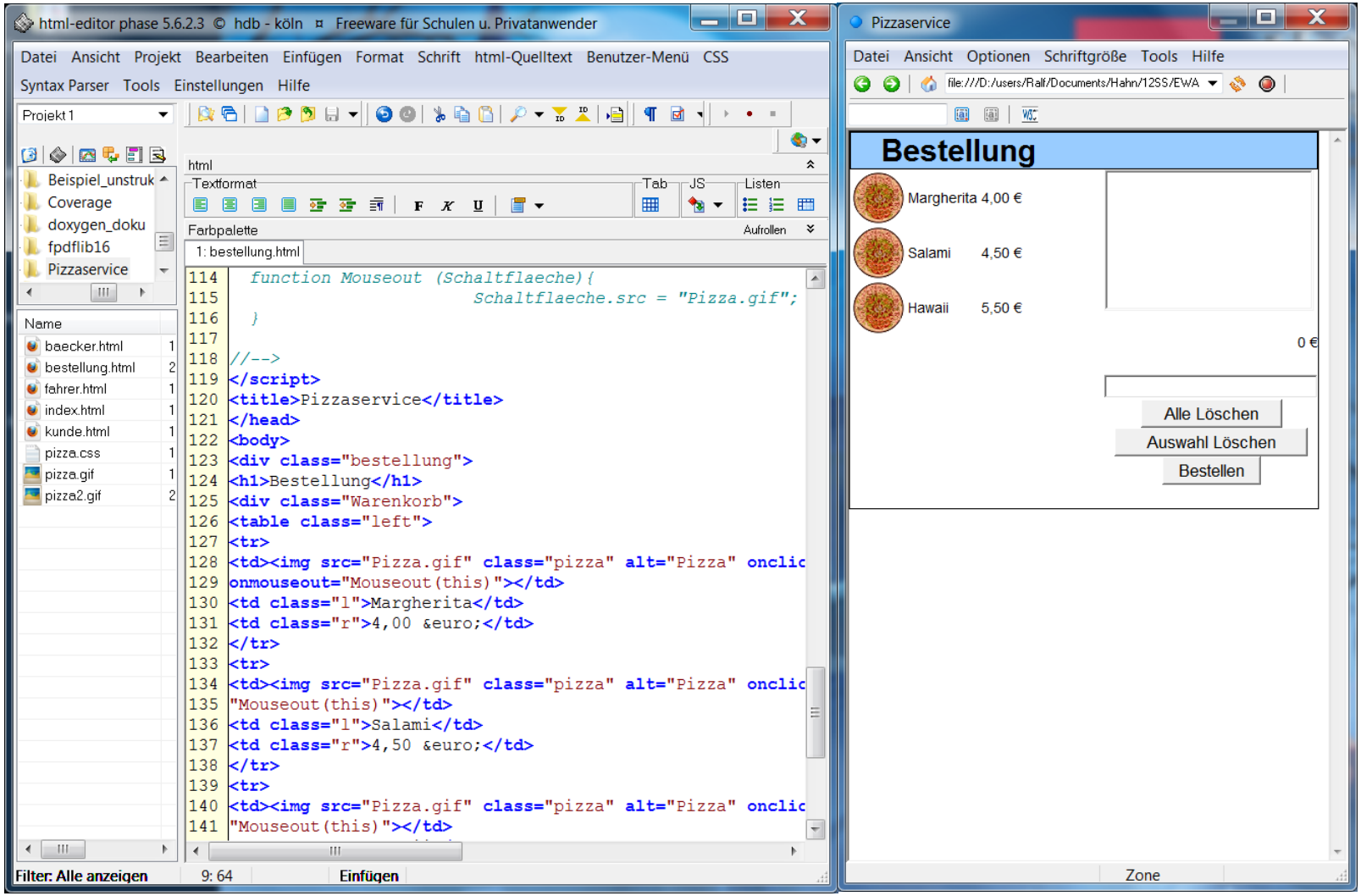
Oder auch: When can I use? http://caniuse.com

# HTML Editor

z.B. Phase 5

- vergleichbar mit komfortabler Programmierumgebung
  - ⇨ Hilfe beim Einfügen von HTML Code
    - übersichtliche, menügeführte Auswahl der HTML-Tags
    - Dialoge und Wizards für komplexere Auswahlen
  - ⇨ Auffinden durch syntaxabhängige Einfärbung unterstützt
  - ⇨ oft mit Preview des Ergebnisses
- aber: man editiert und sieht letztlich den HTML-Code
  - ⇨ der Autor denkt und kontrolliert visuell
  - ⇨ er muss Änderungswünsche in HTML "übersetzen" und die richtige Stelle im Code finden - wie ein Programmierer
- Vorteil:
  - ⇨ hand-optimierter HTML-Code, neueste Features nutzbar
- Nachteil:
  - ⇨ Programmierer muss die Schnittmenge der Browser finden

# HTML Editor - Beispiel

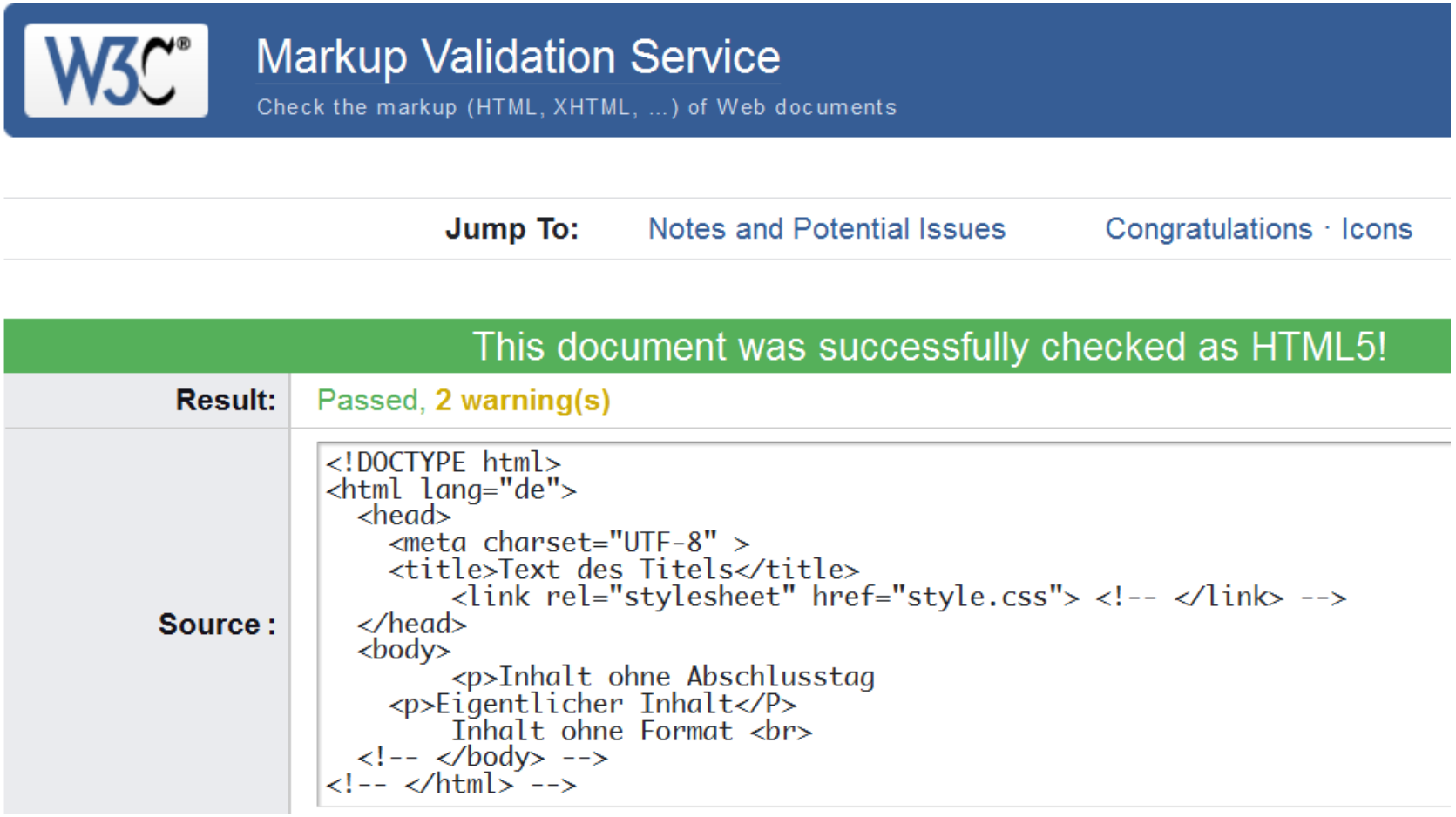


# HTML Prüfung

- Browser ignorieren normalerweise unbekannte oder falsche Tags und Attribute
  - ⇨ es gibt keine Fehlermeldung, allenfalls Fehlverhalten
  - ⇨ das Verhalten im Fehlerfall hängt stark vom Browser ab
- seit HTML5 sind ganz offiziell viele Konstrukte erlaubt, die in HTML 4 noch Fehler gewesen wären
  - ⇨ diverse (schließende) Tags sind optional
  - ⇨ unbekannte Attribute werden ignoriert
- deshalb: HTML Code vor der Veröffentlichung prüfen
  - ⇨ anhand der Spezifikation: Syntax, Tag- und Attributnamen, Schachtelungsregeln, etc. mit einem Validator (z.B. validator.w3.org)
  - ⇨ für HTML5 eventuell zusätzlich auf die Einhaltung von "Programmierrichtlinien" prüfen
  - ⇨ auch generiertes HTML (z.B. aus PHP) prüfen!

# HTML Validator – W3C

http://validator.w3.org




⇨ HTML5 wird validiert, obwohl es einige bedenkliche Konstrukte enthält

# WYSIWYG Tools

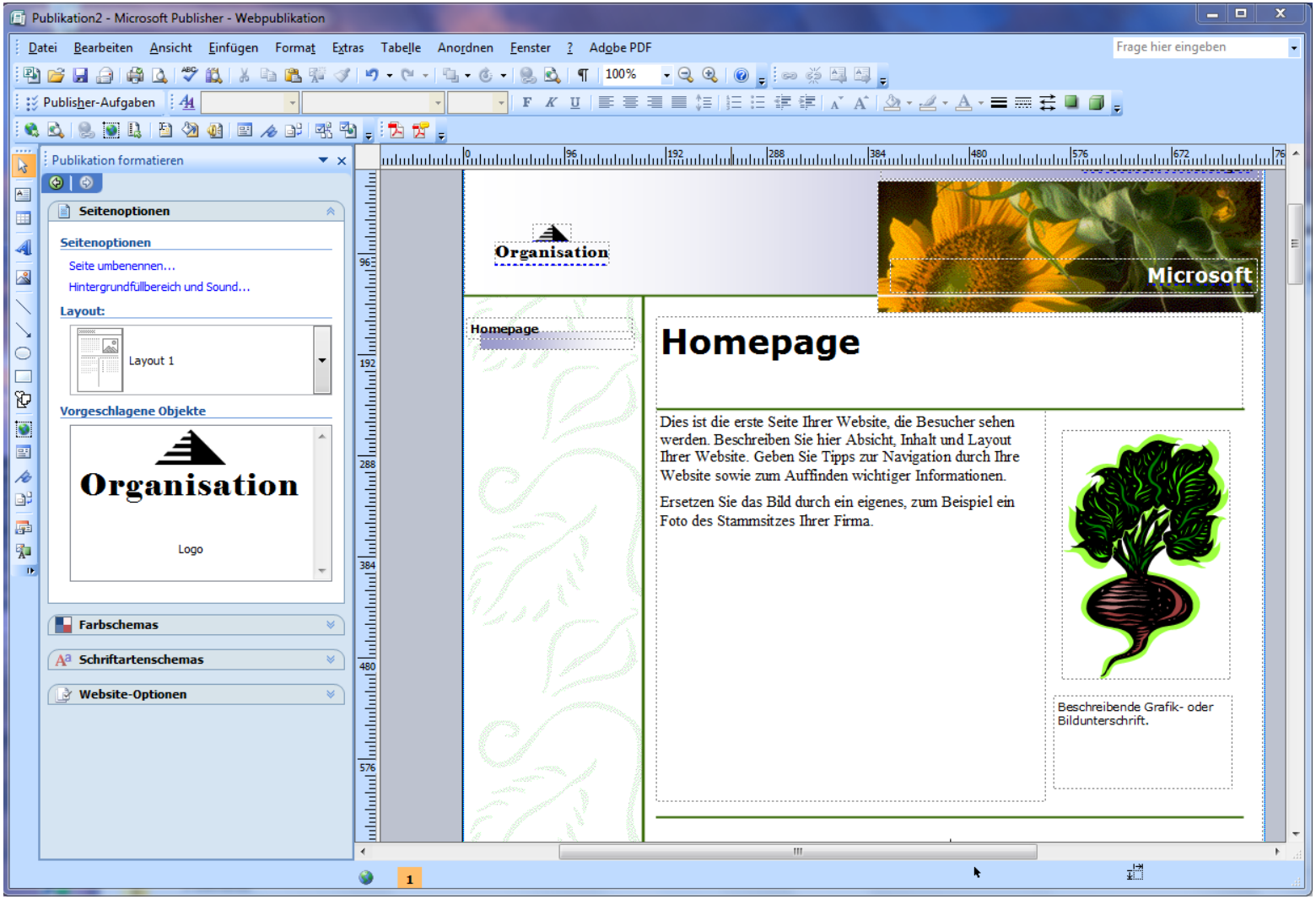
Dreamweaver, MS Publisher

- Hoher Komfort (ähnlich Word)
  - ⇨ HTML wird nicht mehr "programmiert"; Anweisungen und Attribute werden problemorientiert über Dialoge definiert
  - ⇨ HTML ist nur "internes Datenformat"
  - ⇨ HTML kann vom Autor betrachtet werden; muss aber nicht
- nur die Formatiermöglichkeiten von HTML sind erlaubt
  - ⇨ Tabellen und Grafikeinbindung gemäß HTML
- Darstellung etwa so wie im Browser
  - ⇨ eine mögliche WYSIWYG-Variante unter vielen
  - ⇨ zusätzlich Vorschau mit verschiedenen Browsern
- generiertes HTML ist meist "multi-browser-tauglich"

man muss die Prinzipien verstehen

da haben die Entwickler des Tools bereits getüftelt

# WYSIWYG Tool – Beispiel: Microsoft Publisher 2007



# Website "Explorer"

- Verwaltet Website mit allen zugehörigen Dateien
  - ⇨ Websites haben meist sehr viele kleine Dateien; darunter leidet leicht die Übersicht
  - ⇨ eigener problemangepasster "Explorer"
- Übersichtsdarstellungen
  - ⇨ einlaufende und abgehende Hyperlinks jeder Seite
  - ⇨ alle zugehörigen Bilder einer Seite
  - ⇨ alle Seiten, die ein Bild verwenden
- Konsistenzprüfung und Korrektur von Hyperlinks
  - ⇨ korrigiert Website-interne Hyperlinks bei Datei-Umbenennung
- Suchen und Ersetzen über gesamte Website

spezielle Problematik des Web

# Export aus anderen Tools

- früher unbrauchbar, mittlerweile erstaunlich gut
  - ⇨ z.B. PowerPoint für MSIE mit XML und Style Sheets
  - ⇨ z.B. MS Excel Leistungsübersicht im Online Belegsystem
  - ⇨ sogar in der Größe skalierbar
- generierter HTML-Code sehr komplex, viele Dateien
  - ⇨ praktisch schon fast wieder proprietäres Dateiformat
- nicht unterstützt: HTML-Rohform als Zwischenstufe bei Konvertierung existierender Dokumente
  - ⇨ Export für Nachbearbeitung in HTML Tool

# Zusammenfassung HTML

- HTML-Grundlagen
  - ⇨ Grundgerüst: DOCTYPE, <html>, <head>, <body>, <title>, charset...
  - ⇨ Schreibregeln und Syntax
  - ⇨ Tags und Attribute
  - ⇨ Hyperlinks
- Formulare
  - ⇨ Buttons, Listen, Datenübermittlung
- Werkzeuge

Jetzt wissen Sie alles um eine komplette und logisch strukturierte HTML-Seite zu entwickeln!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.1.5 HTML Layout

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Problematik des Layouts

- HTML beschreibt Fließtext
  - ⇨ absichtlich keine feste Platzierung
  - ⇨ Anordnung der Tags erfolgt nach der Reihenfolge in der HTML-Datei
  - ⇨ keine Überdeckung von Objekten
- Darstellung hängt vom System des Betrachters ab
  - ⇨ Egal ob Computermonitor, FullHD-Fernseher, Handy oder Netbook
  - ⇨ Informationsdarstellung mit sehr verschiedenen Auflösungen und Schriftgrößen (ggfs. mit automatischem Zoom)


# Problematik des Layouts (II)

- Dynamische (sich anpassende) Layouts sind schwierig zu entwerfen
  - ⇨ die meisten Seitengestalter denken in statischen Layouts
  - ⇨ traditionell sind in HTML zwei "Layoutmanager" verfügbar:
    `<table>` für Layoutzwecke verpönt
    `<frameset>` seit HTML5 verboten
  - ⇨ stattdessen wird heute CSS verwendet

- Entwurfsmethodik sinngemäß übertragen
  - ⇨ siehe "Dynamisches Layout" in "Entwicklung nutzerorientierter Anwendungen"

- Eine Tabelle ist (immer noch) häufig Basis des Seitenlayouts
  - ⇨ statisches Layoutraster durch Bemaßung in Pixel
  - ⇨ dynamisches Layoutraster durch Bemaßung in Prozent

# Tabelle als Layoutmanager

verpönt im Hinblick
auf Barrierefreiheit !

- Eine Tabelle ist (immer noch) häufig Basis des Seitenlayouts
  - ⇨ normalerweise „blinde“ Tabelle, d.h. ohne Rand
  - ⇨ Freiformen, Grafiken, Buttons etc. als eingebettete Elemente
  - ⇨ verhindert mögliche Layout-anpassungen wegen tabellarische Anordnung von nicht-tabellarischem Inhalt


  - ⇨ ein Screenreader liest die Tabelle von links nach rechts und von oben nach unten

# Layoutvorbereitung für die CSS-Formatierung

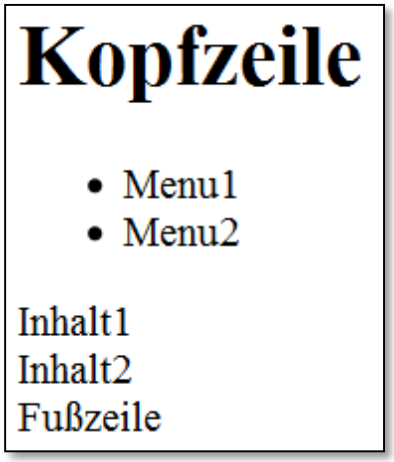
- In HTML wird eine Seite als inhaltlich logische Sequenz von Blöcken aufgebaut
  - ⇨ jeder Block wird mit einem Tag gekennzeichnet
    - mit einem passenden Standard-Tag z.B. `<h1>…</h1>`
    - oder sonst mit `<div>...</div>`
  - ⇨ Elemente, die speziell formatiert werden sollen, aber keinen Block erzeugen sollen, werden durch `<span>...</span>` gekennzeichnet
  - ⇨ die Reihenfolge innerhalb der HTML-Seite ist so gewählt, dass sie der logischen Reihenfolge entsprechen
  - ⇨ diese Sequenz definiert auch die Vorlesereihenfolge des Screenreaders

- Die einzelnen Blöcke werden dann später mit CSS formatiert, positioniert und ausgerichtet
  - ⇨ z.B. CSS-Attribute für den Textfluss: `float, margin, clear`
  - ⇨ www.fbi.h-da.de ist so aufgebaut

# Layoutvorbereitung für die CSS-Formatierung - Beispiel

```
<header><h1>Kopfzeile</h1></header>
<nav><ul>
  <li>Menu1</li> <li>Menu2</li>
</ul></nav>
<section>
   <article>Inhalt1</article >
   <article>Inhalt2</article >
</section>
<footer>Fußzeile</footer>
```

Nicht gerade schön – aber logisch genau das, was wir mit HTML wollen!



# Zusammenfassung

- Problematik des Layouts
  - ⇨ WYSI(not)WYG
  - ⇨ Barrierefreies Layout
- Layout in HTML unerwünscht
  - ⇨ Tabelle für Layout-Zwecke
  - ⇨ Nachteile mit Screenreader
- Vorbereitung für die CSS-Formatierung

Jetzt beherrschen Sie die Grundlagen von HTML und können die Elemente auf einer HTML-Seite anordnen!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2 CSS

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK

# HTML ohne CSS (Beispiel Zen Garden ohne CSS)

## css Zen Garden

### The Beauty of CSS Design

A demonstration of what can be accomplished visually through CSS-based design.

**The Road to Enlightenment**

Littering a dark and dreary road lay the past relics of browser-specific tags, incompatible DOMs, and broken CSS support.

**So What is This About?**

There is clearly a need for CSS to be taken seriously by graphic artists. The Zen Garden aims to excite, inspire, and encourage participation. To begin, view some of the existing designs in the list. Clicking on any one will load the style sheet into this very page. The code remains the same, the only thing that has changed is the external .css file. Yes, really.

**Requirements**

We would like to see as much CSS1 as possible. CSS2 should be limited to widely-supported elements only. The css Zen Garden is about functional, practical CSS and not the latest bleeding-edge tricks viewable by 2% of the browsing public.

Unfortunately, designing this way highlights the flaws in the various implementations of CSS.

xhtml css

**Select a Design:**

- Under the Seal by Eric Stoltz
- Make 'em Proud by Michael McAghon and Scotty Reifsnyder

**Archives:**

- next designs »
- View All Designs

**Resources:**

- View This Design's CSS
- CSS Resources

# Beispiel (Zen Garden mit CSS)






Quelle: http://www.csszengarden.com

siehe auch http://www.oswd.org

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2.1 CSS Grundlagen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Cascading Style Sheets

| CSS Version | |
|---|---|
| 1.0 | 1996 |
| 2.0 | März 1998 |
| 2.1 | Juni 2011 |
| 3.0 | Draft 2012 |

- Definition physischer Attributsätze für
  - ⇨ vordefinierte logische Formate (Überschrift 2, Aufzählung, ...)
  - ⇨ selbstdefinierte Format-Klassen
  - ⇨ einzelne (Text-)Blöcke

vergleichbar mit Formatvorlagen in Word

- vielfältige physische Attribute
  - ⇨ Schriftart, -größe, -stil, Zeichen- und Zeilenabstand, Einrückung
  - ⇨ Text- und Hintergrundfarbe, Rahmen
- Seitengestaltung für Druck
- freie Platzierung und Überlappung von Objekten
  - ⇨ vgl. Autorensysteme
  - ⇨ Basis für Animation von Objekten

# Potential der CSS

- exakte Bestimmung des Erscheinungsbilds
  - ⇨ wichtiger Schritt in Richtung WYSIWYG
  - ⇨ die variable Bildschirmauflösung bleibt ein Problem
- verbesserte Exportierbarkeit aus anderen Tools
  - ⇨ Erstellung von Webseiten mit gewohnten Tools
  - ⇨ kein Fachwissen und keine Tricks mehr erforderlich; HTML-Programmierer werden arbeitslos
- Optimierung für verschiedene Ausgabemedien
  - ⇨ Bildschirm, Drucker, TV, Handy, Screenreader...
- Grundlage für Barrierefreies Webdesign

# Arbeitsteilung mit CSS

- saubere Trennung zwischen Inhalt und Form
  - ⇨ Inhalt logisch formatiert in HTML
  - ⇨ Physisches Format und Fein-Layout separat in CSS
- Arbeitsteilung Autor / Designer wird möglich
  - ⇨ einheitliche Layouts für große Projekte
- Corporate Design kann übernommen werden
  - ⇨ hierarchischer Aufbau (Kaskadierung)
  - ⇨ Übernahme und Abwandlung einer Designvorgabe

# Einbindung von CSS in HTML (1)

- "extern" in eigener CSS-Datei
  - ⇨ kann von mehreren HTML-Dateien genutzt werden

```
<link rel="stylesheet" type="text/css"
                       href="datei.css" />
```

Normalfall

gehört in den HTML-<head>

- "eingebettet" im HTML-Code
  - ⇨ gilt nur für diese eine HTML-Datei

```
<style>

    /* ... Style-Sheet-Definitionen ... */

</style>
```

CSS-Kommentar

# Einbindung von CSS in HTML (2)

- "inline" in jedem HTML-Tag
  - ⇨ gilt nur für dieses eine Objekt

```
<p style="color:red; font-size:36pt;">
     großer roter Text</p>
```

  - ⇨ HTML Inline-Tag `<span>` zur Markierung eines Teilbereich eines Objekts

primärer Zweck

```
<p>Normaler Textabsatz mit
   <span style="font-style:italic; color:red;">
      rot-kursivem Text
   </span> und wieder normal
</p>
```

# Unterstützung verschiedener Ausgabemedien

- verschiedene Bereiche innerhalb eines CSS

```
@media screen {
  /* Style-Sheet-Definitionen für den Bildschirm */
}
@media print {
  /* Style-Sheet-Definitionen zum Drucken */
}
```

- verschiedene CSS-Dateien in HTML einbinden

```
<link rel="stylesheet" media="screen"
                       href="Bildschirm.css" />
<link rel="stylesheet" media="print"
                       href="Drucker.css" />
<link rel="stylesheet" media="speech"
                       href="Screenreader.css" />
```

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2.2 CSS - Formate definieren

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Browser-Default-Formatierung im Vergleich - ohne CSS

- Browser haben unterschiedliche Default-Stile
  - ⇨ hier Internet Explorer 6 und Firefox 2


Um ein definiertes Layout zu erhalten,
muss man die Standard-Formate selbst definieren!

# Standard-Formate modifizieren

- Definition in CSS

vorzugsweise für Ausgestaltung logischer Formate

```
h3  { text-align: center; color:#33FF00; }
p   { border: 1px solid black;
      font-family:Arial, Helvetica; }
*   { color:green; } /* Universalselektor gilt für alle Tags*/
ul  { list-style:none; } /* verbirgt die Aufzählungspunkte */
```

- Anwendung in HTML

kein Attribut in HTML

```
<h3>Überschrift 3. Ebene</h3>
<p>einfacher Fließtext in einem Absatz</p>
```

⇨ ohne CSS zeigt der Browser die "schlichte" Version

# Standard-Formate kontextabhängig

- Definition in CSS

```
h1   { color:red; }
h1 i { color:blue; font-weight:normal; }
```

d.h. Italic geschachtelt in Header 1

dieses Format gilt nur dort

- Anwendung in HTML

```
<h1>Eine Überschrift mit <i>Style-Sheets</i></h1>
<p>Ein Fließtext mit <i>Style-Sheets</i></p>
```

nicht hier

**Eine Überschrift mit *Style-Sheets***

Ein Fließtext mit *Style-Sheets*

# Eigene Format-Klassen

- Definition in CSS
  - ⇨ Unterklassen für Standard Formate

```
p.Hinweis { color:red; }
p.Fussnote{ color:black; }
```

  - ⇨ allgemein verwendbar

```
.Warnung  { color:#DC0081 }
.Zitat    { color:#00DFCA }
```

aber keine Klasse mit Ableitung etc. wie in OO

- Anwendung in HTML

```
<p class="Hinweis">beachten Sie bitte</p>
<p class="Fussnote">das nur am Rande</p>
<p class="Warnung">Achtung! Aufpassen!</p>
<blockquote class="Zitat">des Pudels Kern
    </blockquote>
```

HTML Attribut `class` stellt den Bezug her

# Individuelle Objekt Formate

- Definition in CSS

```
#Block1   { font-weight:bold; font-style:italic; }
#Hotw3    { text-decoration:underline; }
```

- Anwendung in HTML
  - ⇨ jedes Format und jede `id` nur einmal !

Eindeutige Block-IDs kann man auch für JavaScript brauchen

```
<p id="Block1">Extra-Formatierung</p>
<p>Einfacher Text mit <em id="Hotw3">Hotword</em></p>
```

"Hotw3" ergänzt das Format von ; im Konfliktfall mit Vorrang

# Pseudo-Formate

- Sonderfall:
  - ⇨ definieren Eigenschaften, die keine Attribute von HTML-Blöcken sind

- Darstellung von Hyperlinks - Festlegung in CSS

```
a:link      { color:blue;}    /* normaler Link        */
a:visited   { color:green;}   /* bereits besucht      */
a:active    { color:red;}     /* gerade angeklickt    */
a:hover     { color:yellow;}  /* unter dem Mauszeiger */
a:focus     { color:black;}   /* mit Tastatur angewählt */
```

```
p:first-line   { font-weight:bold; }
p:first-letter { font-size:large; color:red;}
```

**M**an kann nur Brücken schlagen zwischen Ufern die man auseinanderhält. Denn wo es keine Gräben gibt, da gibt es auch keine Unterschiede, und wo es keine Unterschiede gibt, da ist kein Leben.

# Weitere Selektoren (eine kleine Auswahl)

- Flexible Möglichkeit, um Tags auszuwählen:
  - ⇨ `[target]` Wählt alle Tags mit einem Attribut target
  - ⇨ `[target=xyz]` Wählt alle Tags mit einem Attribut target, das den Wert xyz hat

- Auswahl mit Parametern
  - ⇨ `p:nth-child(2)` Wählt alle p-Tags, die das 2-te Kind von irgendeinem Tag sind
  - ⇨ `p:nth-child(3n+1)` Wählt alle p-Tags, die das 1-te, 4-te, 7-te,.... Kind von irgendeinem Tag sind
  - ⇨ Anwendung für "gestreifte" Tabellen (`even` und `odd` sind vordefiniert):
    - `tr:nth-child(even) { background-color: LightGrey; }`
    - `tr:nth-child(odd) { background-color: white; }`

Es gibt noch viele weitere Selektoren!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2.3 CSS - Attribute

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Farben und Hintergrundbilder

Farbschema entwerfen mit
http://kuler.adobe.com

- Farbattribute

| | |
|---|---|
| `background-color` | Hintergrundfarbe |
| `color` | Textfarbe |
| `border-color` | Rahmenfarbe |
| `text-shadow` | schattierter Text |

```
background-color: white;
```

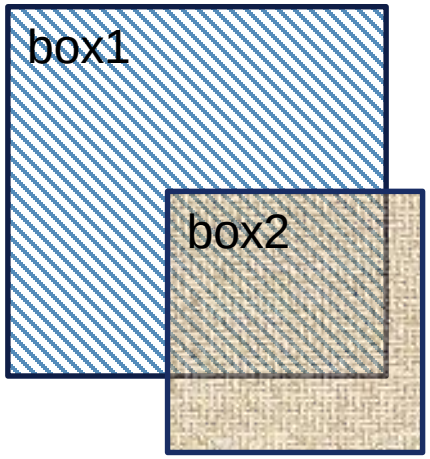
- Notationen für Farbwerte

| | |
|---|---|
| `rgb(255,140,0)` | Farbanteile für rot, grün, blau im Bereich 0..255 |
| `rgb(100%,55%,0%)` | Farbanteile im Bereich 0%..100% |
| `#FF8C00` | Farbanteile hexadezimal |
| `Darkorange` | diverse Farben mit Namen |

- Hintergrundbild
  - ⇨ nicht nur für gesamte Seite, sondern auch für einzelne Blöcke

```
background-image:url(bild.gif)
```

# Transparenz




- Durchsichtige Elemente
  - ⇨ `opacity: 0.5;`
  - ⇨ Einstellung über eine Zahl zwischen 0 (voll transparent) und 1 (volle Deckkraft)
  - ⇨ Macht das gesamte Element mit Rahmen, Hintergrund etc. transparent
  - ⇨ Ein- und Ausblendeffekte (mit Javascript)

- Transparent als „Farbe“
  - ⇨ `background-color: transparent;`
  - ⇨ Macht nur das jeweilige Element transparent

# Schrift

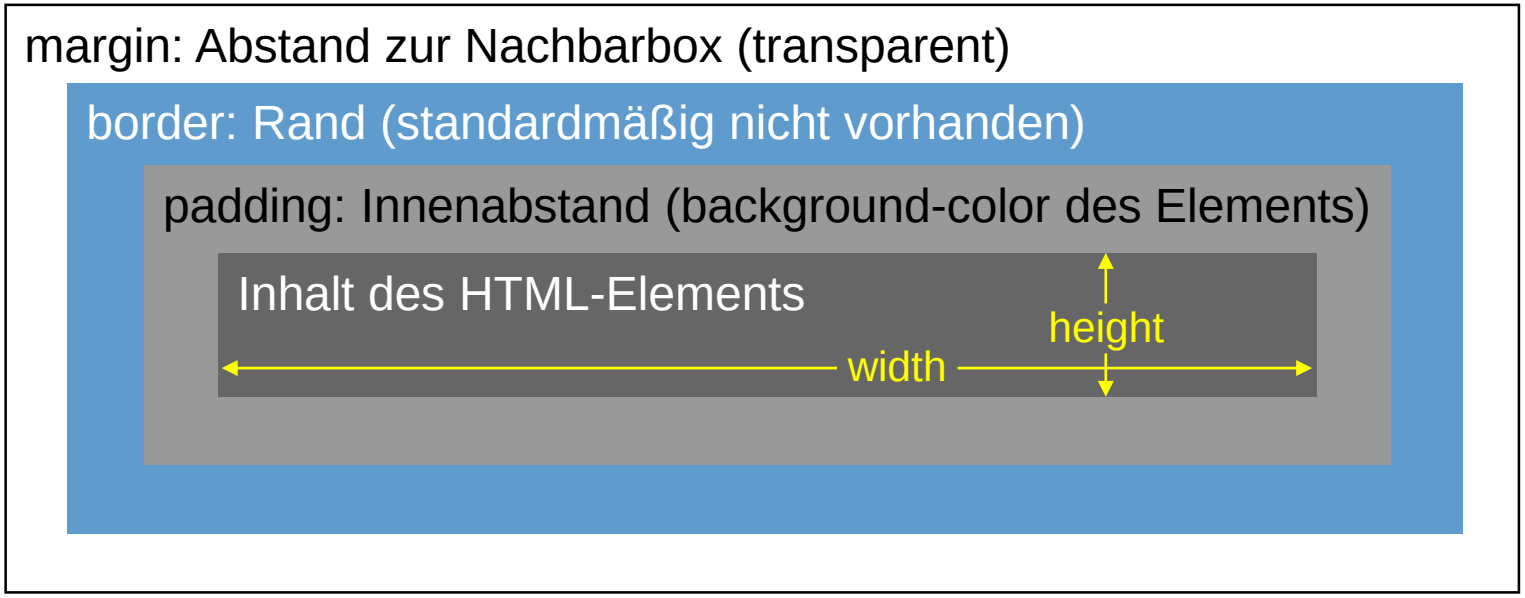
- font-family:
  - ⇨ Arial, Helvetica, "Times New Roman"
  - ⇨ serif, sans-serif, cursive, fantasy, monospace
- font-style:
  - ⇨ italic, normal
- font-size:
  - ⇨ 12pt, 35px, 3em, 1.5cm, large
- font-weight:
  - ⇨ bold, bolder, lighter, 100 .. 900
- font:
  - ⇨ kompakte Kombination o.g. Attributwerte

Die Bedeutung der verschiedenen Maßeinheiten kommt im nächsten Abschnitt!

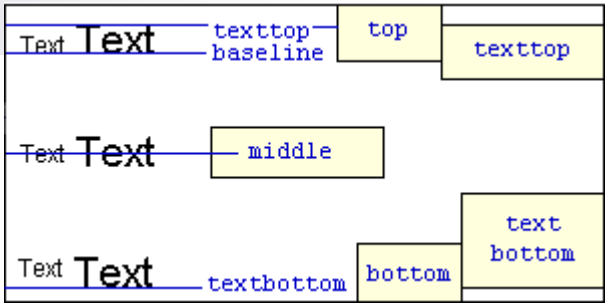
# Aussen- und Innenabstand

die Standardwerte sind browserabhängig, deshalb vollständig spezifizieren!

- `margin, margin-top, margin-bottom, margin-left, margin-right` Aussenabstand in Längenmaß
- `padding, padding-top, padding-bottom, padding-left, padding-right` Innenabstand in Längenmaß
- Achtung: `width` und `height` beziehen sich auf den Inhalt !



# Ausrichtung und Rand




Quelle: SelfHTML

- Ausrichtung
  - ⇨ `line-height` Zeilenhöhe in Längenmaß
  - ⇨ `text-indent` Texteinrückung in Längenmaß
  - ⇨ `text-align: left, center, right, justify` (Blocksatz)
  - ⇨ `vertical-align: top, middle, bottom, text-top, text-bottom`
- Rand
  - ⇨ `border[-top, -left, -right, -bottom]-width`
    `(z.B. border-left-width, border-width)`
  - ⇨ `border[-top, -left, -right, -bottom]-style:`
    `hidden, dotted, dashed, solid, double, groove, ridge, inset, outset`
- Abgerundete Ecken
  - ⇨ `border[-top, -bottom][-left, -right]-radius: x radius y radius`
  - ⇨ `z.B. border-top-left-radius: 3em 2em` oder `border-radius: 5%`

# Erscheinungsbild einer Tabelle

| Spalte A | Spalte B | Spalte C |
|---|---|---|
| Even | Even | Even |
| Odd | Odd | Odd |
| Even | Even | Even |

- Tabellen werden mit den üblichen Elementen formatiert:
  - ⇨ `width, height, padding, border, margin,`
  - ⇨ `text-align, vertical-align`
- Üblicherweise werden die Linien angeschaltet
  - ⇨ `table, th, td { border: 1px solid black; }`
  - ⇨ dann hat aber jede Zelle einen Rahmen, d.h. die Linien sind doppelt

    Lösung: `border-collapse: collapse;`
- Beispiel: Gestreifte Tabelle

```
tr:nth-child(even)    { background-color: LightGrey; }
tr:nth-child(odd)     { background-color: white; }
th {background-color:black; color:white; }
table, th, td { border: 1px solid black;
                        border-collapse:collapse;}
```

# Zeigen und Verbergen

Auf- und Zuklappen von Unterpunkten im Inhaltsverzeichnis mit JavaScript

- Anzeige(-art) bzw. Nichtanzeige ohne Platzhalter (folgende Blöcke verschieben sich)

  `display:`

  | | |
  |---|---|
  | `inline` | Element wird im laufenden Textfluss angezeigt. Der Text "fließt" in Lücken, welche die anderen Elemente bieten |
  | `block` | Rechteckig begrenztes Element steht alleine in einer Zeile |
  | `inline-block` | Rechteckig begrenztes Element, das als Block im Textfluss bleibt (siehe folgendes Beispiel) |
  | `none` | Element wird nicht angezeigt, folgende Blöcke verschieben sich |

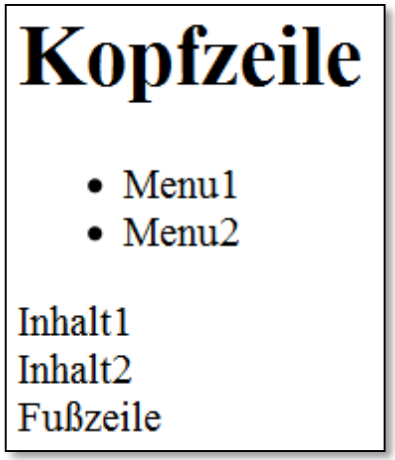
- Anzeige bzw. Nichtanzeige mit Platzhalter

  `visibility:`

  | | |
  |---|---|
  | `visible` | Element wird angezeigt |
  | `hidden` | Element wird versteckt (folgende Blöcke bleiben stehen) |

# Beispiel: HTML mit leerer CSS-Datei

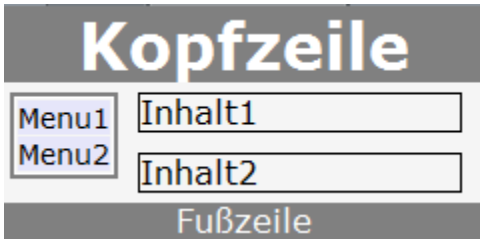
```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="UTF-8" />
    <title>CSS-Vorbereitung</title>
  </head>
  <body>
    <header><h1>Kopfzeile</h1></header>
    <nav><ul>
      <li>Menu1</li> <li>Menu2</li>
    </ul></nav>
    <section>
      <article>Inhalt1</article >
      <article>Inhalt2</article >
    </section>
    <footer>Fußzeile</footer>
  </body>
</html>
```



# Beispiel: Gleiche HTML-Datei mit einfachem CSS

Achtung!
Nicht das Semikolon oder die Klammern vergessen. Sonst funktioniert es nicht!

```
* { padding:0em; margin:0em; } /* keine Default Abstände */
body {color:black; background-color: WhiteSmoke;
      font:1em Verdana;}
footer, header {clear: both; text-align:center;
      color: white; background-color: grey;}
article {display: inline-block; width: 15em;
      border: 0.1em solid black; margin:0.5em;}
nav {display: block; margin: 0.5em; border: 0.2em solid grey;
      text-align:center; float:left;}
nav li {font-size: 1em; margin: 0.1em;
      background-color: Lavender;}
ul {list-style:none;
      border: 0.1em solid white;
      text-align: left;}
```




# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2.4 Maßeinheiten

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Absolute Maßeinheiten für statisches Layout

seit iPhone 4 ("Retina-Display") kann man nicht mehr davon ausgehen, dass der Benutzer ein einzelnes Pixel erkennen kann !

- px (Pixel)
  - ⇨ für Darstellung am Bildschirm
  - ⇨ verbreitet für Bilder und eingebettete Objekte
    - vermeidet Skalierung (hässliche Artefakte bei älteren Browsern)
- pt (point), cm, mm, in (inch)
  - ⇨ für Druckausgabe mit fester Papiergröße
  - ⇨ pt üblich für Schriften
  - ⇨ cm, mm, in für Positionen und Abstände
- Umrechnung
  - ⇨ 1 inch = 1 Zoll = 2.54 cm
  - ⇨ 1 pt = 1/72 inch = 0.0352778 cm; die echte Größe auf dem Bildschirm ist aber abhängig von der Bildschirmauflösung:
    - Windows 96 dpi oder 120 dpi, Mac und Java 72 dpi
    - reine Rechengröße, berücksichtigt nicht die realen Maße des Bildschirms

dots per inch = Pixel pro Zoll

# Relative Maßeinheiten für dynamisches Layout

eigentlich zu bevorzugen

- % prozentualer Anteil bezogen auf
  - ⇨ *für font-size:* die font-size des umschließenden Blocks
    - Sonderfall für <body>: die im Browser wählbare Schriftgröße
  - ⇨ *für width und height:* die Breite bzw. Höhe des umschließenden Blocks
    - Sonderfall für <body>: des Innenbereichs des Browserfensters
  - ⇨ *für margin und padding:* die Breite bzw. Höhe des eigenen Blocks
- em, ex Höhe von M bzw. x der elementeigenen Schrifthöhe
  - ⇨ *für font-size:* die font-size des umschließenden Blocks
    - Sonderfall für <body>: die im Browser wählbare Schriftgröße
  - ⇨ *für andere Attribute:* Wert von font-size des selben Elements
    - ermöglicht z.B. padding / margin abhängig von der Schriftgröße
- rem Höhe von M im root-Element
  - ⇨ in der Regel das Body-Tag mit Schriftgröße von 16pt
  - ⇨ ermöglicht relative Größenfestlegungen unabhängig von der Verschachtelungstiefe und vereinfacht die Berechnung

# Zoom statt Wahl der Schriftgröße

vgl. mit Firefox und Internet Explorer:
http://www.csszengarden.com Oceanscape
http://www.tagesschau.de/

- historische Situation (Hintergrund des Standards)
  - ⇨ Skalierung von Bildern wurde in Browsern schlecht unterstützt (Nearest Neighbour Resampling mit hässlichen Artefakten)
  - ⇨ Skalierung der Seite wurde realisiert durch Skalierung des Textes mit neuem Umbruch; Bilder blieben unverändert
- heutige Situation
  - ⇨ gute Skalierung von Bildern kein Problem dank hoher Rechenleistung
  - ⇨ Benutzer erwarten Skalierung von Bildern und Texten
- Lösung: Einstellbarkeit der Schriftgröße wird ersetzt durch Zoom
  - ⇨ Firefox und Internet Explorer zoomen alle Maßeinheiten gleichartig
    - die Schriftgröße wird geändert und das Layout ggf. neu umgebrochen
    - geht auch für Seiten mit statischem Layout (erfordert i.a. horizontales Scrollen)
    - ist perfekt für Seiten mit dynamischem Layout (kein horizontales Scrollen)
  - ⇨ Internet Explorer bietet zusätzlich eine Einstellung für die Textgröße; bei Firefox heißt das Äquivalent "Nur Text zoomen"
    - betrifft standardkonform nur %, em, ex
    - Bilder gehen nicht mit in der Größe
    - nur akzeptabel für Seiten mit dynamischem Layout

leidiges Problem:
viele Webentwickler haben absolute und relative Maße nicht sinnvoll eingesetzt;
viele Seiten skalierten schlecht

entschärft o.g. Problem

# Empfehlungen für dynamische Layouts

dagegen für Druckausgabe alles in **pt** festlegen

- vgl. "Entwicklung nutzerorientierter Anwendungen"
  Parameter dynamischer Layouts:
  Fenstergröße, Schriftgröße, Textlänge

- Schriftgröße in %, em oder rem (benutzerdefinierbar via Browser)
- Ränder, Außen- und Innenabstände in em (passend zur Schrift)
- Bilder in em (passend zur Schrift)
- Blockgröße
  - ⇨ fensterabhängig in % (falls Umbruch möglich) oder
  - ⇨ schriftabhängig in em (falls zu klein für Umbruch)
- Blockposition
  - ⇨ vorzugsweise fließend (siehe nächster Abschnitt)
  - ⇨ abwägen: fensterabhängig in % oder schriftabhängig in em
  - ⇨ möglichst nicht absolut positionieren

suboptimal:
alles in **px** festlegen und sich auf die Zoom-Funktion moderner Browser verlassen

generell absolute und relative Maßeinheiten nicht mischen !

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2.5 CSS - Layout

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Attribute für das Layout

Bausteine für Barrierefreies Webdesign

- Block aus dem Textfluss herausnehmen
  - ⇨ `float:`
    `left, right, none`

verlangt Festlegung von `width`

- Fortsetzung unterhalb eines `float`-Blocks
  - ⇨ `clear:`
    - `left` unterhalb des letzten links ausgerückten Blocks
    - `right` unterhalb des letzten rechts ausgerückten Blocks
    - `both` unterhalb des letzten ausgerückten Blocks
- wenn der Inhalt größer ist als der Block
  - ⇨ `overflow:`
    - `visible` Blockgröße passt sich an
    - `hidden` Inhalt beschneiden
    - `scroll` Inhalt verschiebbar mit Scroll-Balken (immer sichtbar)
    - `auto` Scroll-Balken nur bei Bedarf

# Breiten- und Höhenangaben

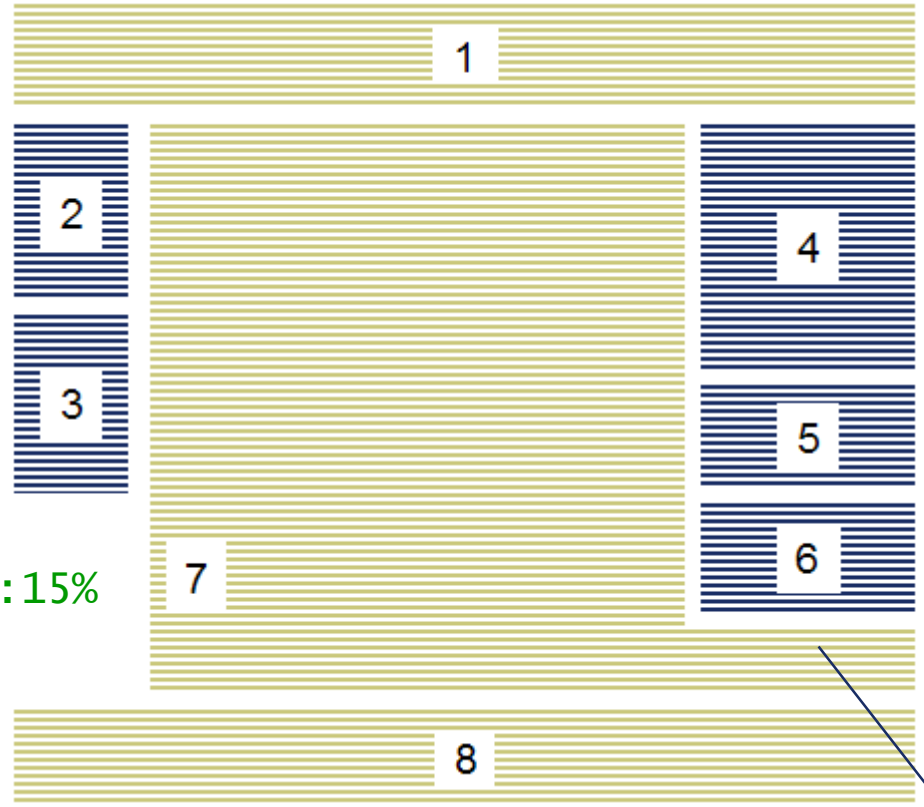
mit `overflow` festlegen, was mit überstehenden Elementen passieren soll

- Breitenangaben
  - ⇨ `width: 15%; /* relativ zur Umgebungsgröße */`
  - ⇨ `min-width: 30em;`
    z.B. falls das <body>-Tag eine bestimmte Mindestgröße haben soll; unterhalb `min-width` wird die gesamte Seite gescrollt
  - ⇨ `max-width: 50em;`
    z.B. falls eine Randspalte eine bestimmte Maximalgröße haben soll

- Höhenangaben
  - ⇨ `height: 40em;`
  - ⇨ `min-height: 30em;`
  - ⇨ `max-height: 50em;`

min-... und max-... in % wäre sinnlos

# Layout-Prinzip: 3-spaltig mit Kopf- und Fußzeile




Klammer um Randblöcke:

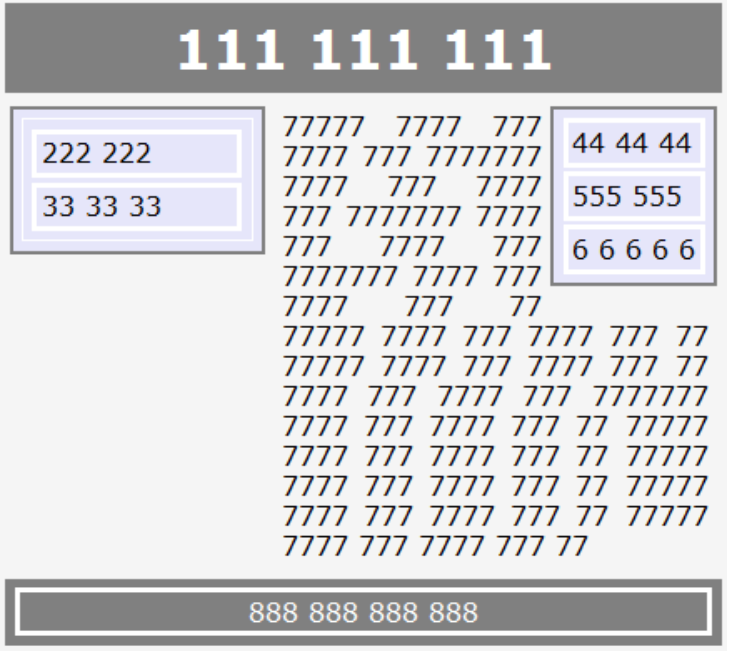
```
<div style=
     "float:right">
  <p id="Block4">
  </p>
  <p id="Block5">
  </p>
  <p id="Block6">
  </p>
</div>
```

Das "Umfließen" klappt nur, wenn der Abschnitt zeilenweise fließt

# Beispiel: HTML 3-"spaltig" mit Kopf-, Menü und Fußzeile

```
...
<body>
    <header><h1>111 111 111</h1></header>
        <nav><ul>
          <li>222 222</li>
          <li>33 33 33</li>
        </ul></nav>

        <div class="Rechts">
                <p>44 44 44</p>
                <p>555 555</p>
                <p>6 6 6 6 6</p>
        </div>
        <section>
           <article>77777 7777 777 … 777 7777777 77</article >
        </section>
        <footer ><p>888 888 888 888</p></footer>
  </body>
```



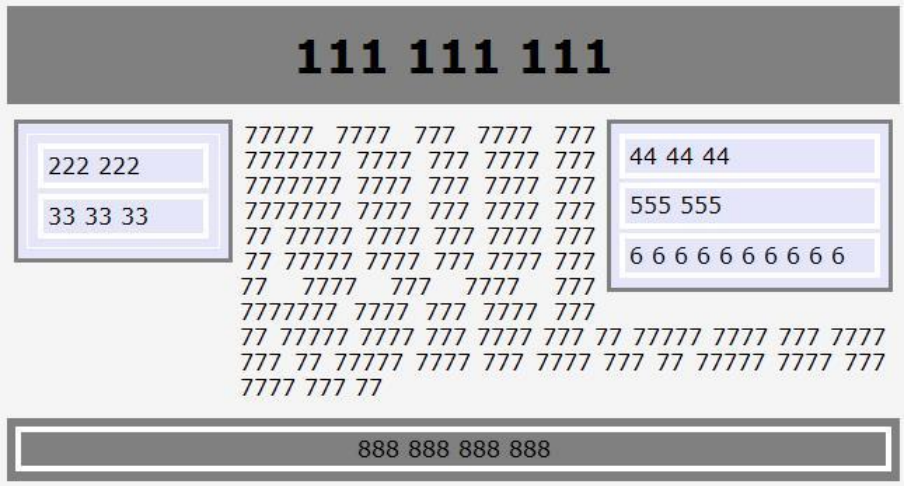
# Beispiel: CSS 3-spaltig mit Kopf-, Menü und Fußzeile

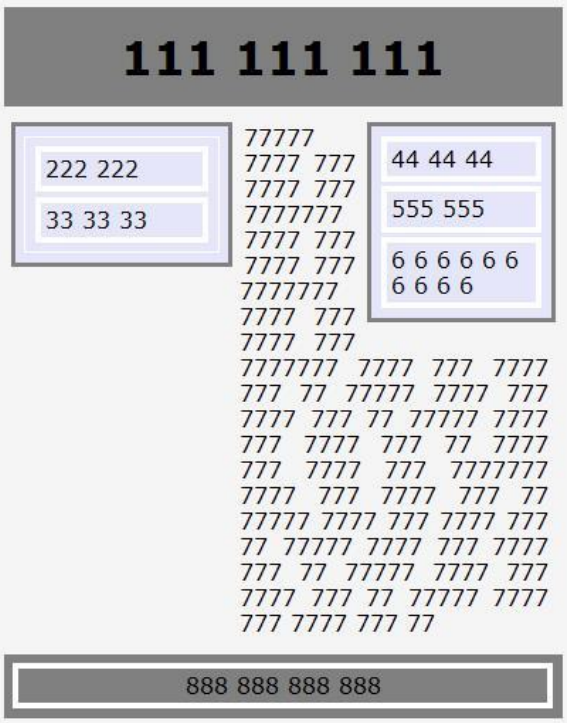
```
/* Design*/
* {margin: 0.2em; padding: 0.2em; color:black;}
body { color:black; background-color: WhiteSmoke; font:1em Verdana;}
footer, header {color: white; background-color: grey;
                          text-align:center;}
nav, .Rechts {margin: 0.5em; border: 0.2em solid grey;
                          background-color: Lavender;}
li, p {border: 0.3em solid white;}
ul {list-style:none; border: 0.1em solid white; text-align: left;}

/* Layout */
body {min-width:25em;}
footer, header {clear: both;}
nav {display: block; float:left; width: 9em; min-width:5em;}
article {text-align: justify; margin-left:10em; }
.Rechts {width: 30%; float: right; display: block; max-width: 12em;}
```

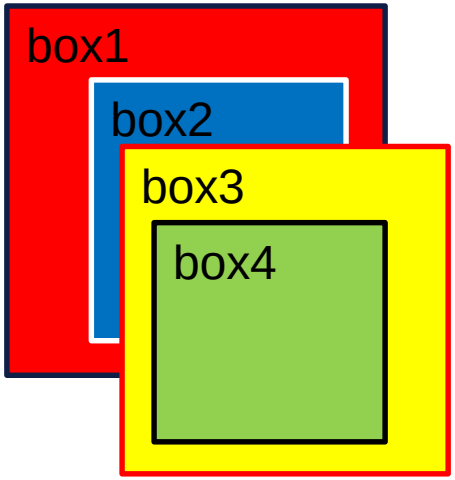
# Beispiel: Das Ergebnis in verschiedenen Fenstergrößen

- Kopf- und Fußzeile
- Menüliste als "Buttons" in einer Zeile
- Der mittlere Inhaltsbereich (7er) umfließt den rechten Block
- Der rechte Block passt die Breite bis zu einer Obergrenze an



# Platzierung und Tiefenstaffelung




- beliebige Platzierung mit Verdeckung
  - ⇨ `position:`
    - `static` (normaler Elementfluss; Normaleinstellung)
    - `relative` (zu ursprünglicher Position im Elementfluss)
    - `absolute` (bezogen auf Elternelement)
    - `fixed` (bezogen auf Browserfenster; scrollt nicht mit)
  - ⇨ `top`, (bzw. `bottom`), `left`, (bzw. `right`), `width`, `height`
    - mit JavaScript dynamisch änderbar ⇨ "Animation"
- Tiefenstaffelung explizit mit `z-index`
  - ⇨ z.B. `z-index:3`
  - ⇨ je größer die Zahl, desto weiter vorne
  - ⇨ positionierte Elemente ohne z-index (d.h. z-index=0) entsprechend der Reihenfolge in der HTML-Datei
    - vor allen Elementen, die nicht positioniert sind
    - oben in der Datei ⇨ im Hintergrund
    - unten in der Datei ⇨ im Vordergrund

Beispiel:

```
position:absolute;
top:35px;
left:240px;
width:150px;
height:150px;
z-index:1
```

# Hintergrundgrafik

- Auswahl des Bildes durch
  - ⇨ `background-image:url(mein_bild.jpg)`
- Wiederholung ("Kacheln der Grafik")
  - ⇨ `background-repeat:`
    - `repeat` (Wiederholung horizontal und vertikal),
    - `repeat-x` (Wiederholung nur horizontal),
    - `repeat-y` (Wiederholung nur vertikal),
    - `no-repeat`
- Positionierung (sofern nicht gekachelt)
  - ⇨ `background-position:` x y`;`
    - mit x aus `left, center` oder `right` und
    - mit y aus `top, center` oder `bottom`
- Beispiel:

```
.content2 {background-image:url(mein_bild.jpg);
background-position: center center;}
```

# Media Queries

- Mit Media Queries können Eigenschaften des Ausgabegerätes abgefragt werden
  - ⇨ Spezielle CSS-Abschnitte oder Dateien je nach Art des Ausgabegerätes, Fenstergröße, Displaygröße, Farbtiefe, Orientierung
- Beispiele
  - ⇨ Smartphones

```
<link rel="stylesheet" href="smartphone.css"
        media="only screen and (min-device-width : 320px)
        and (max-device-width : 480px)" />
```

  - ⇨ Layout für kleine Bildschirmgrößen linearisieren

```
@media screen and (max-width: 600px) {
        header, footer, .main { float: none; width: auto;}
}
```

http://maddesigns.de/css3-responsive/template/#60

# Ergänzungen für die Barrierefreiheit

- Der Aufbau eines barrierefreien Layouts
  - ⇨ ist mit CSS möglich
  - ⇨ wäre erheblicher Aufwand, wenn man es als getrenntes zweites Layout umsetzen würde –
    und würde sich finanziell nur rechnen, wenn die Website eine entsprechende Zielgruppe ansprechen soll
  - ⇨ ist aber gar nicht so aufwändig, wenn man sich an die Standards und Empfehlungen hält
  - ⇨ Zu HTML5 gehört die "WAI-ARIA-Spezifikation"
    (*Web Accessibility Initiative - Accessibility for Rich Intranet Applications*)
    - Definiert u.a. spezielle Attribute, welche unsichtbar die Funktion eines Tags beschreiben: z.B. `role="banner"` oder `role="search"`
  - ⇨ kann auf diversen Seiten und mit Tools geprüft werden
    - z.B. http://wave.webaim.org, WebAccessibilityToolbar, LynxView, TAW3
  - ⇨ gehört zum "guten Stil" eines professionellen Webauftritts

# W3C Web Content Accessibility Guidelines 2.0 – In Kurzform

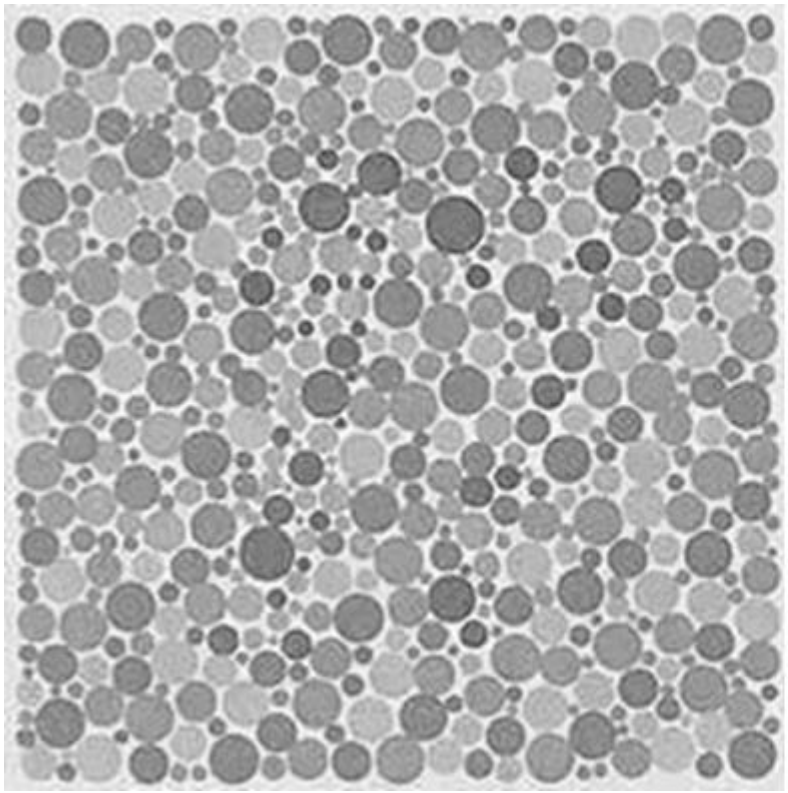
WCAG 2.0 Theme Song bei YouTube

1. Wahrnehmbar (Perceivable)
   - Biete Alternativ-Texte für Inhalte, die nicht textuell sind
   - Biete Untertitel und Alternativtexte für Audio-Inhalte und Videos
   - Mache den Inhalt anpassbar; und verfügbar für Hilfsgeräte
   - Verwende genügend Kontrast damit Dinge leicht zu sehen und zu hören sind
2. Bedienbar (Operable)
   - Mache alle Funktionen über die Tastatur zugreifbar
   - Lass Anwendern genug Zeit den Inhalt zu lesen und zu benutzen
   - Verwende keine Inhalte, die Krämpfe verursachen
   - Hilf Anwendern bei der Navigation und beim Finden von Inhalten
3. Verständlich (Understandable)
   - Mache Text lesbar und verständlich
   - Lasse Inhalte so erscheinen und sich so verhalten wie man es erwartet
   - Hilf Anwendern Fehler zu vermeiden und zu korrigieren
4. Robust (Robust)
   - Maximiere die Kompatibilität mit aktuellen und zukünftigen Technologien

Quelle: http://www.w3.org/TR/WCAG20, Übersetzung: Ralf Hahn

# Farbensehen

www.ichbinfarbenblind.de



- Sehen Sie ein 17? Eine 47? Gar keine Zahl?
  - ⇨ Ca. 10% der Menschen haben eine Farbenfehlsichtigkeit

# Farbensehen

Quelle: Assessment of inherited colour vision defects in clinical practice, *Clin Exp Optom* 2007; 90: 3: 157–175

- Können Sie erkennen, welches Fleisch roh ist?
  - ⇨ 22% der Menschen mit Farbfehlsichtigkeit können keinen Unterschied sehen

# Konkrete Tipps zur Umsetzung der Barrierefreiheit

- Layout ausschließlich über CSS definieren
- Verwendung relativer Maßeinheiten damit die Browsereinstellungen berücksichtigt werden
- keine zappelnden Animationen oder Popups mit schwer treffbaren Elementen (z.B. "Schließen-Kreuz" in vielen Foren)
- Zusätzliche Navigierbarkeit über "versteckte" Menüs
  - erlaubt direktes Anspringen und Selektieren von Hauptinhalt, Navigationsleiste, Header etc.
  - für den "normalen" Leser nicht sichtbar, aber vom Screenreader vorlesbar
  - Über separate CSS Abschnitte für den Screenreader (media "speech")

# Technische Umsetzung für ein zusätzliches Navigationsmenu

```
<nav class="barrierefrei"><ul>
 <li><a href="#nav" title="Link zur Navigationsleiste">
        Zur Navigationsleiste</a></li> ...
</ul></nav>
```

```
.barrierefrei {display: none}; /*normalerweise versteckt */
@media braille, speech { /* für speech, braille */
   .barrierefrei { display: block; } /* sichtbar */
}
```

@media erst ab IE9!
Alte verbreitete Screenreader können das nicht!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.2.6 CSS - Kaskadierung

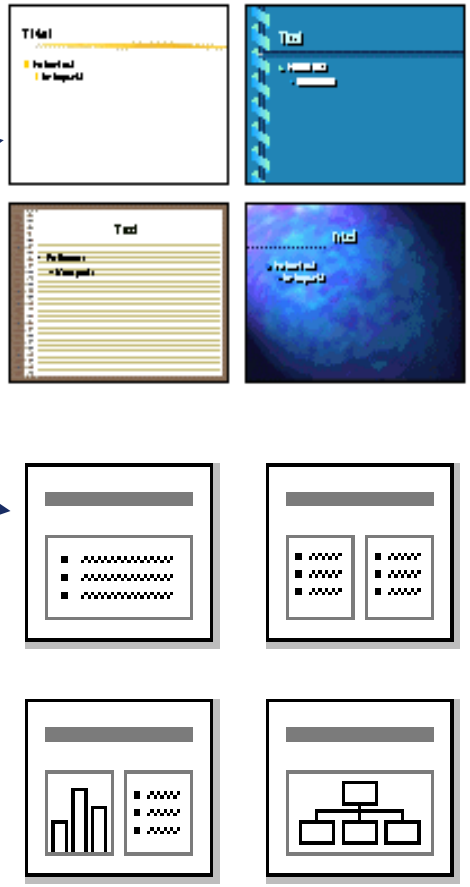
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Sinn einer Format-Hierarchie



- Corporate Identity ⇨ Corporate Design
  - ⇨ Firmenlogo, Designrahmen

1. einheitl. Erscheinungsbild des Dokuments
   - ⇨ vgl. PowerPoint: Design übernehmen
   - ⇨ Farben, Schriften, Hintergrund, Ausrichtung
2. eine Auswahl von Layout-Typen für Seiten
   - ⇨ vgl. PowerPoint: Folienlayout
   - ⇨ 0/1/2 Textblöcke, mit/ohne Bild, hor./vert. geteilt
3. Besonderheiten der einzelnen Seite
   - ⇨ Abweichung vom Layout-Typ
4. individuelles Format einzelner Objekte



# Realisierung einer Format-Hierarchie

vgl. Klassen-Hierarchie in OO

- CorporateDesign.css — kein Verweis
- DokumentDesign.css

```
@import "CorporateDesign.css";
```

extern

- SeitenLayouts.css

```
@import "DokumentDesign.css";
```

extern

- Seite3.html

```
<link href="SeitenLayouts.css" ...>
```

extern

```
<style type="text/css"> <!--
  --> </style>
```

embedded für diese Seite

```
<p style="color:red"> Block </p>
```

inline für einzelne Objekte

Mehrfache Redefinition desselben Attributs für dasselbe Objekt ist möglich !

# Auflösung von Konflikten

bei mehrfacher widersprüchlicher Definition

- Vereinigungsmenge aller Definitionen für ein Attribut eines Objekts bilden
  - ⇨ falls leer: vom umschließenden Objekt (parent) erben
  - ⇨ falls leer: Standardwert nehmen
- Sortieren nach
  - ⇨ Gewichtung (! important) — erstes Kriterium
  - ⇨ Herkunft (Autor vor Leser)
  - ⇨ Spezialisierungsgrad
    - Individuell (id)
    - vor kontextabh. Klasse (p.Hinweis)
    - vor allgem. Klasse (.Warnung)
    - vor redefiniertem Standard Format (p)
  - ⇨ Reihenfolge der Definition — letztes Kriterium
    - inline vor embedded vor extern

font-size:1em !important;

# Zusammenfassung

- CSS-Grundlagen
  - ⇨ Grundidee, Grundgerüst, HTML-Einbindung
  - ⇨ Schreibregeln und Syntax
  - ⇨ Formate modifizieren und definieren
- CSS-Attribute
  - ⇨ Farben, Hintergrund, Schriften, Ausrichtung, Ränder, Platzierung,...
- CSS-Maßeinheiten
  - ⇨ relative, absolute Maße, Zoom vs. Schriftgröße
- CSS-Layout
  - ⇨ Anordnung von Blöcken mit float, Überdeckung, Hintergrundbilder
- CSS-Kaskadierung

Jetzt wissen Sie alles um eine HTML-Seite mit einem ordentlichen Design zu entwickeln!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.3 Clientseitige Programmierung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Interaktion mit Webseiten auf dem Client

- Seitenumschaltung 
  - ⇨ Hotwords, (transparente) Schaltflächen, sensitive Grafik — HTML
- Eingabeformulare
  - ⇨ Listbox, Checkbox, Radiobutton  — HTML
  - ⇨ Konsistenzprüfung der Eingaben (auf dem Client !)  — ECMAScript
- objektbezogene Ereignisbehandlung
  - ⇨ Veränderung der HTML-Seite (auf dem Client !)  — ECMAScript, DOM
- allgemeine Programmierung — Ajax, Java Applet
  - ⇨ aufwändige Visualisierung (Konfigurator beim Autokauf), spezielle Widgets (TreeControl) — Flash

wird in der Vorlesung nicht behandelt

# Zur Abgrenzung: Server-seitig

Große Datenmengen bzw. persistente Daten werden auf dem Server gehalten!

- Content Management System
- Durchsuchen großer Datenmengen
  - ⇨ Datenbankabfrage (z.B. Fahrplanauskunft)
  - ⇨ Volltextsuchmaschine
- Speicherung von Daten
  - ⇨ Gästebuch, Schwarzes Brett, Bestellungen
- Realisierungsmöglichkeiten
  - ⇨ CGI, proprietäre Server-APIs (NSAPI, ISAPI)
  - ⇨ PHP, ASP, JSP
  - ⇨ Java Servlet
  - ⇨ Java Applet mit RMI (remote method invocation)
  - ⇨ ...

# Beispiel (ECMAScript eingebettet in HTML)

```
<!DOCTYPE html>
<html lang="de">
<head> <meta charset="UTF-8"/>
    <title>Sinn des Lebens</title>
    <script>
        var Hinweis = "Hurra! Quadratzahlen!";
        alert(Hinweis);
        function schreibeQuadrate() {
            "use strict";
            var i, x, SinnDesLebens, Satzteil;
            SinnDesLebens = 42;
            Satzteil = "Das Quadrat von ";
            for (i = SinnDesLebens; i > 0; i = i - 1) {
                x = i * i;
                document.getElementById('body').innerHTML +=
                  "<p>" + Satzteil + i + " ist " + x + "</p>";
            }
        }
    </script>
</head>
<body id="body" onload="schreibeQuadrate();">
</body>
</html>
```

aktiviert "modernes" ECMAScript

fügt HTML in das aktuelle Dokument ein

Ausführung sobald die Seite geladen ist

# Historie und Literatur

hatte mit Java nur C gemeinsam

- Ursprung: JavaScript (Netscape) in Navigator 2.0
  - ⇨ von Netscape an Microsoft lizenziert; MS hinkte hinterher
- JScript (Microsoft)
  - ⇨ lizenzunabhängige Sprachvariante mit MS-eigenen Erweiterungen (MSIE versteht JavaScript und JScript)
- ECMAScript (ECMA-262, herstellerunabhängig)
  - ⇨ European Computer Manufacturer's Association, Genf
    - aktuell: 7th Edition "ECMAScript 2016", Juni 2016
  - ⇨ autorisiert von W3C, übernommen als ISO / IEC 16262
  - ⇨ http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf
- Literatur
  - ⇨ https://developer.mozilla.org/en/JavaScript/Guide ☺
  - ⇨ http://javascript.crockford.com/
  - ⇨ David Flanagan: "JavaScript: The Definitive Guide", O'Reilly

darauf konzentrieren wir uns

# Historie: Entwicklung von JavaScript

- ⇨ Ursprung: Programmierung von Eingabeformularen
  - Ereignisbehandlung war auf Formulare beschränkt
  - komplexere Aufgaben erforderten Java,
    aber selbst damit kein Zugriff auf HTML-Dokument
- ⇨ seit ca. 1999: Layout-Elemente als programmierbare Objekte
  - alle Eigenschaften per Skript änderbar;
  - Ereignisbehandlung mit zugeordneten Skripten
  - Seiten können vom Surfer modifiziert werden (z.B. Warenkorb)
- ⇨ seit ca. 2004: Mit Ajax und JavaScript können schnelle und interaktive Webseiten entwickelt werden
  - Moderne Webseiten sind ohne JavaScript praktisch nicht mehr nutzbar
  - JavaScript gibt es als Programmiersprache auch für serverseitige Entwicklung
  - Riesige Webauftritte basieren auf JavaScript (& Frameworks)
  - Diverse Frameworks ersetzen die fehlenden OO-Features in JavaScript
- ⇨ seit 2015: JavaScript beinhaltet endlich Klassen uvm.
  - wird nicht vom iE unterstützt, aber von Edge
  - Diverse fehleranfällige Konstrukte sind verboten

# Historie: Objektbasierend statt objektorientiert!?

- ECMA Script kennt historisch keine Klassen, verwirrt den Begriff "Objekt"
  - ⇨ Aus einem alten Standard: "...Object-based, not object-oriented. Complex object features are left out..."
  - ⇨ sollte einfacher sein für Novizen !?
  - ⇨ für OO-Programmierer sehr gewöhnungsbedürftig
  - ⇨ alles und jedes ist ein Objekt, selbst eine Funktion
    - Zitat aus einer früheren Version des Standards: "All constructors are objects, but not all objects are constructors."
    - erklärt sich durch implementierungsnahe Sicht: Objekt gleichbedeutend mit Speicherbereich
  - ⇨ einige vordefinierte "Objekte" sind eigentlich Klassen
    - Boolean, String, Date, Array, ...
    - Seit ECMAScript 6 gibt es aber auch Klassen und vieles mehr!

Die Verwirrung des Begriffs "Object" findet sich in den Standards, der Literatur und auf vielen Webseiten!

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.3.1 ECMAScript: Definition

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Vergleich mit C

man kann einfach mal drauflos schreiben...

- Syntax sehr ähnlich
  - ⇨ Zuweisung, `if`, `switch`, `for`, `while`, `do`, `try catch`, `//`, `/*...*/`
  - ⇨ Konstanten, Operatoren
- Stringverkettung mit `+`
- Variablen sind dynamisch typisiert (keine Typdeklaration)
  - ⇨ Zahlen sind immer Gleitkommazahlen
  - ⇨ Schlüsselworte `var` bzw. `function` statt Typ in der Deklaration
- Objekterzeugung mit `new`
  - ⇨ wie in Java; kein delete
- nicht zeilenorientiert
  - ⇨ `;` wird am Zeilenende automatisch eingefügt, falls es fehlt (kein Zeilenende hinter `return` oder vor `++` und `--` lassen !)

# Einfache Literale (Notation für Werte)

- Notation wie in C
  - ⇨ Integer-Literal
  - ⇨ Floating-Point Literal
  - ⇨ String-Literal
  - ⇨ Boolean-Literal

```
var antwort = 42;
var pi      = 3.14;
var Gruss   = "Hello World";
var isPrime = true;
```

- Schlüsselworte `null, true, false`

- Besonderheit für Strings
  - ⇨ wahlweise mit `"..."` oder `'...'`
    ermöglicht String im String (z.B. String mit HTML-Attributwert)

# Array

das kann C nicht...

- dynamisch erzeugtes und erweiterbares Objekt
  - ⇨ ganzzahliger Index im Bereich 0..length-1
  - ⇨ Elementtyp beliebig und nicht notwendigerweise einheitlich
- Erzeugung
  - ⇨ ohne Längenangabe für dynamische Erweiterung
    ```
    var Vektor1 = Array ();
    ```
  - ⇨ mit Längenangabe (eine Zahl)
    ```
    var Vektor2 = Array (27);
    ```
  - ⇨ mit Initialisierung (mehr als 1 Wert oder Objekt)
    ```
    var Vektor3 = Array ("abc", 55, "xyz");
    ```
- Zugriff
  ```
  var Element = Vektor2[4];
  Vektor1[0] = "text";
  var AnzahlElemente = Vektor2.length;
  ```

# "Assoziatives Array"

das kann C nicht...

- dynamisch erzeugtes und erweiterbares Objekt
  - ⇨ String als Index
  - ⇨ Elementtyp beliebig und nicht notwendigerweise einheitlich
  - ⇨ vgl. Datenstruktur / struct / Hashtabelle / map / dictionary
- Erzeugung, Erweiterung und Zugriff
  - ⇨
```
var Vektor = [];
Vektor["posLeft"] = 45;
var Element = Vektor["posLeft"];
```
- Verarbeitung
  - ⇨ häufig mit Hilfe von `for` (… `in` …)
```
for (var Element in Vektor) {...Vektor[Element]...}
```
  - ⇨ Vorsicht! `for…in` zählt alle Attribute eines Objekts (hier: das "Array") auf
    - dazu gehören auch geerbte Methoden (ggf. ausfiltern mit `hasOwnProperty`)
  - ⇨ echte Array-Funktionen wie `length` gibt es hier nicht
  - ⇨ der Zugriff über den Index `Vektor[i]` ist für ein assoziatives Array undefiniert

# Literale für Arrays

- konstruieren Objekte
- dürfen auch Ausdrücke als Elemente enthalten, d.h. die Elemente müssen nicht ihrerseits Literale sein
- werden im Rahmen der JSON genutzt (siehe Abschnitt "Ajax")

⇨ Literal für numerisch indiziertes Array

```
var Tiere = ["Hund", "Hamster", "Affe"];
```

ist Abkürzung für

```
var Tiere = new Array ("Hund", "Hamster", "Affe");
```

⇨ Literal für assoziatives Array (erzeugt eigentlich ein Objekt)

```
var Preise =
          { Margherita: 4.0, Salami: 4.5, Hawaii: 5.5 }
```

# Funktionen

- Deklaration mit Schlüsselwort `function`
- Rückgabe von Werten aus Funktionen durch `return`
  - ⇨ ein Rückgabeparameter wird nicht deklariert
  - ⇨ Klammern nicht vergessen !
- Parameterübergabe wie in Java
  - ⇨ formale Parameter sind lokale Variable
  - ⇨ aktuelle Parameter werden bei Aufruf in formale Parameter kopiert
  - ⇨ Objekte werden per Referenz übergeben, nicht geklont
  - ⇨ d.h. call by value für einfache Datentypen, call by reference für Objekte
- Beispiel

```
function Doppel (InParam) {
  var  OutParam = 2 * InParam;
  return OutParam;
}
```

es gibt noch weitere Schreibweisen; das vertiefen wir hier nicht

# Vordefinierte Funktionen

- Vordefinierte Funktionen können einfach aufgerufen werden

| | |
|---|---|
| ⇨ `eval(Zkette)` | interpretiert übergebenes Argument als Code; gibt Ergebnis zurück (z.B. Objekt) |
| ⇨ `isFinite(Wert)` | auf numerischen Wertebereich prüfen |
| ⇨ `isNaN(Wert)` | auf nicht-numerischen Wert prüfen |
| ⇨ `parseFloat(Zkette)` | in Kommazahl umwandeln |
| ⇨ `parseInt(Zkette)` | String(anfang) in Ganzzahl umwandeln |
| ⇨ `Number(Wert)` | Datum in Zahl umwandeln |
| ⇨ `String(Wert)` | In Zeichenkette umwandeln |
| ⇨ `encodeURI(Zkette)`<br>`decodeURI(Zkette)` | CGI-Parameter für URL (de)kodieren (vgl Abschnitt CGI) |
| ⇨ `Zahl.toFixed(n)` | Erzwingt n Nachkommastellen |
| ⇨ ... | Übersicht und Details siehe SELFHTML |

# Klassen

erst seit JavaScript 2015

- Deklaration mit Schlüsselwort `class`
  - ⇨ Die Klasse muss vor der Verwendung deklariert werden
- Der Konstruktor heißt immer `constructor`,
  Einen Destruktor gibt es nicht
- Instanzen werden mit `new` erzeugt
- Vererbung mit: `class super extends base`
- Statische Funktionen und Attribute mit `static`
- Aber *private* und *public* gibt es nicht!
  - ⇨ Stattdessen gibt es diverse Konstruktionen, um Kapselung zu erreichen

Es gibt noch weitere Konstrukte zur Strukturierung von Code (z.B. Module).
Das vertiefen wir hier aber nicht!

# Beispiel: Klasse Bruch in EcmaScript6

Anwendung der Klasse

```
function main ()
{
   var x = new Bruch (3, 5);
   var y = new Bruch (4, 7);

   var m = x.mal(y);
   document.write (m.toString());
}
```

Klassendefinition

```
class Bruch {
   constructor(zaehler, nenner) {
      this.z = zaehler;
      this.n = nenner;
   }
   mal(faktor) {
      var result = new Bruch
            (this.z*faktor.z, this.n*faktor.n);
      return result;
   }
   toString() {
        return '(' + this.z + ' / ' + this.n + ')';
   }
}
```

Attribute

Funktion

Das Beispiel läuft nicht im Internet Explorer!

# Ausnahmebehandlung

- wie in Java / C++, Ausnahmeobjekte jedoch dynamisch typisiert

```
try {
   ...
   if (FehlerAufgetreten)
      throw "Text oder Objekt";
   ...
}
catch (Ausnahme) {
   alert (Ausnahme);    // sofern es Text ist
}
```

- zusätzlicher `finally`-Block möglich wie in Java
  - ⇨ wird in jedem Fall ausgeführt
- sicherheitshalber einbauen, auch ohne eigenes throw
  - ⇨ manche Browser werfen bei manchen JavaScript-Fehlern Ausnahmen aus...

# Ordentliches ECMAScript!? Use Strict

- ECMAScript erlaubt traditionell viele Konstrukte, die fehleranfällig sind oder Seiteneffekte erzeugen
  - ⇨ Seit ECMAScript 6 sind viele dieser Konstrukte verboten
  - ⇨ Aber aus Gründen der Abwärtskompatibilität gibt es sie noch
- Der Befehl "use strict"; aktiviert eine strenge Interpretation
  - ⇨ der Browser bricht dann bei einem Fehler die Ausführung ab (!!)
  - ⇨ Variablen müssen deklariert werden
  - ⇨ die Verwendung des with-Befehls ist verboten
  - ⇨ diverse zweifelhafte Konstrukte sind verboten
  - ⇨ je nach Platzierung gültig für eine Funktion oder das ganze Skript
  - ⇨ alte Browser, die "use strict"; nicht kennen, ignorieren den Befehl
  - ⇨ Vorsicht bei Altlasten und fremden Bibliotheken! Das nachträgliche Aktivieren von "use strict"; kann zu unerwarteten Ergebnissen führen
    - Code bricht plötzlich ab
    - Variablen liefern unerwartete Werte

# Ordentliches ECMAScript (II)

So verwenden wir es in EWA !

- Empfohlener Umgang mit "use strict";
  - ⇨ bei Neuentwicklungen unbedingt einsetzen
  - ⇨ In Funktionen verwenden (und nicht für das ganze Skript)

```
function schreibeQuadrate() {
  "use strict";
    ...
}
```

- Zusätzlich kann auch ein Analysetool verwendet werden
  - ⇨ Bei seltsamen Effekten oft hilfreich
  - ⇨ JSLint: http://www.jslint.com
  - ⇨ ESLint mit ECMAScript 6: http://eslint.org/demo/

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.3.2 ECMAScript: Skript und HTML

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK

# Platzierung von Skripten

"Globaler Code" wird während des Ladens der HTML-Datei ausgeführt. Funktionen erst bei Aufruf oder als Ereignis-Handler

- "extern" in eigener JS-Datei
  - ⇨ "Bibliothek" kann von mehreren HTML-Dateien genutzt werden

```
<script src="MyJS.js"> </script>
```

- "eingebettet" im HTML-Code
  - ⇨ brauchbar nur für diese eine HTML-Datei
  - ⇨

```
<script>
   // ... ECMAScript Anweisungen ...
</script>
```

in <head>
oder <body>

DOM ist aber eventuell noch nicht komplett für Ereignis-Handler !

- falls Skriptausführung im Browser abgeschaltet ist

```
<noscript>
   <p>Bitte aktivieren Sie JavaScript !</p>
</noscript>
```

# Ereignisse und Handler

- vordefinierte Ereignisse

  nicht ECMA Script, sondern HTML

  - ⇨ Maus: `onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout`
  - ⇨ Tastatur: `onkeydown, onkeyup, onkeypress`
  - ⇨ Formular: `onchange, onfocus, onblur, onsubmit, onreset`
  - ⇨ Datei: `onload, onunload, onabort`

  optimal für Initialisierung

- Zuordnung "inline" im HTML-Tag
  - ⇨ normalerweise: Funktionsaufruf als Event-Handler (beliebige ECMAScript-Anweisungen sind aber möglich)

```
<p onclick="Funktionsaufruf(27)"> ein Text </p>
```

oder

```
<input type="radio" onclick="document.forms['f1'].submit();"
... />
```

# Reihenfolge von Maus-Ereignissen

Achtung !
onmouse... funktioniert nicht (richtig)
auf Touchscreens

- beim Schieben:
  - ⇨ `onmouseover` Cursor geht in den Objektbereich
  - ⇨ `onmousemove` wiederholt, solange in Objektbereich

- beim Klicken:
  - ⇨ `onmousedown` — Analog „Selektieren“ in Windows-Menüs
  - ⇨ `onmouseup` — Analog „Ausführen“ in Windows-Menüs
  - ⇨ `onclick` down und up an derselben Stelle
  - ⇨ `ondblclick`

- beim Verlassen
  - ⇨ `onmouseout` Cursor verlässt den Objektbereich

t

# Überprüfung von Formulareingaben

- Script-Funktion mit Boole'schem Ergebnis

```
function FormulardatenOK()
{
   if      (!Feld1_OK) return false;
   else if (!Feld2_OK) return false;
   else                return true;
}
```

- Zuordnung zum Ereignis `onsubmit`
  - ⇨ das Abschicken der Formulardaten wird unterdrückt, wenn die Funktion `false` liefert

Sonderfall; nur bei onsubmit

```
<form onsubmit="return FormulardatenOK();">
  <input type="submit" value="Abschicken">
</form>
```

# Vorsicht mit globalem Code !

- globale Anweisungen werden ausgeführt, sobald sie eingelesen sind
  - ⇨ d.h. während des Aufbaus der HTML-Seite
    - die Anzeige der Seite wird ggfs. verzögert
  - ⇨ Achtung: im `<head>` existiert der DOM-Baum noch nicht !
  - ⇨ sicherer: Initialisierungen in `<body onload="Init()">`

- globale Anweisungen beschränken auf Deklaration globaler Variablen und deren Initialisierung
  - ⇨ alles andere im Handler für `onload` von `<body>`
  - ⇨ insbesondere Zugriff auf DOM nicht als globaler Code !

# Initialisieren und Aufräumen

- besondere Ereignisse für `<body>`
  - ⇨ `<body onload="Initialisieren();" onunload="Aufraeumen();">`
  - ⇨ oder alternativ über DOM zuweisen
  - ⇨ `document.getElementsByTagName("body")[0].onload = Initialisieren`
  - ⇨ `document.getElementsByTagName("body")[0].onunload = Aufraeumen`
- `body.onload` ruft "Konstruktor" der Seite
  - ⇨ tritt ein, nachdem die HTML-Datei komplett geladen wurde
  - ⇨ Zugriff auf DOM jetzt möglich (ist nun komplett aufgebaut)
  - ⇨ `window.onload` nach Laden von Bildern, Skripten, CSS-Dateien
- `onunload` ruft "Destruktor" der Seite
  - ⇨ tritt ein, bevor die Seite verlassen wird

in HTML

im Skript

Funktionszeiger (ohne Klammern!)

# Zeitsteuerung

- Verzögerte Ausführung
  - `window.setTimeout (Anweisung, Verzoegerung);`
  - Anweisung: beliebige JavaScript-Anweisung (meist Funktionsaufruf), geschrieben als String
  - Verzögerung in msec

- Periodische Ausführung
  - ```
    var ID = window.setInterval
                              (Anweisung, Periodendauer);
    window.clearInterval (ID);
    ```
  - Periodendauer in msec
  - ID identifiziert Timer-Objekt (es können mehrere parallel laufen)
    ```

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.3.3 ECMAScript:
## Dokument Objekt Modell – DOM

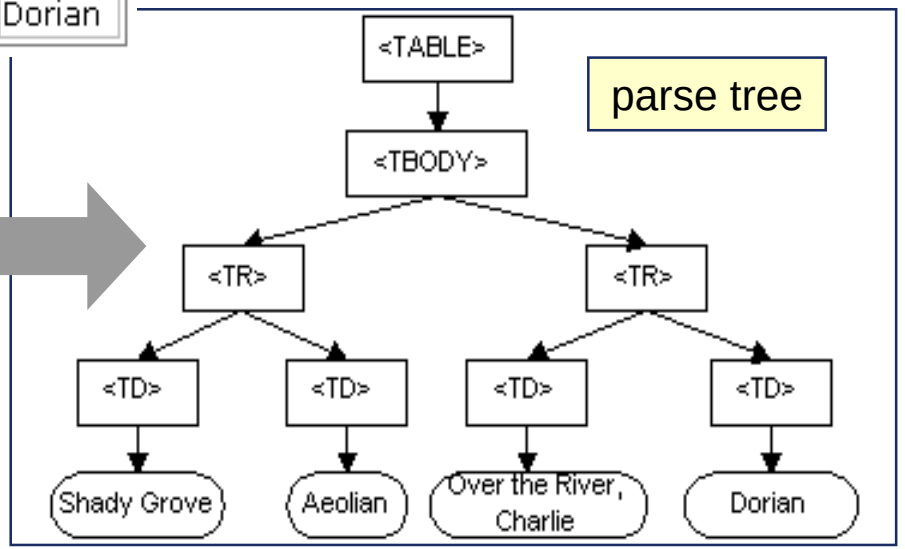
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# HTML als Baumstruktur

- ergibt sich zwanglos aus der Schachtelung von Tags

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```

| Shady Grove | Aeolian |
|---|---|
| Over the River, Charlie | Dorian |




Attribute und Texte
sind eigene Unterknoten !

(aus W3C: Document Object Model Level 2 Core Specification)

# Grundproblem / Grundidee DOM

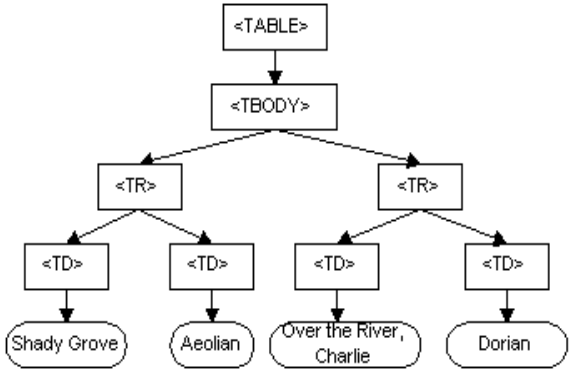
Browser

| | |
|---|---|
| Shady Grove | Aeolian |
| Over the River, Charlie | Dorian |

sichtbar

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
```

public




Browser-intern; spezif. Zugriffe

- mache den "internen Baum" durch Standard-schnittstellen allgemein zugänglich
- Der Browser muss die Abbildung der Schnittstellen auf die interne Datenstruktur implementieren

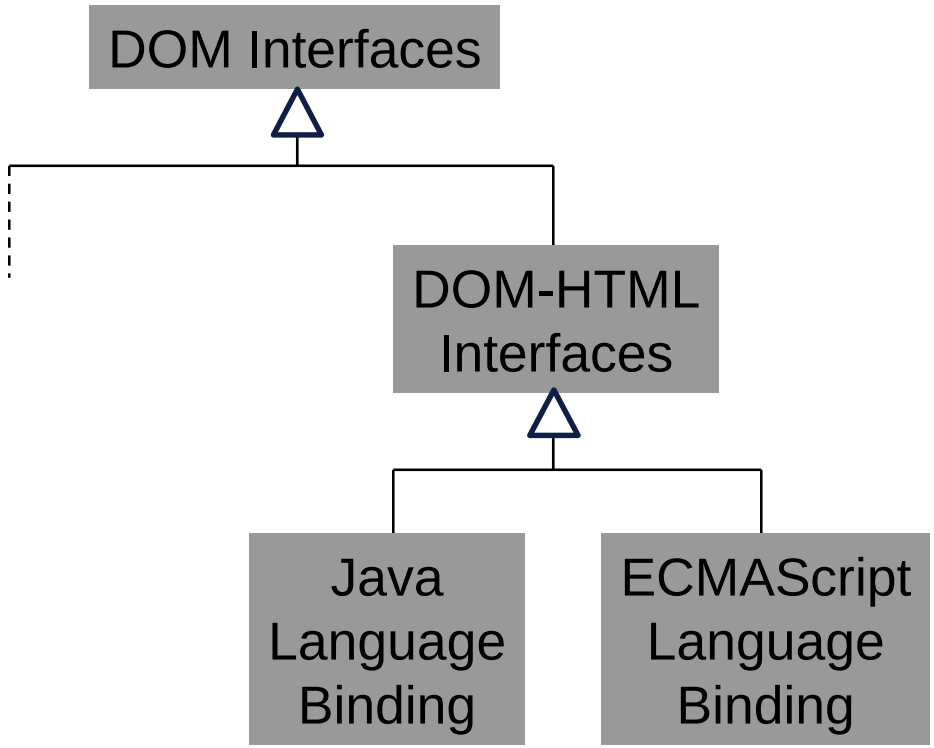
  ⇨ " DOM"

# DOM

http://www.w3.org/TR/DOM-Level-3-Core

- Status: DOM Level 3 ist seit 2004 W3C Empfehlung
  - ⇨ API für XML Dokumente, spezialisiert für HTML
    - Programmierschnittstelle, die Zugriffsmöglichkeiten auf HTML Seiten definiert
    - Konkretisierung der Interfaces für Java und JavaScript (ECMAScript)
    - ECMAScript-Anbindung ist heute in allen Browsern implementiert
    - Anwendungen: Animationen mit Dynamic HTML, Rich Internet Applications, Autorenwerkzeuge, Archivierung
- Vorteile
  - ⇨ Skriptsprachen können browserunabhängig HTML-Seiten im Browser verändern
  - ⇨ Webseiten können so komfortabel wie klassische GUI-Clients implementiert werden
    - AJAX (Asynchronous JavaScript And XML) erweitert diese Möglichkeiten noch

# Hierarchie der Standardisierungsdokumente

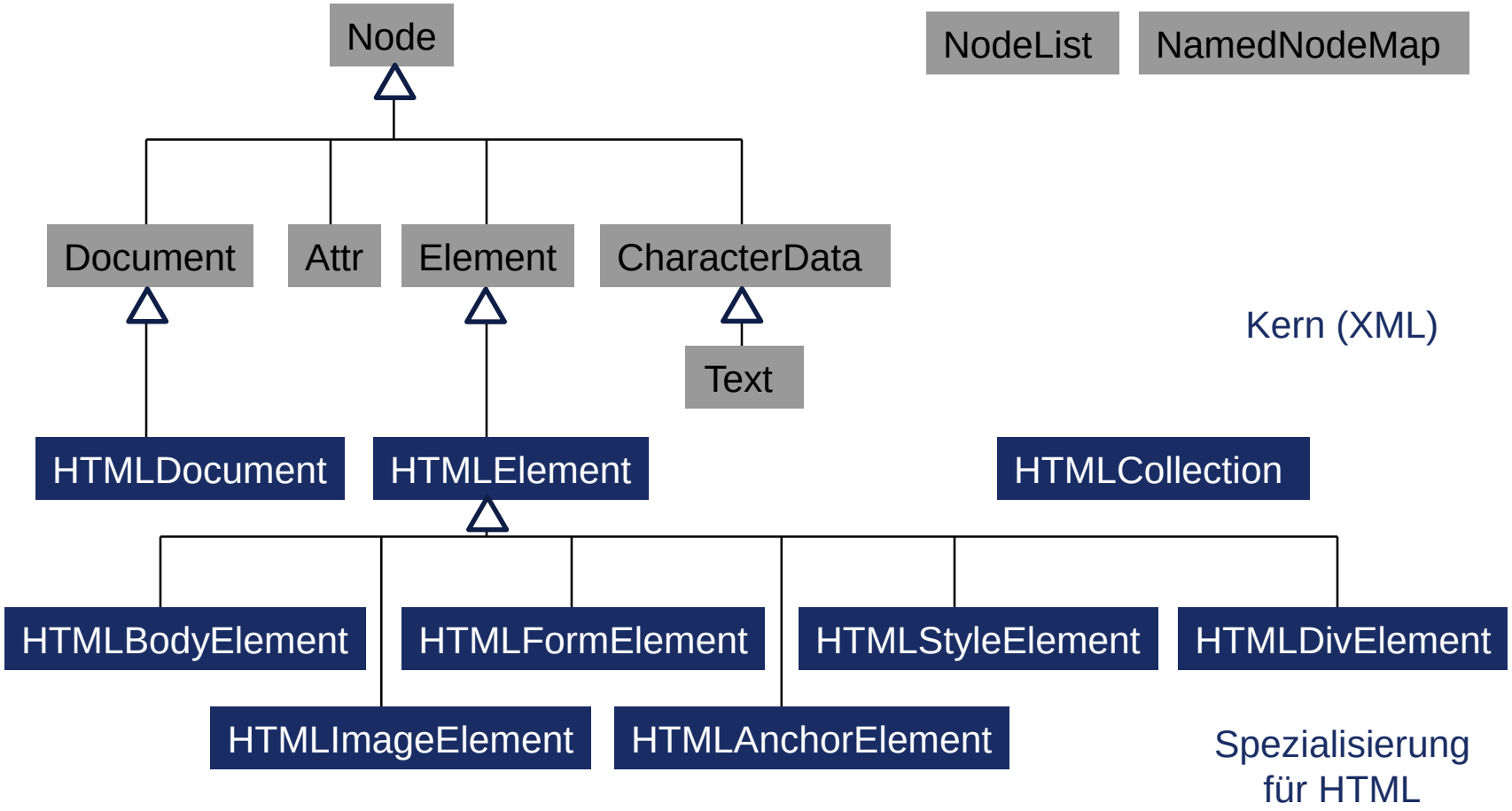



Kern (XML)

Spezialisierung für HTML

Konkretisierung für Programmiersprachen

# Hierarchie der Interfaces und Klassen



# Hochschule Darmstadt
# Fachbereich Informatik

## 2.3.4 ECMAScript: Zugriff auf DOM

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Zugriff auf Knoten über Kern-Klassen

- Ausgangspunkt ist `document` oder ein Element-Knoten
- Direkter Zugriff auf eindeutiges Element per `id`
  - ⇨ `Knoten = document.getElementById("xyz")`
  - ⇨ jedes benötigte HTML-Element mit eindeutiger `id` bezeichnen
- Zugriff auf Elemente aus Collection mit demselben Tag
  - ⇨ `Knoten = document.getElementsByTagName("h3")[2]`
- Zugriff auf Elemente aus Collection mit demselben `name`
  - ⇨ `Knoten = document.getElementsByName("abc")[0]`
  - ⇨ nicht jedes Tag darf `name` haben
  - ⇨ evtl. mehrere Elemente mit demselben `name` (vgl. Radiobuttons)

nur für document

- alle Varianten liefern einen HTML-Element-Knoten ab

# Zugriff auf Knoten über HTML-Collections

- die Klassen HTMLxxx haben entsprechende "Collections"
  - ⇨ HTMLDocument: images, applets, links, forms, anchors
  - ⇨ HTMLFormElement: elements

- Anwendungsbeispiel
  - ⇨
```
MeinFormular = document.forms["Anmeldung"];
MeinFormular.elements["Eingabe"].value = 0;
```

- aus diesen Collections kann per id oder per name ausgewählt werden
  - ⇨ `"Anmeldung"` und `"Eingabe"` können in HTML als id oder als name eingetragen sein
    - id hat Vorrang
    - name ist nicht bei allen Tags zulässig

# name-Attribut versus id-Attribut

Aber: Daten in Formularen werden nur übertragen, wenn die Felder ein name-Attribut haben!

- `name` ist nur bei manchen Tags zulässig
  - ⇨ muss nicht eindeutig sein
    - bei Radiobuttons: Gruppenbildung über denselben Namen
  - ⇨ wird für verschiedene Zwecke eingesetzt
    - `<a>` Sprungmarke (veraltet; in HTML5 id in beliebigem Tag)
    - `<input>` Parametername für die Datenübertragung
- `id` ist bei allen Tags zulässig
  - ⇨ muss dateiweit eindeutig sein
  - ⇨ ist gedacht für eindeutige Knotenadressierung im DOM
- für Knotenadressierung `id` bevorzugen
  - ⇨ `name` ist dafür nur aus Kompatibilitätsgründen noch zulässig; in DOM2 Core nicht enthalten
- `id` muss mit einem Buchstaben beginnen, dann Buchstaben, Ziffern oder - _ : .
  Nicht erlaubt sind Sonder-, Leer- oder andere Interpunktionszeichen.

# Weitere Möglichkeiten zum Zugriff auf Knoten

- speziell beim Aufruf von Handlerfunktionen:
  - ⇨ `this` verweist auf das Element, in dem der Code steht
    `<div onclick="Verbergen(this);"> ... </div>`
    - nur gültig innerhalb eines HTML-Tags
    - innerhalb einer Funktion, die zu einer Klasse gehört, verweist `this` auf das Objekt, auf das die Funktion gerade angewandt wird

findet man oft – ist aber nicht standard-konform !

- veraltete Methode für manche Objekte:
  - ⇨ wahlfreier Zugriff ohne Nennung der Collection unter Verwendung des name-Attributs
  - ⇨
```
document.MeinFormular.Eingabe.value = "alt";
document.MeinBild.src = "bild.gif";
```
  - ⇨ korrekt wäre der Zugriff über die Collection:
    `document.forms["MeinFormular"]…`

# Baumoperationen über Kern-Klassen

DOM2 Core

sehr hilfreich: Mozilla DOM Inspector

- Die Klassen Node, Document und Element bieten Methoden zum Durchwandern und Manipulieren des Baums
- Erzeugung mit

```
Document.createAttribute()      Attributknoten
Document.createElement()        HTML-Elementknoten
Document.createTextNode()       Knoten für Textinhalt
```

- Eigenschaften zum Durchwandern

```
Node.attributes, Node.childNodes, Node.parentNode
Node.firstChild, Node.lastChild
Node.nextSibling, Node.previousSibling
```

  ⇨ Achtung: Blanks zwischen Tags werden in leere Textknoten abgebildet! Dagegen hilft Node.firstElementChild, Node.lastElementChild, Node.nextElementSibling, Node.previousElementSibling

- Methoden zur Strukturänderung

```
Node.appendChild(...), Node.removeChild(...)
Element.setAttributeNode(...)
Element.removeAttribute(...)
```

# Zugriff auf Attribute

- alle Attributwerte in DOM2 HTML sind Strings
- Zugriff über Kern-Klassen (allgemein)

  DOM2 Core

  ⇨ jedes Attribut ist in einem eigenen Unterknoten gespeichert

  ⇨ Element.getAttribute und .setAttribute zum Zugriff auf bereits existierende Attributknoten (das sind alle HTML-Attribute)

```
var meinBild = document.images["BildId"];
meinBild.setAttribute("src", "bild.gif");
var bilddatei = meinBild.getAttribute("src");
```

  ⇨ (Attributknoten allokieren und in Baum einbauen)

- Zugriff über HTML-Klassen (kompakt)

  DOM2 HTML

  ⇨ alle Klassen HTMLxxxElement haben ihre jeweiligen Attribute

  ⇨

```
var meinBild = document.images["BildId"];
meinBild.src = "bild.gif";
```

Sonderfall: `class` → `className`
(class ist ein Schlüsselwort)

# Zugriff auf vom Programmierer definierte Datenattribute


- Zu jedem Tag können Datenattribute hinzugefügt werden
  - ⇨ mit dem Prefix "data-" zur Kennzeichnung
  - ⇨ mit einem Namen ohne Großbuchstaben
  - ⇨ Bsp.: data-preis, data-key, data-xxx
  - ⇨ In HTML oder auch über das DOM
  - ⇨ Der Zugriff erfolgt im DOM über `getAttribute("data-…")`
- Beispiel Pizza-Service:
  - ⇨ Definition:

```
<li id="S17" data-preis="4.99">Salami</li>
```

  - ⇨ Zugriff im DOM:

```
var Pizza=document.getElementById("S17");
var Preis=parseFloat(Pizza.dataset.preis);

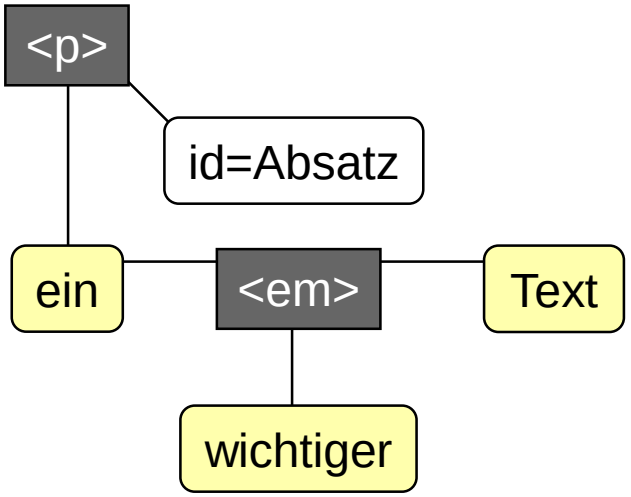
//konventionelle Methode klappt immer:
var Preis=parseFloat(Pizza.getAttribute("data-preis"));
```

Der Zugriff über dataset funktioniert nicht im IE 10!

Siehe auch: http://www.w3.org/html/wg/drafts/html/master/dom.html#custom-data-attribute

# Zugriff auf Text

```
<p id="Absatz">ein
        <em>wichtiger</em> Text </p>
```




- von einem Tag eingeklammerter Text ist in einem eigenen Unterknoten gespeichert

  ⇨ der Text steckt dort im Attribut `nodeValue`

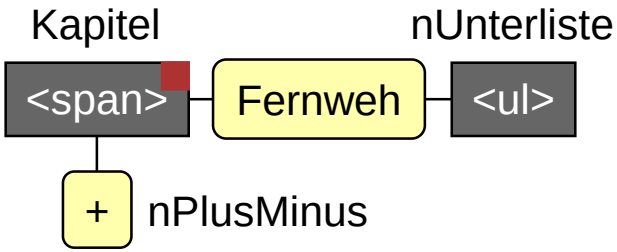
  ⇨ Änderung durch Zuweisung, aber unformatiert (ohne HTML-Tags)

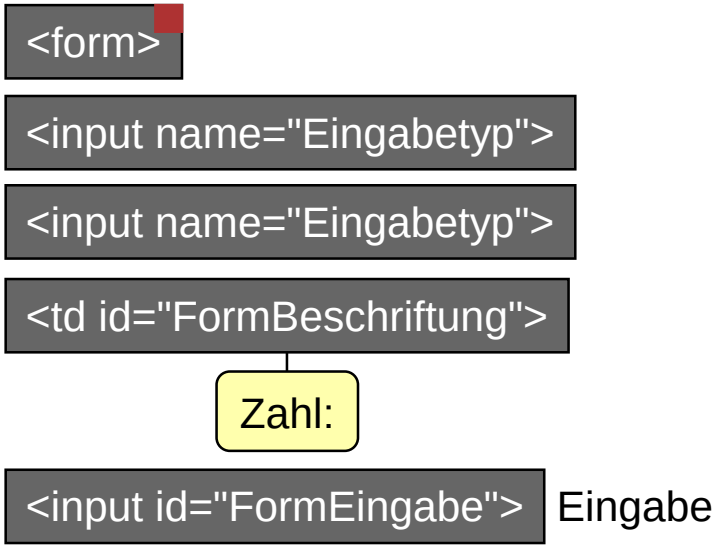
```
var Absatz = document.getElementById("Absatz");
var ein = Absatz.firstChild.nodeValue;
var em = Absatz.firstChild.nextSibling;
var wichtiger = em.firstChild.nodeValue;
var Text= Absatz.lastChild.nodeValue;
```

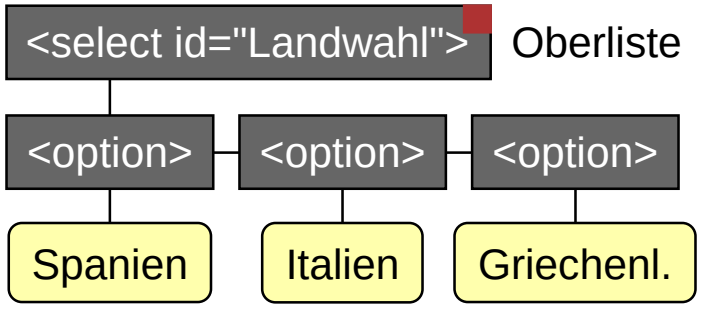
# Zugriff auf Styles

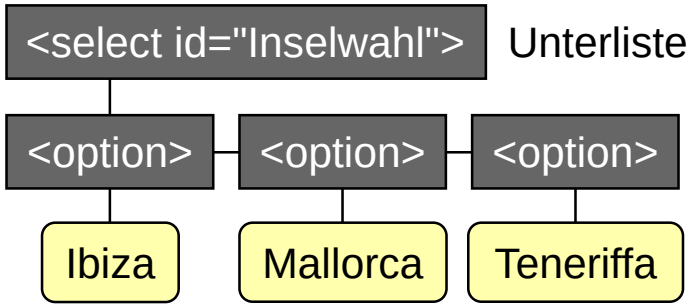
- auf inline-Styles des HTML-Elements, nicht auf CSS-Datei
  - ⇨ `<p id="Hugo" style="font-size:1.2em; font-weight:bold">`
  - ⇨ haben Vorrang vor CSS-Datei, vgl. Kaskadierungsregeln
- Werte sind Strings
  - ⇨ Vorsicht bei Arithmetik; Strings enthalten px, %, pt, em
- Sonderregel für CSS Attributnamen im Skript
  - ⇨ Bindestriche sind nicht zulässig in Bezeichnern; deshalb Bindestrich weglassen und nächsten Buchstaben groß: `fontSize, fontWeight`
- Style ist als Unterobjekt realisiert
  - ⇨ `document.getElementById("Hugo").style.fontSize = "1.2em";`

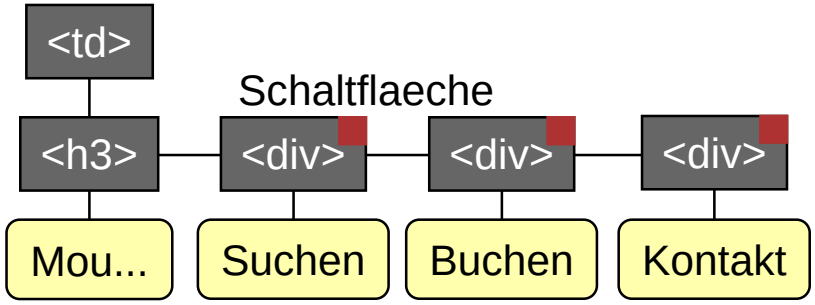
# Beispiele für ECMAScript und DOM



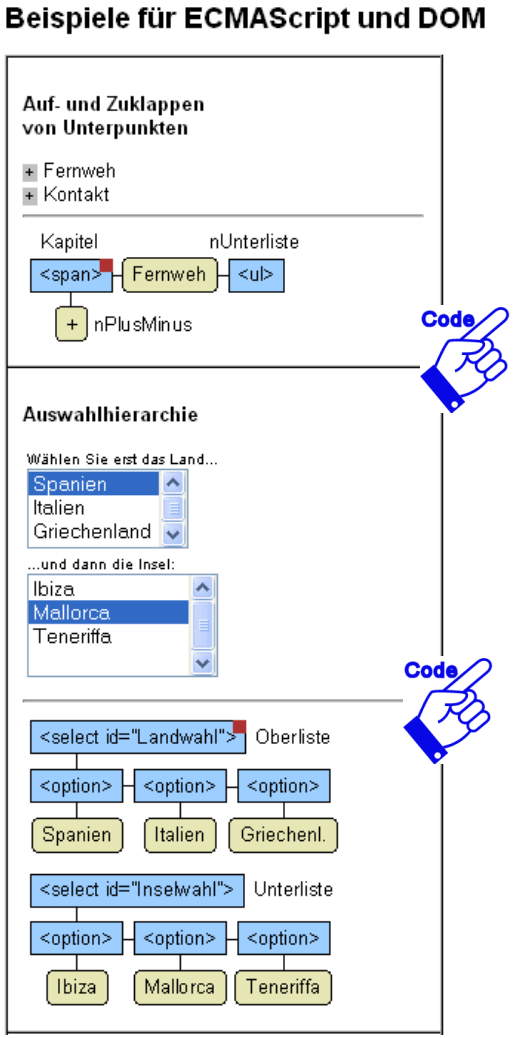



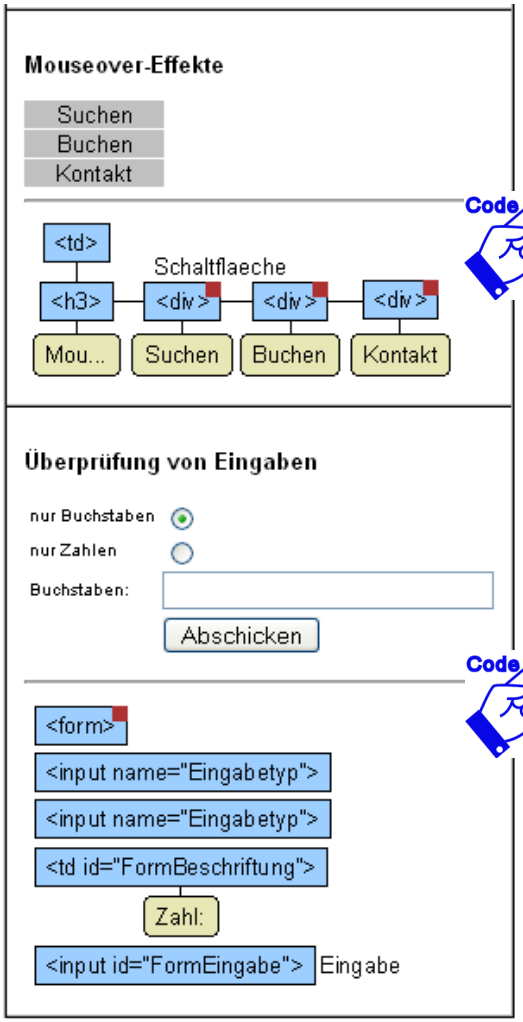
# Beispiel ECMA-Script und DOM (I)




```
function AufZuKlappen (Kapitel)
{
    try {    // Beispiel zur Ausnahmebehandlung
        var nPlusMinus = Kapitel.firstChild;
        var nUnterliste = Kapitel.nextSibling.nextSibling;
        var istZu = (nPlusMinus.nodeValue == "-");
        if (istZu) {
            nPlusMinus.nodeValue = "+";
            nUnterliste.style.display = "none";
        }
        else {
            nPlusMinus.nodeValue = "-";
            nUnterliste.style.display = "block";
        }
    }
    catch (Ausnahme) {
        zeigeAusnahme (Ausnahme);
    }
}
// Auswahlhierarchie
var Inseln_Spanien = new Array ("Ibiza", "Mallorca", "Teneriffa");
var Inseln_Italien = new Array ("Elba", "Sardinien");
var Inseln_Griechenland = new Array ("Korfu", "Kreta", "Rhodos", "Samos");
function Vorauswahl(OberlisteID, UnterlisteID)
{
    var Oberliste = document.getElementById(OberlisteID);
    var Unterliste = document.getElementById(UnterlisteID);
    var Auswahl = Oberliste.options[Oberliste.selectedIndex].text;
    var Inseln = eval ("Inseln_" + Auswahl);

    while (Unterliste.firstChild != null)
        Unterliste.removeChild (Unterliste.firstChild);
    for (i=0; i<Inseln.length; i++) {
        var neuesElement = document.createElement ("option");
        var neuerText = document.createTextNode (Inseln[i]);
        neuesElement.appendChild (neuerText);
        Unterliste.appendChild (neuesElement);
    }
}
```

# Beispiel ECMA-Script und DOM (II)




```
// Mouseover-Effekte
        function Mouseover (Schaltflaeche)
        {
            Schaltflaeche.className = "ButtonOver";
            // Sonderfall, weil class ein reserviertes Wort ist
        }
        function Mouseout (Schaltflaeche)
        {
            Schaltflaeche.className = "ButtonNormal";
        }
        function Mousedown (Schaltflaeche)
        {
            Schaltflaeche.className = "ButtonDown";
        }
        var bgStyle = null;
        var bgColor = null;
        function Mouseup (Schaltflaeche)
        {
            if (Schaltflaeche.className=="ButtonDown") {
                bgStyle = Schaltflaeche.parentNode.style;
                if (bgColor==null)  // gegen retriggern
                    bgColor = bgStyle.backgroundColor;
                bgStyle.backgroundColor = "rgb(242,216,216)";
                window.setTimeout("bgStyle.backgroundColor = bgColor;",500);
            }
            Mouseover (Schaltflaeche);
        }
// Überprüfung von Eingaben
        function BuchstabenZiffernUmschaltung()
        {
            if (document.getElementsByName("Eingabetyp")[0].checked==true)
                document.getElementById("FormBeschriftung").firstChild.nodeValue = "Buchstaben:";
            else
                document.getElementById("FormBeschriftung").firstChild.nodeValue = "Zahl:";
        }
        function isAlpha(Zeichen)
        {
            return (Zeichen>="a" && Zeichen<="z") || (Zeichen>="A" && Zeichen<="Z");
        }
```

# Beispiel: DOM-Zugriffe aus ECMA-Script (DOM Core)

- // UnterlisteID ist die ID meiner Select-Liste
```
var Unterliste =
document.getElementById("UnterlisteID");
```
- // offline!!
  // den neuen Teilbaum anlegen und initialisieren
```
var neuesElement =
    document.createElement("option");
var neuerText =
    document.createTextNode("Meine Option");
```
- // den Textknoten an die Option anhängen
```
neuesElement.appendChild(neuerText);
```
- // jetzt den neuen Teilbaum einhängen
```
Unterliste.appendChild(neuesElement);
```

Low-Level Verfahren
DOM Core

# Beispiel: DOM-Zugriffe aus ECMA-Script (DOM HTML)

- // UnterlisteID ist die ID meiner Select-Liste

```
var Unterliste =
document.getElementById("UnterlisteID");
```

- // offline!!
  // den neuen Teilbaum anlegen und initialisieren

```
var neuesElement =
    document.createElement("option");
neuesElement.text = "Meine Option";
neuesElement.value = "1";      // optional
neuesElement.selected = true; // optional
```

  ⇨ new Option(...) ist veraltet und kein Standard!

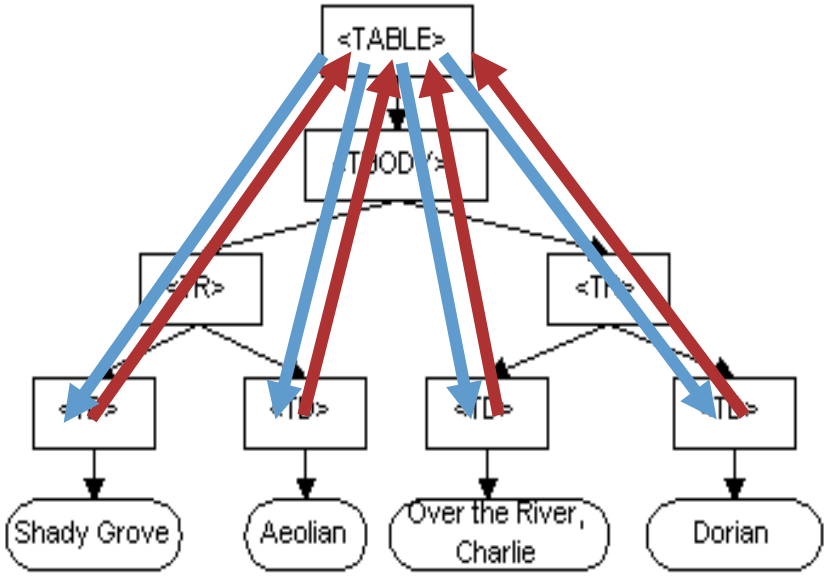
erzeugt bereits ein HTMLOptionElement

High-Level Verfahren DOM HTML

- // und die neue Option hinten anhängen

```
Unterliste.options[Unterliste.length]=neuesElement;
```

# Ereignisweiterleitung




- typisch für Ereignisorientierung
  - ⇨ vgl. ToolBook, Director, Windows
- Ereignisse werden in der DOM-Hierarchie weitergeleitet
  - ⇨ Netscape (event capturing): in der HTML-Schachtelungsstruktur von außen nach innen
  - ⇨ Microsoft (event bubbling): in der HTML-Schachtelungsstruktur von innen nach außen
  - ⇨ W3C: erst von außen nach innen, dann von innen nach außen
- Zuordnung (Registrierung) von Handlern
  - ⇨ an allen besuchten Elementen können Handler aufgerufen werden
    - default ist bubbling; capturing nur via addEventListener-Parameter
  - ⇨ meistens wird aber pro Ereignis nur ein Handler registriert

# Hochschule Darmstadt
# Fachbereich Informatik

## 2.3.5 Ajax

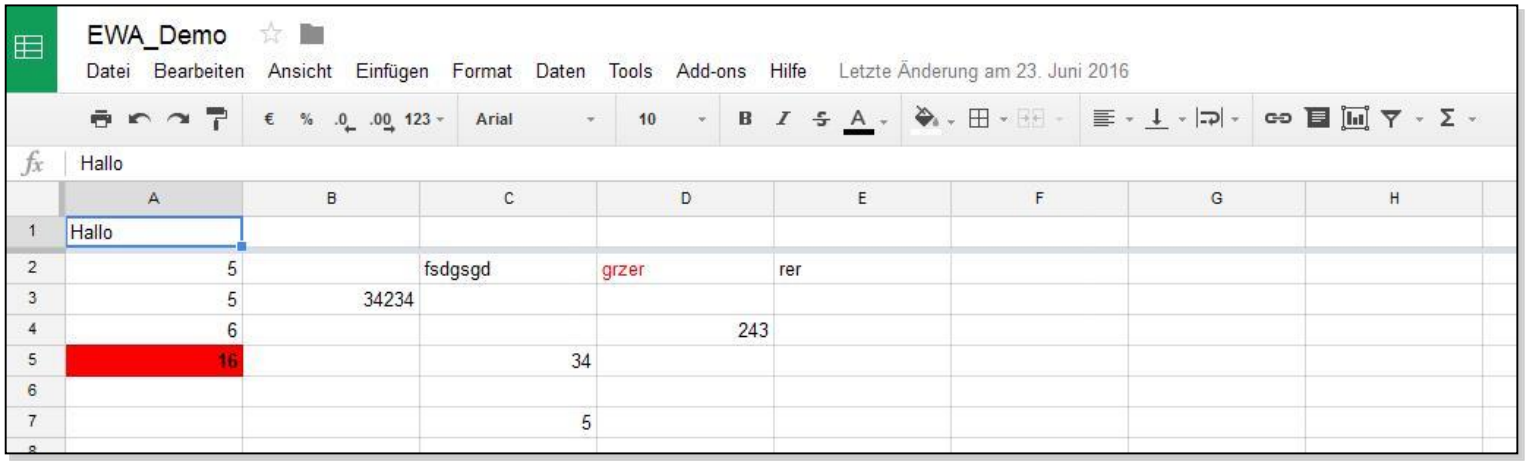
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Ajax – Asynchronous JavaScript and XML

Begriff geprägt von J.J.Garrett 2005

- Ziel: webbasierte Anwendungen sollen sich anfühlen wie klassische GUI-basierende
  - ⇨ ohne Neuladen der Seite für jede Kleinigkeit
  - ⇨ Rich Internet Applications
  - ⇨ z.B. Google Docs

vorher schon intensiv genutzt durch Google, z.B. Google Maps



# Ajax – Grundidee

- Funktionsprinzip
  - ⇨ HTML-Seite wird nicht neu geladen
  - ⇨ Daten werden bei Bedarf im Hintergrund vom Server nachgeladen
  - ⇨ JavaScript behandelt verschiedene Ereignisse und manipuliert bei Bedarf das DOM

"asynchron"

- Komponenten sind bereits weitgehend bekannt
  - ⇨ HTML, CSS, JavaScript, DOM, (XML)
- Neuerungen
  - ⇨ XMLHttpRequest zum Laden im Hintergrund
  - ⇨ JavaScript wird essentiell – ohne JavaScript funktioniert AJAX nicht
  - ⇨ es gibt diverse Frameworks zur Erleichterung des Umgangs mit JavaScript und DOM (z.B. jQuery, prototype.js, Dojo)

seit Microsoft IE 5

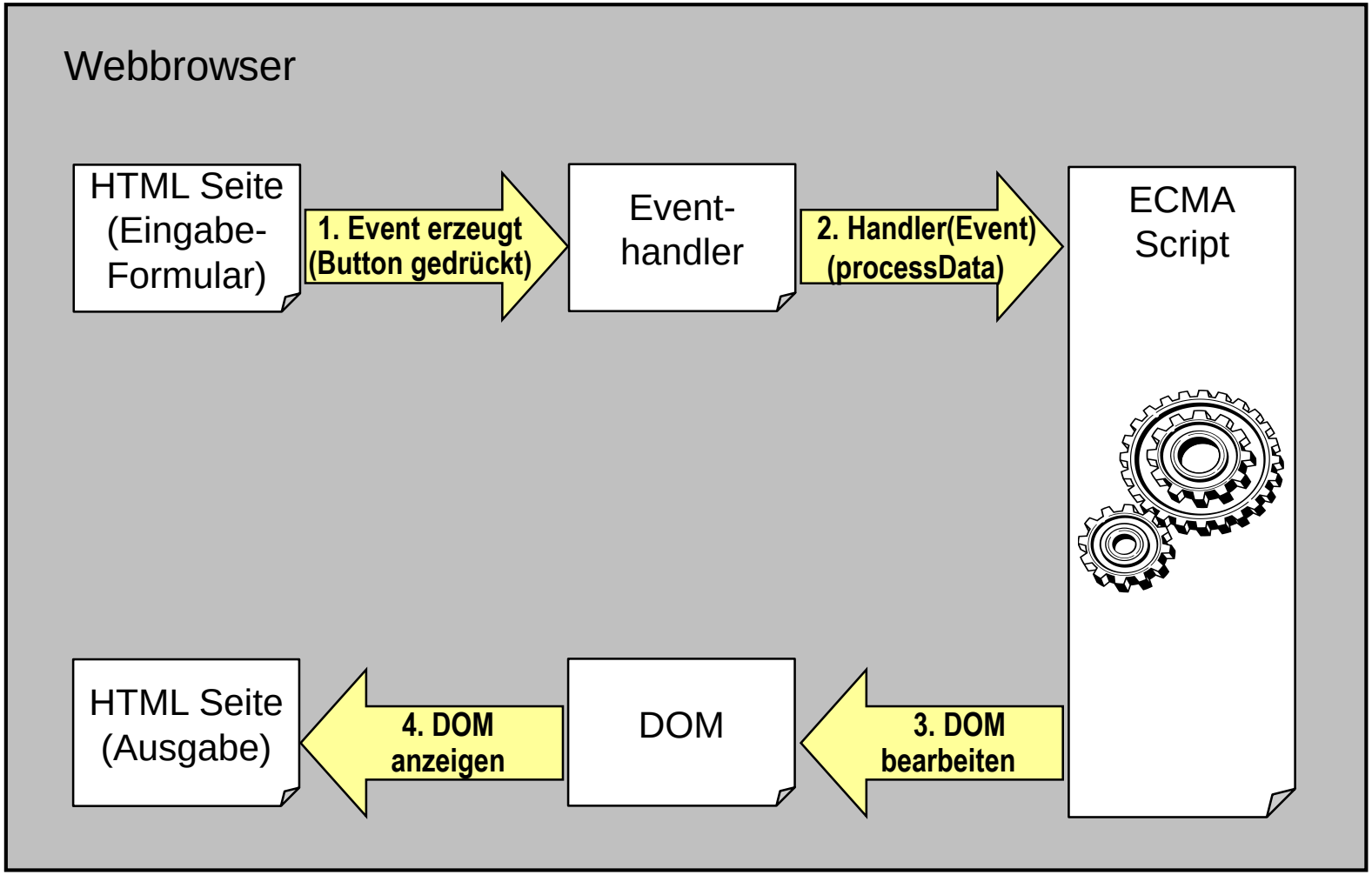
# Hinführendes Beispiel – Teil1


- Wir entwickeln eine HTML-Seite, welche auf Knopfdruck einen String auf einer Webseite einfügt
  - ⇨ einfaches HTML mit einem Button, etwas Javascript und DOM

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8" />
  <title>Test</title>
  <script type="text/javascript">
function processData () {
  var myText = document.getElementById("meinText");
  myText.firstChild.nodeValue = '26.01.2009 11:51:02';
}
</script>
</head><body><h1>Hier steht der Text</h1>
  <p id="meinText">mein Text</p>
<form action="index.html"> <input type="button" value="Los!"
       onclick="processData()"/>
</form> </body> </html>
```

Der Text wird bei Knopfdruck eingefügt

# Schematischer Ablauf zu Beispiel 1



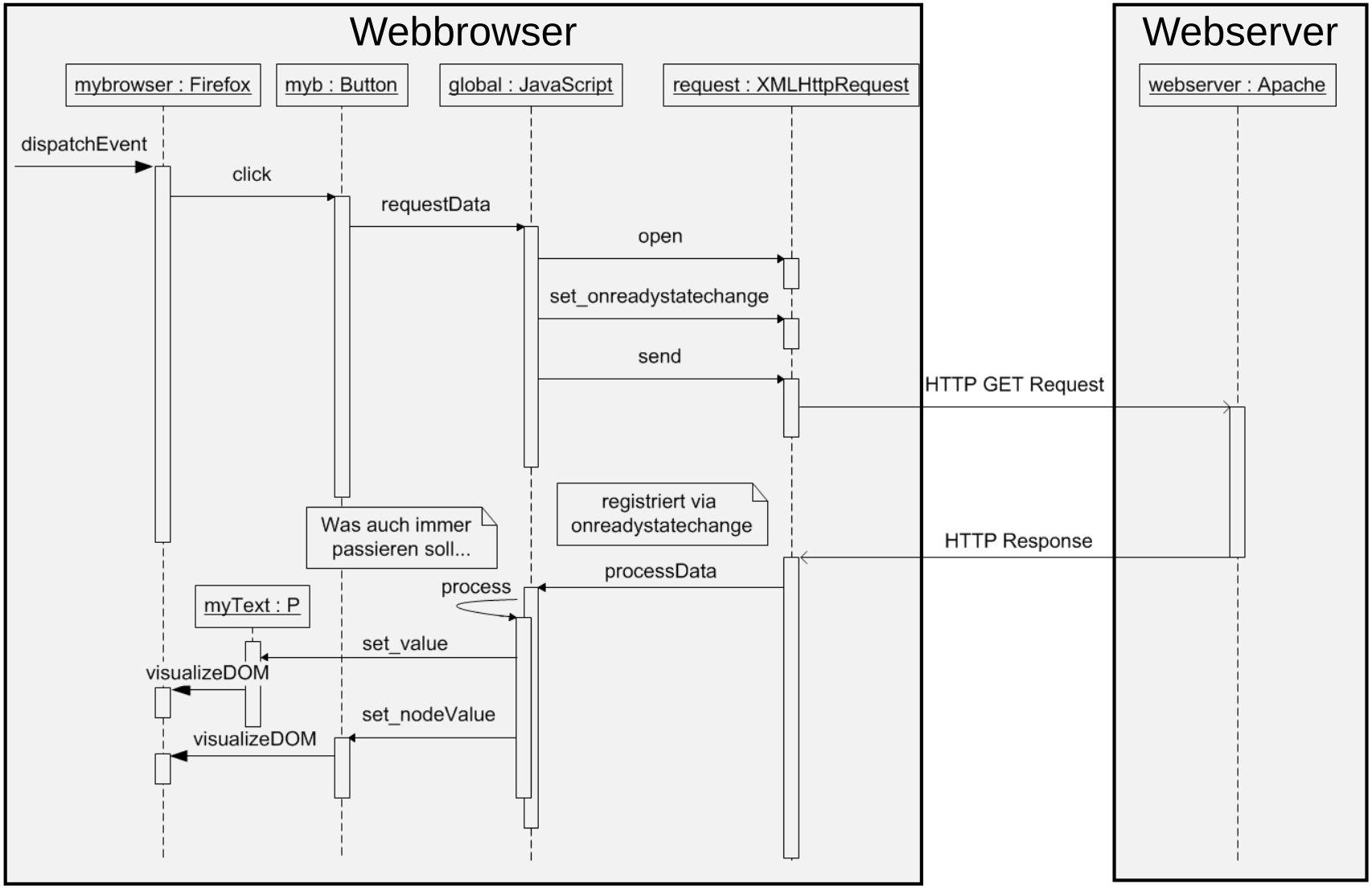
# Hinführendes Beispiel – Teil 2

- Es sollen nun Daten eingefügt werden, die der Webserver liefert
  - ⇨ Ein Webserver bietet die Möglichkeit Programme aufzurufen z.B. PHP-Code um die aktuelle Uhrzeit als String abzufragen
  - ⇨ aber dann wartet ("hängt") der Browser bis der Webserver antwortet
  - ⇨ und wie kann man neue Daten an eine bereits geladenen HTML-Seite übertragen, ohne die ganze Seite neu zu laden?

- `XMLHttpRequest` ermöglicht genau diese Funktion
  - ⇨ Festlegung einer Funktion zur Verarbeitung von (zurückkommenden) Daten ("Callback-Handler")
  - ⇨ Festlegen der Abfrage (URL etc.)
  - ⇨ Absenden der Abfrage mit asynchroner Antwort

```
var request = new XMLHttpRequest();                  // RequestObject anlegen
request.open("GET", "zeit.php");                     // URL für HTTP-GET festlegen
request.onreadystatechange = processData;            // Callback-Handler zuordnen
request.send();                                      // Request abschicken
```

# Sequenzdiagramm zu Beispiel 2



# Codeauszüge zu Beispiel 2 (Teil 1)

```
<!-- HTML wie in Beispiel 1 -->
<input type="button" id="myb" value="Los!" onclick="requestData()"/>
...

// request als globale Variable anlegen (haesslich, aber bequem)
var request = new XMLHttpRequest(); // für Firefox & IE7

function requestData() { // Daten asynchron anfordern
 request.open("GET", "php/0_zeit.php"); // URL für HTTP-GET
 request.onreadystatechange = processData; //Callback-Handler zuordnen
 request.send(null);   // Request abschicken
}
```

# Codeauszüge zu Beispiel 2 (Teil 2)

```
function processData() {
  if(request.readyState == 4) {        // Uebertragung = DONE
       if (request.status == 200) {  // HTTP-Status = OK
         if(request.responseText != null)
              process(request.responseText);// Daten verarbeiten
         else error ("Dokument ist leer");
       } else error ("Uebertragung fehlgeschlagen");
  } else ;                             // Uebertragung laeuft noch
}

function process (intext) {           // Text ins DOM einfuegen
   var myText = document.getElementById("myText");
   myText.firstChild.nodeValue = intext;
   document.getElementById("myb").value="Fertig!";
}
```

# XMLHttpRequest im Detail (1)

http://www.w3.org/TR/XMLHttpRequest

aus W3C Working Draft:

```
interface XMLHttpRequest
{
// event handler
attribute EventListener onreadystatechange;     // Zuordnung der Callback-Funktion

// state
const unsigned short UNSENT = 0;
const unsigned short OPEN = 1;
const unsigned short SENT = 2;
const unsigned short LOADING = 3;
const unsigned short DONE = 4;
readonly attribute unsigned short readyState;     // aktueller Zustand der Übertragung
↓
```

# XMLHttpRequest im Detail (2)

↓

// request

ggf. mit GET-Daten

```
void open(in DOMString method, in DOMString url);       // Anforderung vorbereiten
void open(in DOMString method, in DOMString url, in boolean async);
void open(in DOMString method, in DOMString url, in boolean async,
          in DOMString user);
void open(in DOMString method, in DOMString url, in boolean async,
          in DOMString user, in DOMString password);

void setRequestHeader(in DOMString header, in DOMString value);     // für HTTP

void send();                                            // Anforderung abschicken
void send(in DOMString data);
void send(in Document data);
```

POST-Daten

```
void abort();                                           // Übermittlung abbrechen
```

↓

# XMLHttpRequest im Detail (3)

↓

```
// response
DOMString getAllResponseHeaders();              // HTTP Header der Antwort
DOMString getResponseHeader(in DOMString header);

readonly attribute DOMString responseText;      // Ergebnis als Text ...
readonly attribute Document responseXML;        // ... oder XML DOM

readonly attribute unsigned short status;       // HTTP Status als Nummer ...
readonly attribute DOMString statusText;        // ... oder Klartext
};
```

- XMLHttpRequest sendet und erwartet standardmäßig UTF-8
  - ⇨ wenn das Gesamtsystem mit UTF-8 arbeitet, sind keine besonderen Maßnahmen erforderlich
  - ⇨ sonst muss in PHP mit utf8_decode ausgewertet und mit utf8_encode geantwortet werden

# XML in AJAX

- Bisher haben wir nur einfache Strings vom Webserver übertragen und überhaupt kein XML verwendet
  - ⇨ damit lassen sich aber nur schwer komplexere Datensätze übertragen
  - ⇨ `XMLHttpRequest` erlaubt auch die Übertragung von XML
  - ⇨ der Zugriff erfolgt dann für einen `request` über `request.responseXML` (statt `request.responseText`)
  - ⇨ der XML-Ausdruck wird automatisch eingelesen und als ein separates DOM zur Verfügung gestellt
  - ⇨ der Zugriff erfolgt mit normalen DOM-Methoden – also z.B. `request.responseXML.getElementsByTagName('zeit');`

- Das Umwandeln der Daten von bzw. nach XML ist umständlich
  - ⇨ "Serialisierung" der Objekte mit Attributen
  - ⇨ XML ist zwar sehr flexibel – aber es geht einfacher, wenn man diese Flexibilität nicht braucht

# Alternative Datenübertragung mit JSON

- JSON (JavaScript Object Notation)
  - ⇨ Einfache und schlanke Notation zur Darstellung von Listen, Strings, Zahlen und assoziativen Arrays
  - ⇨ die gängigen Programmiersprachen bieten eine JSON Bibliothek zum Serialisieren von Objekten nach JSON
  - ⇨ der ECMA Script Befehl `JSON.parse()` deserialisiert ein übergebenes Argument und liefert das Ergebnis als Object (z.B. assoziatives Array)
    - `eval()` geht auch, ist aber anfällig für Code Injection

Objekt "Kunde" in XML:

```
<kunde>
  <id>481048</id>
  <name>James Bond</name>
  <email>hurz@glgl.de</email>
  <artikel>17</artikel>
  <artikel>22</artikel>
</kunde >
```

Objekt "Kunde" in JSON:

```
{
  "id" : 481048,
  "name" : "James Bond",
  "email" : "hurz@glgl.de",
  "artikel" : [
    17,
    22
  ]
}
```

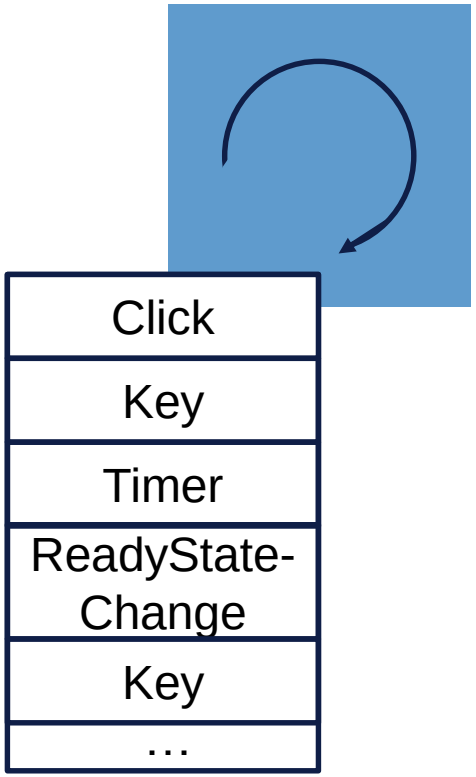
Notation für Literale in Javascript !

# AJAX – was kann bzw. muss man noch machen?

- Mit einem `XMLHttpRequest` können auch Daten an den Server geschickt werden
  - ⇨ per GET: request.open("GET", "script.php?name="+encodeURI(value))
  - ⇨ per POST: request.send("name="+value)
  - ⇨ das auf dem Webserver aufgerufene Programm (url aus request.open) muss die Daten dann entsprechend verarbeiten

- Bei der Datenübertragung und bei der Darstellung kann sehr viel schief gehen
  - ⇨ die verschiedenen Browser reagieren oft unterschiedlich
    - z.B. unterscheidet sich das Anlegen des `XMLHttpRequest`s beim IE6
    - beim IE7 kommt es auf die Reihenfolge an, in welcher der `XMLHttpRequest` "gefüllt" wird (erst open dann onreadystatechange)
    - ältere Browser unterstützen AJAX überhaupt nicht
  - ⇨ ordentliche Fehlerbehandlung und gründliche Tests sind sehr wichtig

# Event-Loop und Worker Threads




- Ein Browserfenster muss viele Events verarbeiten
  - ⇨ Seitenaufbau, Handler, Timer Callbacks, XmlHttpRequest Callbacks etc.
- Es gibt einen GUI Thread pro Browser-Tab bzw. -Fenster
  - ⇨ in diesem Thread läuft eine Message Loop
  - ⇨ Alle Aktivitäten werden in diesem Thread sequentialisiert
  - ⇨ Aktivitäten unterbrechen sich nicht gegenseitig
- Die Laufzeit von Handlern muss "kurz" sein, sonst friert die GUI ein!
- Für aufwändige Berechnungen gibt es separate Threads:
  - ⇨ "Web Worker"
  - ⇨ haben keinen direkten Zugriff auf DOM

# AJAX – bitte etwas bequemer!?

- Das Auslesen und Übertragen der Daten in die HTML-Seite mit DOM-Aufrufen ist umständlich
  - ⇨ Egal ob mit JSON oder XML
- Das Erzeugen von schicken Effekten im Stil von Web 2.0 ist sehr aufwändig
  - ⇨ über Javascript können die Style-Optionen so beeinflusst werden, dass derartige Effekte entstehen:
    Einblenden, Einschweben, Pulsieren, Blinken etc.
  - ⇨ Drag & Drop über Mouseover-Effekte
  - ⇨ hat im engeren Sinn aber nichts mit AJAX zu tun, weil es auch ohne asynchrone Datenübertragung funktioniert
- Das Unterstützen der verschiedenen Browser ist ein Albtraum
  - ⇨ unterschiedliche Schnittstellen, Bugs und Marktpolitik erschweren die browserübergreifende Implementierung

# AJAX - / Javascript-Frameworks

- Es gibt diverse Frameworks, welche Bedienelemente, Animationen, Drag&Drop, Hilfsfunktionen für DOM uvm. zur Verfügung stellen
  - ⇨ JavaScript-Bibliotheken mit Funktionen, die eingebunden werden
    - Dojo unter http://dojotoolkit.org
    - prototype.js unter http://www.prototypejs.org
    - Script.aculo.us unter http://script.aculo.us (basiert auf prototype.js)
    - jQuery unter http://jquery.com/

  Die Frameworks verwenden oft unterschiedliche Paradigmen!

  - ⇨ Mit solchen Frameworks ist der Einsatz von Effekten sehr einfach. Zum Beispiel mit Script.aculo.us :

```
var myText = document.getElementById("myText");
myText.appear();
myText.pulsate({pulses: 5, duration: 1.5 });
<h1 onclick="this.fade();">Gleich kommt's!</h1>
```

- Ein gutes Design will aber trotzdem gelernt sein...
  - ⇨ allein durch Effekte werden weder Inhalt noch Layout besser

# Vor- und Nachteile

- Technisch:
  - ⇨ kein Plug-In erforderlich
    - aber JavaScript muss aktiv sein
  - ⇨ aufwendig zu testen
    - wie jede richtige Applikation
  - ⇨ leidet unter Browser Bugs
    - wie alles im Web
  - ⇨ MVC-Architektur vermisst Push vom Server
    - Polling als Ersatz
  - ⇨ Ladezeit für Frameworks vs. schnelleres Nachladen
  - ⇨ Barrierefreiheit leidet

- sieht aus wie ein Browser, verhält sich aber nicht so:
  - ⇨ kein Zurück-Button
    - Benutzer erwarten ein "Undo"
  - ⇨ kein Lesezeichen setzbar
  - ⇨ von Suchmaschinen nicht auffindbar
  - ⇨ visuelles Feedback sollte Benutzer vertrösten
    - Sanduhr während Ladezeit

Fazit: Setzen Sie AJAX mit Maß und Verstand ein!

# Zusammenfassung I

- Skripte: Große „Freiheit“ in der Entwicklung
  - ⇨ Irgendwie funktioniert es – oder auch nicht!
  - ⇨ Mache das Browser-Ergebnis möglichst eindeutig
    - seit HTML5 ist das DOM zu jedem HTML-Konstrukt genau definiert
    - vor HTML5 erzeugten verschiedene Browser verschiedene DOMs zu einer Webseite – und verhielten sich entsprechend unterschiedlich
    - zuverlässige Adressierung verwenden (z.B. über id); nicht auf eine feste Position in einer Liste verlassen
  - ⇨ Unterschiedliches Browser-Verhalten "knapp außerhalb" des Standards (z.B. xxx&euroxxx ohne ; ) wird vom Firefox verstanden
  - ⇨ Konzepte vorher überlegen (siehe Abschnitt Entwurf)
  - ⇨ an den Standard halten
  - ⇨ Tools verwenden und gründlich Testen
    - Validatoren für HTML, CSS
    - Codeanalyse mit JSLint
    - Skript-Debugger

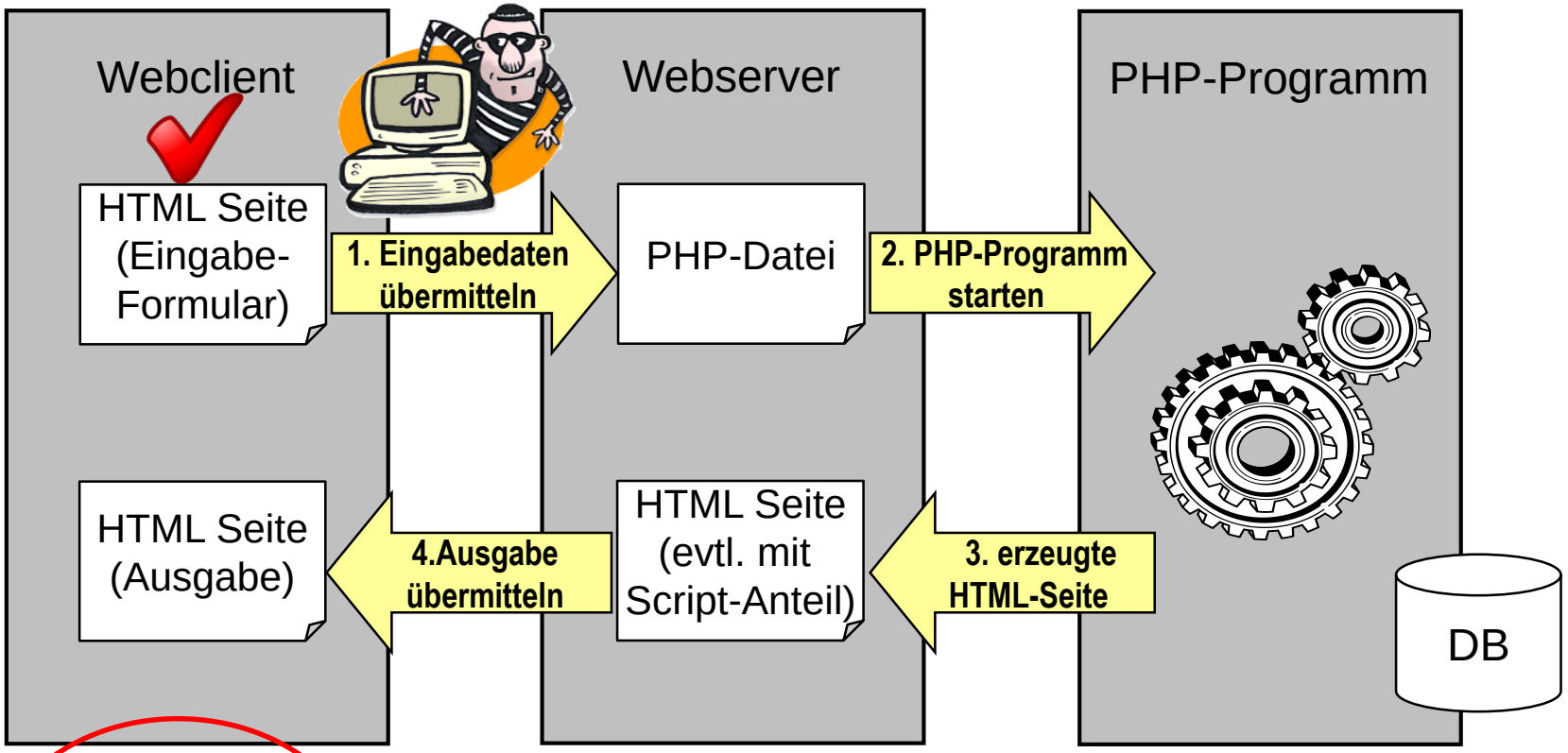
z.B. Browser-Plugins für Firefox verwenden: Firebug, WebDeveloper, HTML-Validator, DOM Inspector...

# Zusammenfassung II

- ECMA-Script-Grundlagen
  - ⇨ Grundidee, Grundgerüst, HTML-Einbindung
  - ⇨ Standardisierte Schreibregeln und Syntax
  - ⇨ Objektbasierend
- Document Object Model (DOM)
  - ⇨ Grundidee, Sinn und Zweck, Standard
  - ⇨ Zugriffsmöglichkeiten auf Knoten, Attribute und Styles
  - ⇨ Baumoperationen
  - ⇨ Einbettung in ECMA-Script
  - ⇨ Arbeiten mit ECMA-Script
- Ajax

Jetzt wissen Sie alles um eine dynamisch änderbare HTML-Seite zu entwickeln, die Formulardaten überprüft!

# Übersicht




- HTML
- CSS
- ECMA-Script
- DOM
- AJAX

- HTTP

- Server-Konfiguration

- CGI
- PHP
- MySQL

# Hochschule Darmstadt
# Fachbereich Informatik

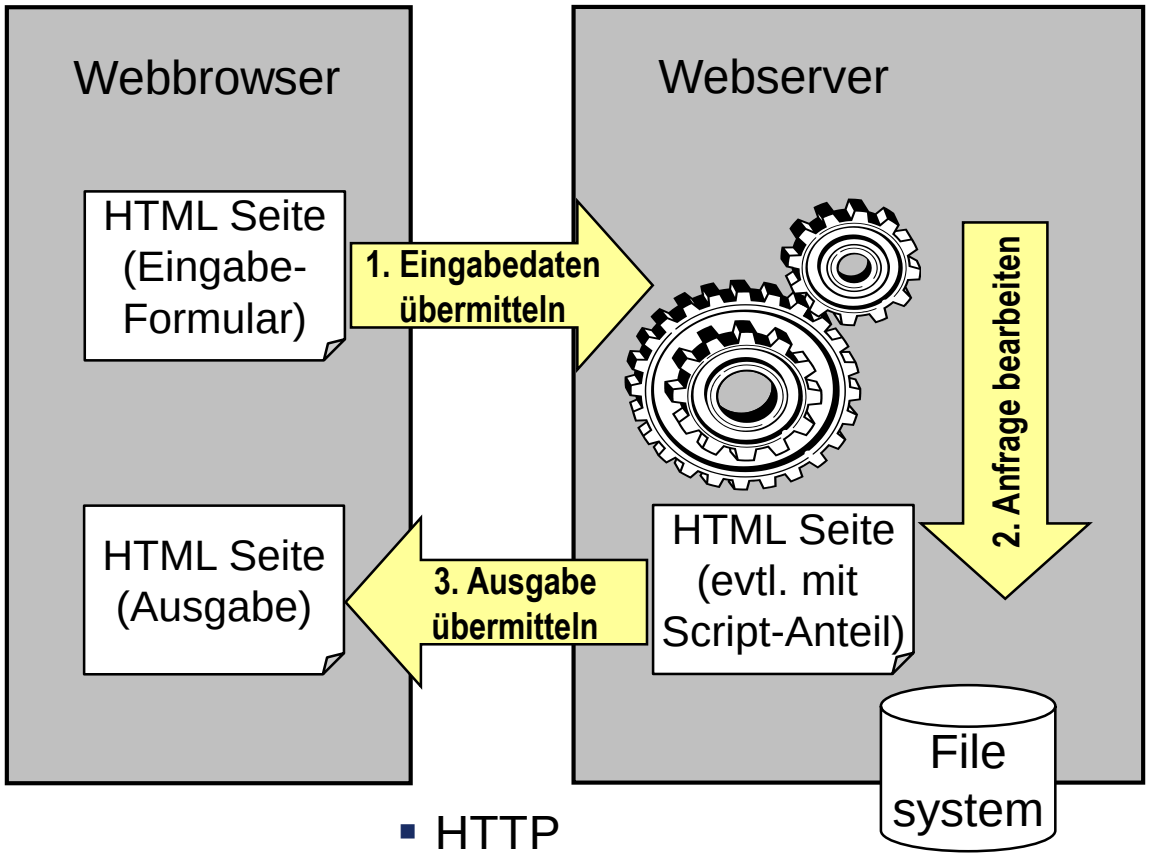
## 3. Webserver

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Einleitung

- Was ist ein Webserver?
  - ⇨ eine (spezielle) Software
  - ⇨ übermittelt auf Anfrage Daten mit dem HTTP-Protokoll
- Was braucht ein Webserver?
  - ⇨ TCP/IP-Unterstützung
    (vom Betriebssystem; darauf setzt das HTTP-Protokoll auf)
  - ⇨ Internet-Zugang (sinnvoll, aber für die Funktion nicht nötig)
- Was ist zu tun?
  - ⇨ Installation
  - ⇨ Zuordnung der "öffentlichen" Daten / Verzeichnisse
  - ⇨ Anbindung an andere Software (Skripte, Office,…)
  - ⇨ Konfiguration der Berechtigungen

# Der Webserver



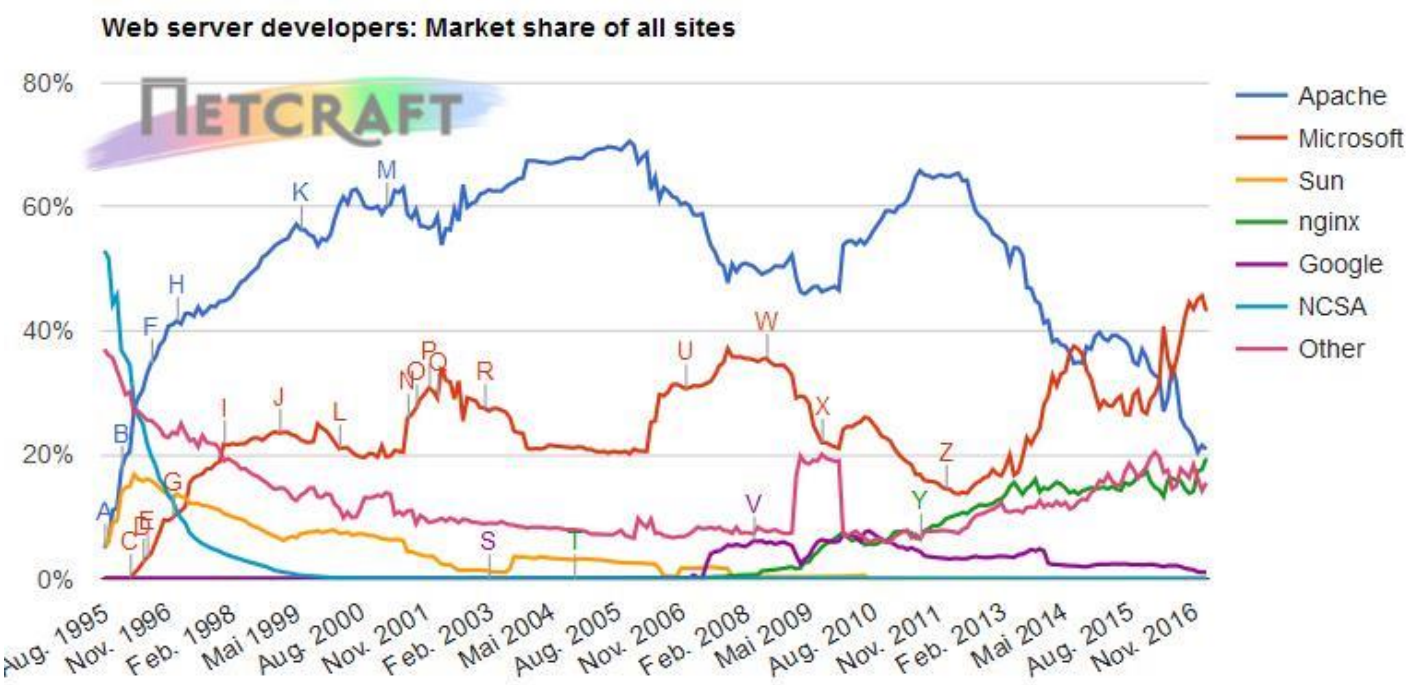

- HTTP
- Server-Konfiguration

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.1 Webserver Software

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Marktanteil von Webservern




| Developer | January 2017 | Percent | February 2017 | Percent | Change |
|---|---|---|---|---|---|
| Microsoft | 821,905,283 | 45.66% | 773,552,454 | 43.16% | -2.50 |
| Apache | 387,211,503 | 21.51% | 374,297,080 | 20.89% | -0.63 |
| nginx | 317,398,317 | 17.63% | 348,025,788 | 19.42% | 1.79 |
| Google | 17,933,762 | 1.00% | 18,438,702 | 1.03% | 0.03 |

Quelle:

http://news.netcraft.com/

Web Server Survey: 02/2017
„Market Share of all Sites"

# Verfügbare Webserver (Auswahl)

- Apache http://httpd.apache.org
  - ⇨ Weit verbreitet im Web. OpenSource-Produkt und Freeware. für UNIX-Plattformen und für MS Windows/DOS verfügbar.
- Microsoft's Internet Information Server (IIS)
  - ⇨ Kommerzieller Webserver für Windows Server
- Google Web Server (GWS)
  - ⇨ Google betreibt damit ca. 10 Millionen eigene Websites, Blogs etc. GWS steht der Allgemeinheit nicht zur Verfügung
- nginx
  - ⇨ freier Webserver unter BSD-Lizenz
  - ⇨ kleiner und schlanker Webserver
- lighttpd
  - ⇨ freier Webserver unter BSD-Lizenz mit Optimierung auf Massendaten
  - ⇨ eingesetzt z.B. bei YouTube oder SourceForge

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.1.1 Webserver:
## Installation und Konfiguration

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Allgemeines

- Konfiguration erfolgt je nach Webserver entweder Dialog-gesteuert, oder über Konfig-Datei
- Webserver läuft entweder als Anwendung oder als Prozess im Hintergrund (Dienst)
- Verwendung des Webservers ist auch lokal (ohne Internetzugang) möglich (z.B. zu Testzwecken)
- Manche Webserver unterstützen „Virtual Hosts“, d.h. mehrere Web-Zugänge werden auf einem Server realisiert
- Die Konfiguration der Webserver unterscheidet sich. Details in der jeweiligen Dokumentation.
- Typischerweise drei Benutzerrollen (relevant für Zugriffsrechte):
  - ⇨ Administrator
  - ⇨ Anbieter von Inhalten/Webseiten
  - ⇨ Besucher der Webseiten *im Skript meist "User" bzw. "Benutzer" genannt*

# Grundeinstellungen allgemein (1)

- IP-Adresse und Hostnamen des Servers
  - ⇨ Für lokalen Betrieb: 127.0.0.1, ::1 oder localhost.
    Test: Im Web-Browser (nach Start des Webservers)
    http://localhost/ aufrufen.
- Port des Servers
  - ⇨ normalerweise Port 80
- HTTP-Wurzelverzeichnis für HTML-Dateien
  - ⇨ Pfadname (je nach Syntax Ihres Betriebssystems) unterhalb dessen sich die lokalen HTML-Dateien befinden. z.B. c:\www\myhtml unter Windows
- Default-HTML-Dateiname für Verzeichnisse
  - ⇨ index.html oder index.htm

# Grundeinstellungen allgemein (2)

- Physisches Verzeichnis für ausführbare Scripts
  - ⇨ normalerweise cgi-bin
    Pfadname (je nach Syntax Ihres Betriebssystems) mit CGI-Scripts z.B. c:\www\cgi-bin
- Virtuelles Verzeichnis für CGI-Scripts
  - ⇨ normalerweise /cgi-bin
    Pfadname zu den CGI-Scripts für WWW-Zugriffe
    Zugriff über http://localhost/cgi-bin/myCGI.pl
- Pfad zu Perl-Interpreter und anderen Interpretern
  - ⇨ z.B. c:\programme\perl\bin\perl.exe unter Windows

# Grundeinstellungen allgemein (3)

- Log-Dateien
  - ⇨ Protokollierung der Zugriffe: access.log
  - ⇨ Fehlerprotokollierung: error.log
- Timeouts
  - ⇨ für Senden und Empfangen: 60 (= eine Minute)
  - ⇨ Die Angaben erfolgen in der Regel in Sekunden
- MIME-Typen
  - ⇨ Dateiformate, die der Webserver kennt und an den aufrufenden Web-Browser überträgt
  - ⇨ Andere Dateitypen sendet der Server nicht korrekt bzw. mit dem eingestellten Standard-MIME-Typ (text/plain)

# Apache

- im April 1995 erstmals in einer Version 0.6.2 publiziert
- Open-Source-Entwicklung (steht jedem kostenlos zur Verfügung)
- 1999 Gründung der Apache Software Foundation
- weltweit am häufigsten eingesetzte Webserver
  - ⇨ http://httpd.apache.org/
  - ⇨ aktuelle Version (03/2017): Apache 2.4.25

- Einfache Installation im Paket: XAMPP
  - ⇨ Apache Distribution enthält MySQL (bzw. MariaDB), PHP, Perl uvm.
  - ⇨ verfügbar für Linux, Windows, Mac, Solaris
  - ⇨ http://www.apachefriends.org/de/index.html

# Apache Grundeinstellungen

- Konfig-Datei `httpd.conf` im Verzeichnis *…\xampp\apache\conf*
  - ⇨ Definition und Verwendung von Variablen
    `Define DOCROOT "C:/web"`
    Wurzelverzeichnis der Apache-Installation
    `ServerRoot "C:/xampp/apache"`

Um die Wartung zu vereinfachen, sollten mehrfach auftretende Pfade als Variablen definiert werden! (DRY: Don't repeat yourself)

  - ⇨ Port, über den der Server kommuniziert (Standard)
    `Listen 80`
  - ⇨ eMail-Adresse des Administrators für Probleme
    `ServerAdmin na.me@h-da.de`
  - ⇨ Mapping URL ⇨ Verzeichnis für die Startseite (wenn der Besucher nur den Servernamen als URL eingibt, landet er in diesem Verzeichnis)
    `DocumentRoot "C:/xampp/apache/htdocs"` oder besser
    `DocumentRoot ${DOCROOT}` nach vorherigem `Define`
  - ⇨ Suchreihenfolge nach Dateinamen, falls URL ohne Dateinamen gegeben
    `DirectoryIndex index.html index.php`

# Mapping für weitere Verzeichnisse

- Alias definiert die Abbildung URL ⇨ Verzeichnis
  - ⇨ Dokumente können in anderen Verzeichnissen abgelegt werden als mit DocumentRoot festgelegt wurde
  - ⇨ Beim Zugriff auf ein Alias über eine URL wird Groß-Kleinschreibung unterschieden - auch unter Windows
  - ⇨
```
Define MANUAL "C:/Dokumentation"
Alias /manual ${MANUAL}
```

- ScriptAlias definiert für Server-Skripte die Abbildung URL ⇨ Verzeichnis
  - ⇨ d.h. für Dateien, die nicht zum Client gesendet sondern im Server ausgeführt werden
  - ⇨
```
Define PHP "C:/php"
ScriptAlias /php/ ${PHP}/
```

Der / am Ende bewirkt, dass auch beim Zugriff in der URL ein / angegeben werden muss!

# Mapping für Anbieter-Verzeichnisse

- Gliederung der Dokumente nach Anbietern
  - ⇨ weitere Variante zur Definition der Abbildung URL ⇨ Verzeichnis
  - ⇨ z.B. für persönliche Homepages von Dozenten auf www.fbi.h-da.de

- Aufruf der Startseite mit ~Anbietername
  - ⇨ z.B. http://www.fbi.h-da.de/~r.hahn

```
<IfModule mod_userdir.c>
    UserDir "C:/Anbieterverzeichnisse/"
</IfModule>
```

UserDir bezieht sich nicht auf die Besucher der Webseite!

- Anbieter-Verzeichnisse werden i.a. von den Anbietern selbst gepflegt
  - ⇨ eventuell mit eigenen Berechtigungs-Dateien (.htaccess)
  - ⇨ Zugriffsrechte gut überlegen und steuern mit `AllowOverride`

# Optionen für Verzeichnisse

- Einstellungen für Verzeichnisse werden innerhalb einer `<Directory>`-Anweisung gesetzt
  - ⇨ CGI-Skripte ausführen
    - `ExecCGI`
  - ⇨ Verknüpfungen zu anderen Verzeichnissen folgen
    - `FollowSymLinks  SymLinksIfOwnerMatch`
  - ⇨ Inhaltsverzeichnis zeigen, wenn Indexdatei (z.B. index.html) fehlt
    - `Indexes`
  - ⇨ Standardeinstellung setzen
    - `All`
  - ⇨ Beispiel:

```
Define EWA "C:/Links/ewa"
Alias /ewa ${EWA}
<Directory ${EWA}>
   Options ExecCGI Indexes FollowSymLinks
</Directory>
```

Options x y z setzt neu für dieses Verzeichnis;
Options +x +y akkumuliert mit vererbten Options

# MIME-Types

- HTTP-Header kennzeichnet das beigefügte Dokument mit dessen MIME-Type
  - ⇨ Multipurpose Internet Mail Extension
- Server ermittelt MIME-Type aus Datei-Endung
  - ⇨ Zuordnung gängiger Typen in /conf/mime.types
    `TypesConfig conf/mime.types`
  - ⇨ zusätzliche Definitionen in /conf/httpd.conf
    `AddType application/vnd.ms-excel .csv`
  - ⇨ Standardvorgabe, falls kein MIME-Type ermittelt werden kann
    `DefaultType text/plain` oder
    `DefaultType application/octet-stream`
- Browser entscheidet, wie das Dokument dargestellt wird
  - ⇨ was nicht angezeigt werden kann, wird zum Download angeboten
  - ⇨ Mozilla verwendet den übermittelten MIME-Type,
    Internet Explorer verwendet die Datei-Endung

# Log-Dateien

- Log-Datei für Fehlermeldungen
  ```
  ErrorLog logs/error.log
  ```
- Log-Datei für Zugriffe
  ```
  CustomLog logs/access.log access
  ```
- Woher kamen die Verweise ?
  ```
  CustomLog logs/referer.log referer
  ```
- Welche Browser wurden verwendet ?
  ```
  CustomLog logs/agent.log agent
  ```
- Alternativ: alles zusammen
  ```
  CustomLog logs/common.log common
  ```
- eigenes Logging-Format definieren
  ```
  LogFormat "Formatstring" Name
  ```

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.1.2 Webserver: Zugriffsschutz und Sicherheit

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Zugriffsschutz allgemein

- Ein Webserver ermöglicht den Zugriff auf Dateien im Filesystem
  - ⇨ es muss festgelegt werden, wer auf welche Verzeichnisse zugreifen darf
    - z.B. nur bestimmte IPs, bestimmte Domains usw.
- Fehler bei der Konfiguration sind kritisch.
  Deshalb werden die Rechte in 2 Stufen definiert:
  1. Zentraler Zugriffsschutz
     - Grundlegende Rechte werden innerhalb der Apache-Konfiguration (httpd.conf) festgelegt.
     - Diese Datei kann nur von einem Administrator bearbeitet werden
     - Änderungen an dieser Datei erfordern einen Neustart des Apache
  2. Lokaler Zugriffsschutz (z.B. für Anbieterverzeichnisse)
     - Diese Rechte können von den Anbietern der Inhalte vergeben werden
     - Änderungen erfordern keinen Neustart des Webservers
     - In der httpd.conf muss aktiviert werden, dass der Webserver in bestimmten Verzeichnissen nach einer bestimmten Datei (.htaccess) sucht und deren Inhalt als Zugriffsregeln interpretiert

# Zentraler Zugriffsschutz in httpd.conf

- Standardeinstellung sehr restriktiv

```
⇨ <Directory />
    AllowOverride None          .htaccess wirkungslos
    Require all denied          niemand hat Zugriff
  </Directory>
```

Unix root

alles andere muss dann explizit freigegeben werden

- (Dokumenten-)Verzeichnisse gezielt öffnen

```
⇨ <Directory "C:/Programme/Apache/htdocs">
    AllowOverride All           .htaccess wird akzeptiert
    Require all granted         alle haben Zugriff
  </Directory>
```

Verzeichnis-bezogener Schutz wird an Unter-verzeichnisse vererbt

- Dateiname für verzeichnisspezifischen Schutz festlegen

```
⇨ AccessFileName .htaccess      Name festlegen
  <Files ".ht*">
     Require all denied         und Zugriff über Browser sperren
  </Files>
```

# Lokaler Zugriffsschutz mit Berechtigungsdateien

- Der Webserver soll im Verzeichnis C:\myDir eine Datei `.htaccess` mit lokalen Zugriffsberechtigungen akzeptieren
  - ⇨ Voraussetzung (in der `httpd.conf`)
    - Der Name der Berechtigungsdatei entspricht dem festgelegten Namen (üblicherweise .htaccess):
      ```
      AccessFileName .htaccess
      ```
    - Die Verwendung von lokalen Berechtigungsdateien ist (für das betroffenen Verzeichnis) erlaubt
      ```
      <Directory "C:\myDir">
          AllowOverride All
          Require all granted
      </Directory>
      ```

Großzügige Freigaben sollten lokal eingeschränkt werden!

  - ⇨ Die Berechtigungsdatei bezieht sich auf das Verzeichnis, in dem sie steht
    - darin werden Bedingungen formuliert, die für den Zugriff erfüllt sein müssen
    - z.B. nur lokaler Zugriff durch `Require local`

# Zugriffsschutz mit Require

Require ersetzt deny/allow!

- Innerhalb der .htaccess-Dateien oder auch in Directory-Containern der httpd.conf können Zugriffsregeln definiert werden:
  - ⇨ Einzelne Regeln mit `Require,` z.B.
    - `Require all granted` allen den Zugriff erlauben
    - `Require all denied` allen den Zugriff verweigern
    - `Require local` alle lokalen Zugriffe zulassen (IPv4, IPv6,…)
    - `Require not host .de` Zugriffe erlauben, die nicht von *.de kommen
    - `Require ip 192.168.2` Zugriffe von IP 192.168.2.* erlauben
  - ⇨ Gruppen von Regeln mit
    - `<RequireAny>` – eine der Bedingungen muss erfüllt sein
    - `<RequireAll>` – alle aufgeführte Bedingungen müssen erfüllt sein
    - `<RequireNone>` – keine der Regeln darf erfüllt sein
    - diese Anweisungsblöcke können auch verschachtelt werden

Es gibt noch viele weitere Konfigurationsmöglichkeiten
httpd.apache.org/docs/current/mod/mod_authz_core.html

# Zugriffsschutz mit lokaler Passwortdatei

- Authentisierung
  - ⇨ Neben der Anbindung an Verzeichnisdienste (z.B. LDAP) unterstützt Require auch lokale Passwortdateien
  - ⇨ ein Anbieter kann so seine eigene Zugangsliste mit Accounts verwenden
- Verwendung einer lokalen Passwortdatei für ein Verzeichnis
  - ⇨ .htaccess (in dem Verzeichnis)

```
AuthType Basic
AuthName "Text für Authentisierungs-Popup"
AuthUserFile c:\pw.sec
Require valid-user
```

  - ⇨ Passwort-Datei `pw.sec` an sicherem Ort ablegen!
    Erzeugen mit: `htpasswd.exe -c pw.sec Besucher`

```
hahn:$apr1$dFJ6Tos3$3A3C1ldjFtCH3A7.cVDVD0
studi:$apr1$sWyY6auL$N.lh9cvH/fo7YdUMHcxX0/
```

# Zugriffsschutz - Beispiele

a) "alle" - offen für die ganze Welt

```
Require all granted
```

b) "niemand" – im Web nicht verfügbar

```
Require all denied
```

c) "alle außer… " Rechner von bestimmten Domains

```
<RequireAll>
 Require not host .to
 Require not host .tv
</RequireAll>
```

d) "niemand, außer…" bestimmten IP-Adressen

```
< RequireAny>
   Require ip 141.100
   Require ip 192.168
</RequireAny>
```

e) "niemand, außer mit Passwort"

```
AuthType Basic
AuthName "Anzeigetext"
AuthUserFile c:\pw.sec
Require valid-user
```

# Sicherheit von Apache-Passwörtern

- Basic Authentification wird quasi im Klartext (base64-kodiert) übertragen und die HTML-Antwort ebenfalls
  - ⇨ etwas besser: MD5 Digest; noch besser: verschlüsselte Verbindung
- Passwort wird im Browser gespeichert und bei jeder Seitenanforderung an denselben Server übermittelt
  - ⇨ notwendig, weil HTTP zustandslos ist
- Username/Passwort ist im Server nur durch schwache Verschlüsselung geschützt
  - ⇨ nicht dasselbe Passwort für Website und Bankkonto verwenden !
- Webserver erkennt keine Einbruchsversuche durch Ausprobieren (mehrfach falsches Passwort eingegeben)
  - ⇨ Betriebssysteme erkennen dies üblicherweise
  - ⇨ allenfalls in Log-Datei erkennbar

# Zugriffsrechte des Servers selbst

- Sicherheitsmaßnahme, falls Zugriffsrechte nicht sauber und vollständig definiert sein sollten
  - ⇨ soll den Server selbst schützen, weniger die Dokumente
- Apache wird normalerweise vom User "system" (root) gestartet
- Apache startet Child-Prozesse, die die Requests beantworten
- Child-Prozesse können eingeschränkte Zugriffsrechte haben
  - ⇨ User und Group z.B. so konfigurieren, dass
    - nur die freigegebenen Verzeichnisse lesbar sind
    - nur das Nötigste via CGI schreibbar ist
  - ⇨ Problem unter Unix: verschiedene User (Anbieter) gegeneinander abschotten

# Beliebte Fehler im Umgang mit Apache unter Windows

- Apache wurde unter Unix entwickelt und nach Windows portiert
  - ⇨ Die Konfiguration über httpd.conf erfordert genaue Einhaltung einer (oft inkonsistenten) Syntax
    - bei Fehlern in der Konfiguration lässt sich Apache nicht mehr starten!
    - Pfade brauchen teilweise am Ende einen "/"
    - Manchmal wird "/" verwendet, manchmal "\"
    - unbedingt die Beispiele in den Kommentaren als Vorlage beachten !
  - ⇨ beim Zugriff auf ein Verzeichnis über ein Alias wird die richtige Schreibweise (für den Alias) sogar unter Windows erzwungen

Test auf Fehler in httpd.conf:
apache\bin\httpd.exe -t

- Windows ignoriert Groß-Kleinschreibung in Pfaden und Dateinamen
  - ⇨ Skripte und Links werden unter Windows auch gefunden, wenn die Groß-Kleinschreibung der Dateinamen falsch ist
  - ⇨ beim Verschieben eines Projekts auf einen Web Server unter Unix werden diese Dateien nicht mehr gefunden

# Zusammenfassung

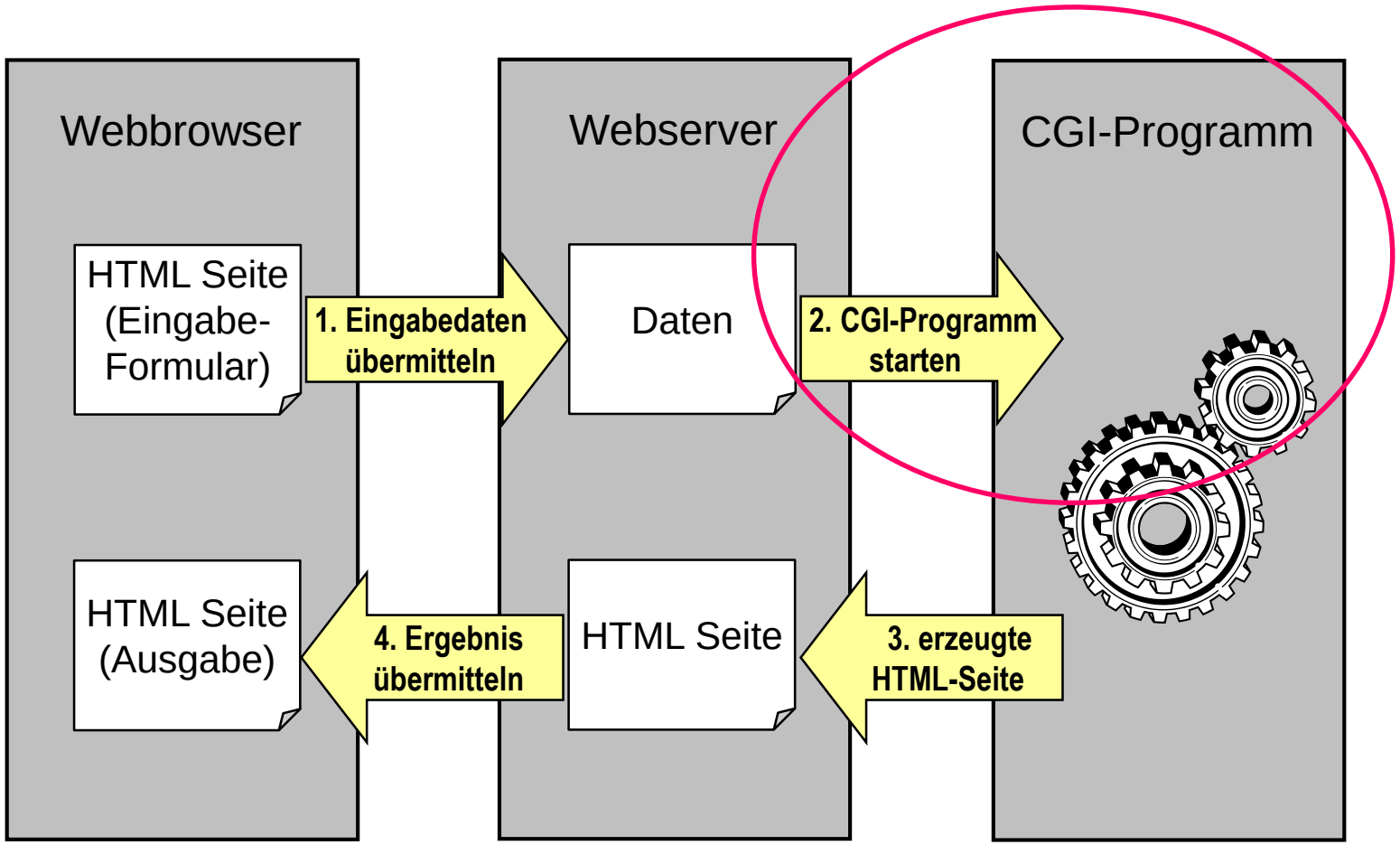
- Grundlagen
  - ⇨ Was ist ein Webserver?
  - ⇨ Webserver am Markt
- Grundeinstellungen
  - ⇨ IP-Adresse & Ports, Log-Dateien
  - ⇨ Verzeichnisse für Dokumente, Skripte, Anbieter
  - ⇨ Besondere Dateinamen (index.html, .htaccess,...)
  - ⇨ Pfade zu Skripten und ausführbaren Programmen (z.B. Perl)
  - ⇨ MIME-Typen und Dateiendungen
- Apache
  - ⇨ Grundeinstellungen
  - ⇨ Zugriffsberechtigungen und Zugriffsschutz

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.2 CGI

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# CGI - Common Gateway Interface



# CGI - Common Gateway Interface

Es gibt noch alternative Kommunikationswege! Wir beschränken uns auf 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 / unterstützt
- 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 .../cgi-bin/printenv.pl
(In Standard-Installation enthalten)

- Werte werden bei einem CGI-Aufruf vom Webserver gesetzt
  - Informationen zum Request
    ```
    REQUEST_URI=/cgi-bin/CgiTest.exe?Name=x&Kommentar=y
    SCRIPT_NAME=/cgi-bin/CgiTest.exe
    HTTP_REFERER=http://localhost/CgiTestFormular.htm
    HTTP_COOKIE=Keksname=Kruemel; nochEinKeks=hart
    ```
  - und je nach Methode
    ```
    REQUEST_METHOD=GET
    QUERY_STRING=Name=x&Kommentar=y
    ```
    sofern gesetzt
  - oder
    ```
    REQUEST_METHOD=POST
    CONTENT_LENGTH=28
    CONTENT_TYPE=application/x-www-form-urlencoded
    QUERY_STRING=      (ist leer)
      Formulardaten hier aus der Standardeingabe (cin in C++)
    ```
    Auf das Ende achten!

# Umgebungsvariablen (2)

- 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
```

# Umgebungsvariablen (3)

- Informationen über den Client (Browser)

```
HTTP_ACCEPT=image/gif, 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
```

Es können auch noch die installierten Fonts und Browser Plugins abgefragt werden!
Diese Daten machen Ihren Rechner in der Regel eindeutig erkennbar
vgl. https://panopticlick.eff.org/

# Codierung von Parametern in einer URL (I)

der Browser macht das automatisch
in PHP: urlencode()
in ECMAScript: encodeURI()

- 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
  - ⇨ Achtung: `<a href="http://www.xyz.de?x=5&amp;y=7">`
- 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 Parametern in einer URL (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
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
   - als Bilddaten im GIF oder JPEG-Format

Es gibt kein Daten-Ende-Zeichen

# 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>" << endl;
          cout << "<html lang=\"de\">" << 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)


```perl
# 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

# Apache für CGI konfigurieren

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

```
ScriptAlias /cgi-bin/ "C:/xampp/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-cgi.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

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"

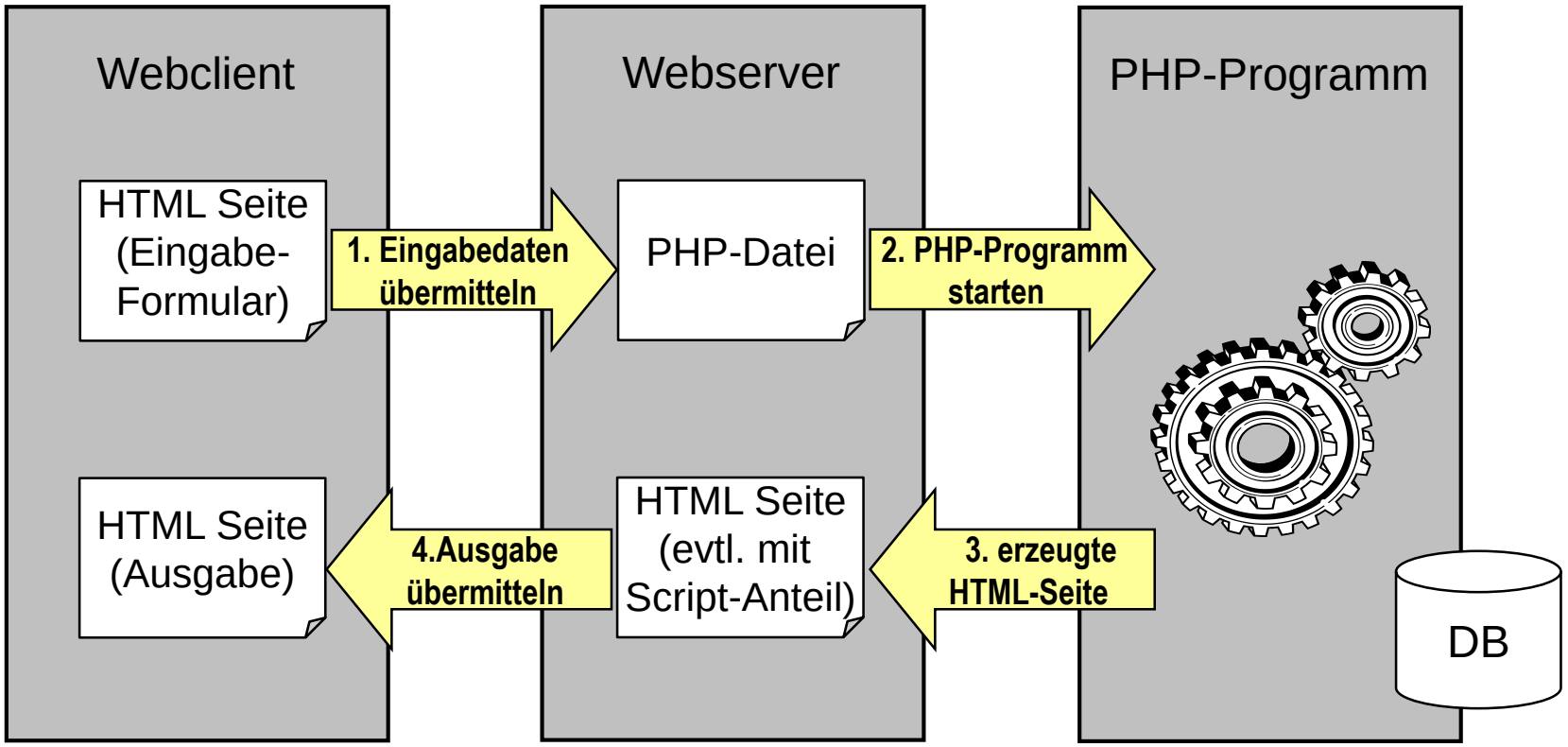
    content="3; URL=/cgi-bin/script.pl">
  - ⇨ Anwendung: nach 3 Sekunden die neue URL aufrufen

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3 PHP

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Übersicht




- HTML
- CSS
- ECMA-Script
- DOM
- AJAX

- Server-Konfiguration

- CGI

- PHP
- MySQLi
- Seitenklassen
- Frameworks

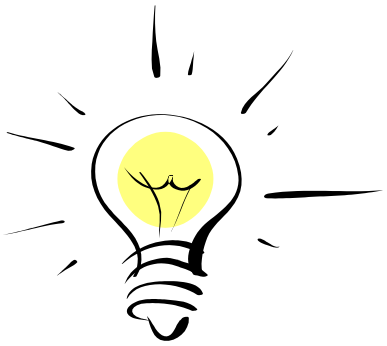
# Situation

- 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
  - ⇨ 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 ECMAScript)
- ⇨ 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

diese ursprüngliche Idee ist längst veraltet

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)
  - ⇨ echte Komposition/Assoziation von Objekten immer noch nicht unterstützt

Open Source

- 2003: PHP 5.0
  - ⇨ diverse Optimierungen
  - ⇨ endlich richtige Objektorientierung
- 2009: PHP 5.2.8 in der XAMPP-Suite enthalten (kein PHP 4 mehr!)
- 2015: PHP 7 erscheint und ist seit XAMPP7 in der Suite enthalten

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3.1 PHP Grundlagen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Primitives Beispiel

```
<?php
  header ("Content-type: text/html");
?>
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8" />
  <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
    `<? ... ?>` `<?= $variable ?>` `<% ... %>`
    `<script language="php"> ... </script>`
- 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++
- dynamisch 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: $Anzahl</td>");
    ```
  - ⇨ Sonderzeichen wie in C++: `\n` `\t` `\"` `\\` `\$`
  - ⇨ `'...'` ist ebenfalls möglich als Stringklammer, allerdings werden darin keine Variablen und Sonderzeichen ausgewertet
  - ⇨ damit sind "geschachtelte Strings" möglich, z.B. HTML-Attribute in PHP-Strings
    ```
    echo ('<p class="Kopf">');
    ```
- Verkettung von Strings mit dem Operator `.`
  - ⇨ `$Begruessung = "Hallo ".$Name;`
- Zugriff auf einzelne Character (beginnt bei 0)
  - ⇨ `$Zeichen = $MeinString[$Zeichenposition];`

Der Zugriff über `$MeinString{$Pos};` ist deprecated!

- \0 ist (im Gegensatz zu C++) keine Endemarke, ausgenommen jedoch Funktionen, die als "nicht binary safe" markiert sind

# 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
```
  - ⇨
```
$oFarbe = array('Erdbeere'=>'rot', 'Banane'=>'gelb');
```
    // Zuweisung als Tupel (Item, Value): $oFarbe['Erdbeere'] = 'rot'
- Abarbeitung indiziert:
  - ⇨
```
for ($i=0; $i<count($arr); $i++)
  echo ($arr[$i]);
```
- Abarbeitung als Kollektion:
  - ⇨
```
foreach($oFarbe as $obst => $farbe)
   echo (" $obst hat $farbe \n");
```

# Beispiel: Umgebungsvariablen mit PHP

```
<?php header ("Content-type: text/html"); ?>
<!DOCTYPE html>
<html lang="de">
<head>
    <meta charset="UTF-8" />
        <title>Umgebungsvariablen</title>
    </head>
    <body>
        <h2>Umgebungsvariablen</h2>
        <pre>
<?php
 foreach($_ENV as $key => $value) {
   echo "$key=$value\n";
 }
?>
        </pre>
   </body>
   </html>
```

$_ENV erfordert in PHP.ini:
variables_order = "EGPCS"

# Globale assoziative Arrays

"Super-Globals" erfordern keine global-Deklaration

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 (falls in php.ini aktiviert)
- `$_SERVER` Umgebungsvariablen vom Server
- `$_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 bis Version 5.4 (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

  - ⇨ vor PHP 5.4 in php.ini: register_globals = Off
- vor PHP 5.4: magic_quotes_gpc = Off (sonst wird " zu \")
- `isset` prüft jeweils, ob die Variable überhaupt existiert

# 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");
 ?>
```

oder gleich $_REQUEST verwenden

# Aufteilung von PHP-Code in Dateien

kleiner Nachteil:
PHP muss immer
mehrere Dateien öffnen

- 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 einbinden

```
require_once "Pfadname.php";
```

  - ⇨ vergleichbar mit #include in C++
  - ⇨ `require_once` und `include_once` vermeiden Endlos-Einbindungen
  - ⇨ 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 !

- Variablen sind "require-übergreifend" sichtbar

# Konfiguration von PHP

- `php.ini` ist zentrale Konfigurationsdatei
- seit PHP 5.4 ohnehin abgeschafft:
  - ⇨ `register_globals = Off`

    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 7.0.13**

| | |
|---|---|
| System | Windows NT RAHADA15 10.0 build 14393 (Windows 10) i586 |
| Build Date | Nov 8 2016 13:30:03 |
| Compiler | MSVC14 (Visual C++ 2015) |
| Architecture | x86 |
| Configure Command | cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-sdk\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--without-analyzer" "--with-pgo" |
| Server API | Apache 2.0 Handler |
| Virtual Directory Support | enabled |
| Configuration File (php.ini) Path | C:\WINDOWS |
| Loaded Configuration File | C:\xampp\php\php.ini |
| Scan this dir for additional .ini files | (none) |
| Additional .ini files parsed | (none) |

# Apache für PHP konfigurieren

PHP-Interpreter wird für jeden Request erneut geladen

- PHP unter Verwendung des CGI
  - ⇨ PHP-Installationsverzeichnis zum Skript-Verzeichnis erklären:
    ```
    ScriptAlias /php/ "/apache/php/"
    ```
  - ⇨ PHP-Interpreter (CGI-Version) für Dateien mit MIME-Type PHP aufrufen:
    ```
    Action application/x-httpd-php /php/php-cgi.exe
    ```
  - ⇨ MIME-Type PHP der Datei-Endung .php zuordnen:
    ```
    AddType application/x-httpd-php .php
    ```
  - ⇨ auch index.php als Startseite zulassen:
    ```
    DirectoryIndex index.php
    ```
  - ⇨ CGI Ausführung erlauben (nicht nötig für Standard-PHP-Verzeichnisse)
    ```
    Options ExecCGI
    ```

PHP-Dateien können in jedem Dokument-Verzeichnis liegen

PHP-Interpreter wird einmalig bei Apache-Start geladen

- effizient: PHP als Apache-Module
  - ⇨ wird eingebunden, wenn Apache compiliert wird oder als Dynamic Shared Object (vgl. DLL unter Windows) beim Start
  - ⇨ In der Installation von XAMPP ist PHP5 bzw. PHP7 integriert

# Parallele Ausführung von Skripten

Im Internet ist das ganz normal ...

- mehrere Aufrufe desselben Skripts können gleichzeitig ausgeführt werden (d.h. quasi-parallel, nebenläufig, in verschiedenen Threads oder Prozessen)
  - ⇨ Sequenz (im Prinzip): Client1: $no = get_no_of_entries()<br>Client2: delete entry[0]<br>Client1: for (i=0 to $no-1) entry[i]=…
  - ⇨ Threads müssen sorgfältig programmiert sein, weil sie sich Speicher teilen; Prozesse kosten mehr Speicher und Rechenzeit, sind dafür sicherer
- Skript muss reentrant sein (kein Schreib-/Lesezugriff auf globalen Speicher) – was bedeutet das in PHP ?
  - ⇨ Apache erzeugte ursprünglich für jeden Request einen eigenen Prozess
    - kann seit 2.0 auch Multi-Threading; das wird für PHP nicht empfohlen!
  - ⇨ die PHP Laufzeitumgebung stellt jedem Request einen unabhängigen Satz von statischen und globalen Variablen zur Verfügung
    - damit wird auch bei threadbasierter Ausführung ein eigener Prozess simuliert
  - ⇨ aber: manche PHP-Bibliotheken nutzen eigenen globalen Speicher und sind daher nicht für threadbasierter Ausführung geeignet (meist nicht dokumentiert)
- Verwendung einer DB Storage Engine, die Transaktionen und Locking unterstützt (z.B. InnoDB oder BerkeleyDB, aber nicht MyISAM)
  - ⇨ Vorsicht: DB-Optimierungen schalten oft die Serialisierung ab

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3.2 PHP Entwicklung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# PHP Entwicklung

SELFPHP
http://www.selfphp.info

- Eigene Funktionen definieren mit *function*

```
function hello($text1, $text2)
{
    echo "Hello World <br></br>";
    echo "$text1 $text2 <br></br>";
    return true; // Rückgabewert
}
```

  - ⇨ Optionale Funktions-Parameter mit Default-Wert:
    `function hello($text1, $text2="Test")`
  - ⇨ Call by Reference
    `function change(&$myReference)`

- Während der Entwicklung: alle Fehlercodes ausgeben mit
  `error_reporting(E_ALL);`

Im Produktivbetrieb die Fehlermeldungen in Log-Datei schreiben und nicht ausliefern!

# Strings

- Ein String kann auf drei verschiedene Weisen geschrieben werden.
- Einfache Anführungszeichen (single quote) " als "

```
echo 'Variablen werden $nicht $ausgewertet\n';
//Ausgabe: Variablen werden $nicht $ausgewertet\n
```

- Doppelte Anführungszeichen (double quote) " als " oder \"

```
$myvar = 'Variable';
echo "{$myvar}n werden ausgewertet\n";
//Ausgabe: Variablen werden ausgewertet [newline]
```

komplexe Berechnungen und Abgrenzung von Variablen-namen in {}

- Heredoc Notation
  Stringbehandlung wie mit doppelten Anführungszeichen - nur ohne Anführungszeichen; Zeilenumbrüche werden übernommen

```
echo <<<EOT
{$myvar}n werden ausgewertet
EOT;
//Ausgabe: Variablen werden ausgewertet [newline]
```

kein Semikolon!

" als "

Schluss"tag" muss in der ersten Spalte stehen - und ganz alleine!

# Strings und Zeichenkodierung in PHP

- PHP Strings sind single byte strings (1 Byte pro Zeichen)
  - ⇨ passt perfekt zu ISO-8859-x Zeichensätzen
  - ⇨ aber UTF-8 stellt manche Zeichen mit 2 bis 4 Bytes dar
- manche Stringfunktionen arbeiten daher nicht richtig für Umlaute
  - ⇨ `strlen` und `strpos` zählen falsch
- Ausweg: PHP Extension mbstring
  - ⇨ kein default, muss explizit installiert werden (in XAMPP enthalten)
  - ⇨ bietet Ersatz für die gängigen Stringfunktionen
  - ⇨ Funktionen erkennbar am Präfix `mb_`
  - ⇨ Initialisierung: `mb_internal_encoding("UTF-8");`

```
<?php
$myString = "ääääh?";
echo "strlen: ".strlen($myString)."<br>";
echo "mb_strlen: ".mb_strlen($myString,
'utf8');
?>
```

Alle PHP Funktionen,
die mit Strings arbeiten,
sorgfältig testen !

# Metazeichen in Strings

hier sind nicht Umlaute gemeint, sondern Zeichen, die in der Syntax von HTML / ECMAScript / SQL eine besondere Bedeutung haben

- im erzeugten HTML bzw. den ECMAScript Anteilen muss die jeweilige Syntax beachtet werden

- das Stringformat erfordert die Ersetzung mancher Sonderzeichen
  - ⇨ Stringbegrenzer ist `"` und muss zur Ausgabe "escaped" werden: `\"`
  - ⇨ für ECMA-Script sogar "doppelt": `&quot;`

- für die Weiterverarbeitung und Speicherung (z.B. in der Datenbank) sollen Daten in "reiner Form" stehen

  - ⇨ PHP bietet diverse Funktionen zum Wandeln von Strings zwischen den Formaten

# Metazeichen in Strings - sprachübergreifend

- übermittelte Formulardaten werden Strings für
  - ⇨ HTML zum Anzeigen (ersetzt & < > " ' )
    ```
    $Wert = htmlspecialchars($Wert);
    ```
  - ⇨ SQL für Anfragen etc. (ersetzt " ' )
    ```
    $Wert = $mysqli->real_escape_string($Wert);
    ```

`htmlentities()` ersetzt auch die Umlaute!

- Erzeugen von JavaScript in HTML aus PHP-echo:

PHP ⬇

Browser ⬇

```
echo("<p onclick=\"alert(&quot;Hallo&quot;);\">");
     <p onclick="alert(&quot;Hallo&quot;);">
                alert("Hallo");
```

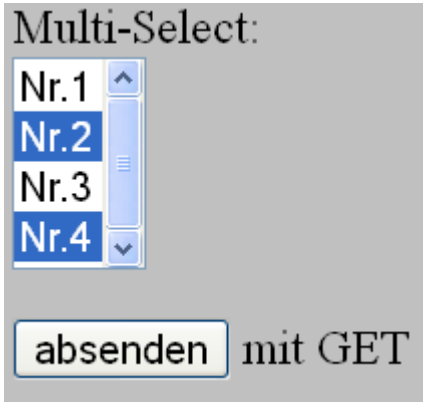
- Einfacher mit heredoc-Notation:

```
echo <<<EOT
<!– Hier steht der ganz normale HTML-Code -->
<!DOCTYPE…
</html>
EOT;
```

`$myvar = <<<LABEL ... LABEL;` liest den umschlossenen Text in die Variable $myvar !

# Übergabe mit Mehrfachauswahl – Das Problem

```
<form action="21_FormularEcho.php" method="get">
    <p>Multi-Select:</p>
    <p><select name="myselect" size="4" multiple>
        <option value="1">Nr.1</option>
        <option value="2">Nr.2</option>
        <option value="3">Nr.3</option>
        <option value="4">Nr.4</option>
    </select></p>
    <p><input type="submit" value="absenden"/> mit GET</p>
</form>
```




ergibt für GET: http://localhost/..../Echo.php?myselect=2&myselect=4

überträgt (URL): myselect=2&myselect=4

PHP liefert: myselect=4
(als Umg.variable in $_GET)

PHP

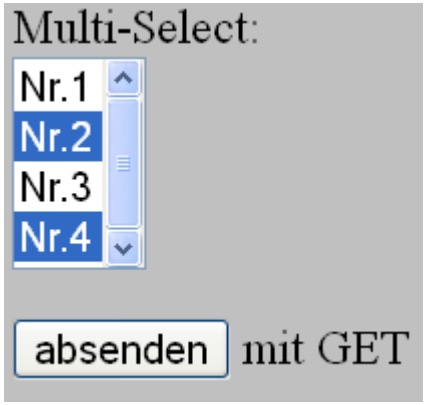
mit POST sieht man nichts, aber das Problem bleibt!

⇨ Es wird nur der letzte Wert als Umgebungsvariable übernommen!

# Übergabe mit Mehrfachauswahl – Die Lösung

```
<form action="21_FormularEcho.php" method="get">
    <p>Multi-Select:</p>
    <!-- ohne [] steht nur das letzte Select in PHP zur Verfügung -->
    <p><select name="myselect[]" size="4" multiple>
        <option value="1">Nr.1</option>
        <option value="2">Nr.2</option>
        <option value="3">Nr.3</option>
        <option value="4">Nr.4</option>
    </select></p>
    <p><input type="submit" value="absenden"/> mit GET</p>
</form>
```

array




[ ]

überträgt: myselect%5B%5D=2&myselect%5B%5D=4

PHP `print_r($_GET);` liefert jetzt:

Befehl zum Ausgeben von Arrays

Array ( [myselect] => Array ( [0] => 2 [1] => 4 ) ) myselect=Array

Zugriff auf Position `$i` über

```
$MYS = $_GET["myselect"];
echo ($MYS[$i]);
```

# Formulare und Sicherheit

später ausführlicher

- wer weiß schon, welche Daten ein gehacktes Formular übermittelt ...
- deshalb NICHT die übermittelten Parameter auswerten:

```
⇨ foreach ($_POST as $Name => $Wert)
      $Datensatz[$Name] = $Wert;
```

- sondern die erwarteten Parameter:

```
⇨ if (isset ($_POST["ElemA"]) &&
       is_numeric ($_POST["ElemA"])
      $Datensatz["ElemA"] = $_POST["ElemA"];
  Speichern ($Datensatz);
```

Syntax und Wertebereich überprüfen

# Überprüfung auf unerwünschte Zeichen

- am einfachsten mit regulären Ausdrücken
  - ⇨ `bool preg_match (string pattern, string subject);`

ereg vermeiden; ist nicht binary safe und deprecated

- Beispiele
  - ⇨ Dezimalzahl mit Vorzeichen und max. 10 Ziffern:
    - `$isDecimal = preg_match("/^-{0,1}[0-9]{1,10}$/", $p);`
    - Minus darf 0 oder 1-mal auftreten, dann 1 bis 10 Zeichen aus der Menge `0-9`
  - ⇨ übliche Bezeichner-Syntax:

```
$isAlphaNum = preg_match(
                    "/^[a-zA-Z][a-zA-Z0-9_]+$/", $p);
```

    - zuerst ein Buchstabe und dann Buchstaben, Zahlen oder _

# Objektorientierung (I)

brauchbar seit PHP 5

- Klassen und Objekte ähnlich Java
  - ⇨ Objekte werden ausschließlich mit `new` erzeugt und über Referenzen angesprochen
  - ⇨ Attribute deklarieren mit: `var $Attribut;`
    - besser noch: `public` / `protected` / `private $Attribut;`
  - ⇨ Basisklassenkonstruktor explizit aufrufen

    `parent::` bezeichnet Basisklasse

    `$this` verweist auf Objekt

    `->` für Zugriff auf Attribute (ohne `$`) und Methoden
- Vererbung
  - ⇨ `class a extends b { ... }` // aber keine Mehrfachvererbung
  - ⇨ dafür `interface i { ... }` und `class c implements i { ... }`
- wenig genutzt in älteren Bibliotheken
  - ⇨ Bibliotheken bestehen oft noch aus Gruppen von Funktionen
  - ⇨ OO-Beispiel: herstellerunabhängige Kapselung des Datenbankzugriffs in PHP Data Objects PDO

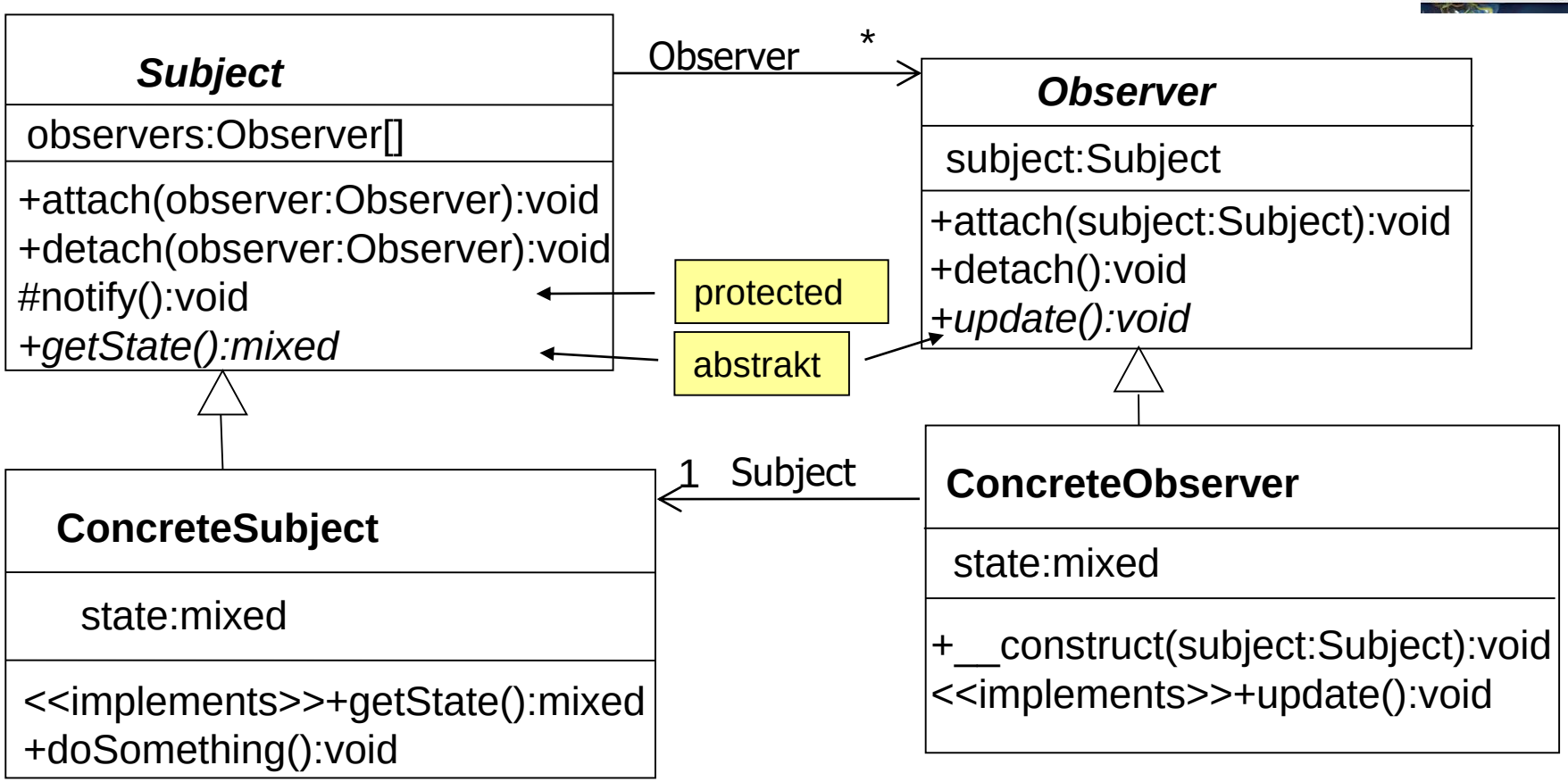
# Objektorientierung (II)

- "Neue" PHP-Sprachelemente im OO-Umfeld:
  - `public`, `protected`, `private` statt `var` //keine privaten Klassen
  - `static $x = 0; const y = 0;`
  - `$instance = new SimpleClass();` // weist Referenz zu
  - `$Kopie = clone $instance;`
  - `class a extends b { ... }`
  - `function f1 (a $param) { ... }`
  - `abstract class ..., abstract function ...`
  - `interface i { ... }` `class c implements i { ... }`
  - `final function f2 { ... }` // wie Java; d.h. nicht virtual
  - magic methods mit dem Namen __xxx werden implizit aufgerufen
    `__construct`, `__destruct`, `__clone`
  - Vergleich: `==` gleiche Attributwerte `===` selbes Objekt

neu: Type Hinting für Klassen

Seit PHP5 sind die üblichen OO-Konzepte umsetzbar !

# Einsatz von Klassen am Beispiel Observermuster (I)


Quelle: S. Bergmann http://www.professionelle-softwareentwicklung-mit-php5.de

# Einsatz von Klassen am Beispiel Observermuster (II)

```php
require_once 'Observer.php';
abstract class Subject {
    protected $observers = array();
    public function attach(Observer $observer) {
        $this->observers[] = $observer;
    }
    public function detach(Observer $observer) {
        for ($i = 0; $i < sizeof($this->observers); $i++) {
            if ($this->observers[$i]===$observer) {
                unset($this->observers[$i]);
            }
        }
    }
    protected function notify() {
        for ($i=0; $i<sizeof($this->observers); $i++)
            if (isset($this->observers[$i]))
                $this->observers[$i]->update();
    }
}
```

Datentyp Observer bekannt machen

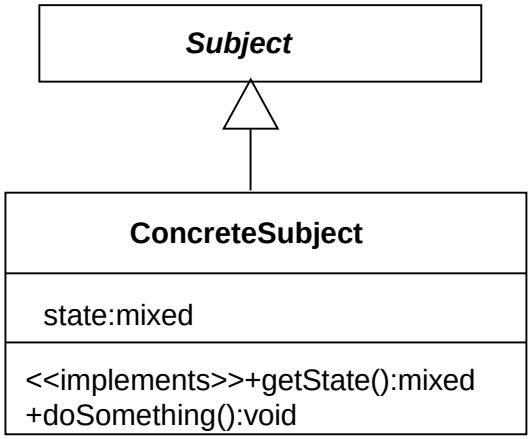
erzeugt Array mit Lücke(n)

| ***Subject*** |
| --- |
| observers:Observer[] |
| +attach(observer:Observer):void<br>+detach(observer:Observer):void<br>#notify():void<br>*+getState():mixed* |

Quelle: S. Bergmann http://www.professionelle-softwareentwicklung-mit-php5.de

# Einsatz von Klassen am Beispiel Observermuster (III)

```
class ConcreteSubject extends Subject {
    protected $state = NULL;
    public function getState() {
        return $this->state;
    }
    public function onEvent() {
        $this->notify();
    }
    //weitere Methoden
}
```




Observer und ConcreteObserver werden analog umgesetzt !

Quelle: S. Bergmann http://www.professionelle-softwareentwicklung-mit-php5.de

# Fehlerbehandlung – die traurige Art

- Häufige Art der Fehlerbehandlung
  - ⇨ sofortiger Abbruch
    ```
    die ("ungültiger Parameter");// Meldung an Browser
    ```
    oder
    ```
    exit (5);                    // Fehlercode an OS
    ```
  - ⇨ Unterdrückung der Fehlermeldungen mit dem Fehlerkontrolloperator `@`
    (darf vor jeder Expression stehen)
    ```
    @$f = fopen("myfile.txt","r");
    ```

- Normalerweise endet ein PHP-Programm am Dateiende und erzeugt dann (hoffentlich) eine sinnvolle HTML-Ausgabe
  - ⇨ Bei Fehlern entsteht eine unvollständige HTML-Datei und es hängt vom Browser ab, was angezeigt wird
  - ⇨ es entstehen häufig leere Seiten oder Layout-Trümmer
  - ⇨ Der Anwender erhält oft keine Rückmeldung wo das Problem liegt – geschweige denn, ob er etwas dagegen tun kann

# Ausnahmebehandlung – wie es sein sollte

- Seit PHP 5 gibt es echte Ausnahmebehandlung

```
try {  // hier kommt der problematische Aufruf
  if (...) {  // was auch immer schief gehen kann
    throw new Exception("Meine Fehlermeldung");
  }
}
catch (Exception $e) {            // typisiert !
  echo $e->getMessage();          // besser: komplettes HTML
}
```

  - ⇨ HTML-Schachtelungsstruktur kann ordentlich abgeschlossen werden
  - ⇨ sinnvolle Meldungen können gezielt angezeigt werden (z.B. Alternative Kontaktmöglichkeit oder Info, dass der Admin benachrichtigt wurde) etc.
  - ⇨ `finally` (vgl. Java und ECMAScript) gibt es nicht in PHP

# Ausnahmebehandlung: Beispiel

```
<?php
header ("Content-type: text/html");   // MIME-Type der Antwort definieren; vor HTML !
error_reporting (E_ALL);              // alle möglichen Fehlermeldungen aktivieren

function dividiere ($dividend, $divisor) {
    if ($divisor == 0){ // Es soll durch null geteilt werden -> Exception
        throw new Exception("Division durch null ist nicht definiert");
    }
    $erg = $dividend/$divisor;
    return $erg;
}
// hier kommt der HTML-Header und der Beginn des <body>
try {                               // leitet den Block ein, in dem Exceptions abgefangen werden
    dividiere(2,0);                 // generiert eine Exception
}
catch (Exception $fehler){          // fängt die Exception ab
    echo "<p>Ausnahmefehler: ".$fehler->getMessage()."</p>";
}
// hier kommen das Ende des <body> und die üblichen schließenden Tags
```

Abgewandelt von: C. Möhrke "Besser PHP programmieren"

# Ausnahmebehandlung für verschiedene Fälle

- Häufig sollen Fehler durch die Anwendung unterschieden werden
  - ⇨ das Öffnen einer nicht vorhandenen Datei erzeugt immer einen Fehler
  - ⇨ dieser Fehler ist manchmal kritisch (Datenbank nicht vorhanden) und manchmal eher nicht (css-Datei fehlt)
  - ⇨ Durch Ableiten von der Klasse `Exception` können verschiedene Fehlertypen definiert und unterschiedlich abgefangen werden
  - ⇨ Der Konstruktor muss definiert werden!

```
class criticalError extends Exception {
    public function __construct ($Message) {
        parent::__construct($Message);
    }
}
class noncriticalError extends Exception {…}
…
try{…}
catch (criticalError $fehler){…}
catch (noncriticalError $fehler){…}
```

Abgewandelt von: C. Möhrke "Besser PHP programmieren"

# PHP bietet sehr viele Funktionen

Nachbau der aus C++ bekannten STL:
Standard PHP Library (SPL)

- Array-Funktionen
- Crack-Funktionen
- Dateisystem-Funktionen
- Datums- und Zeit-Funktionen
- Functions-Funktionen
- Image-Funktionen
- Klassen und Objekt-Funktionen
- Kontroll-Mechanismen
- Mail-Funktionen
- Mathematische-Funktionen
- MySQL-Funktionen
- PDF-Funktionen
- PHP-Deklaration
- PHP-Informationen
- Reguläre Ausdrücke
- Session-Funktionen
- String-Funktionen
- URL-Funktionen
- Variablen-Funktionen
- Verzeichnis-Funktionen
- Sonstige-Funktionen

http://www.selfphp.de oder http://www.php.net/
bieten eine Übersicht der PHP-Funktionen

# Anwendung von PHP: PDF-Erzeugung

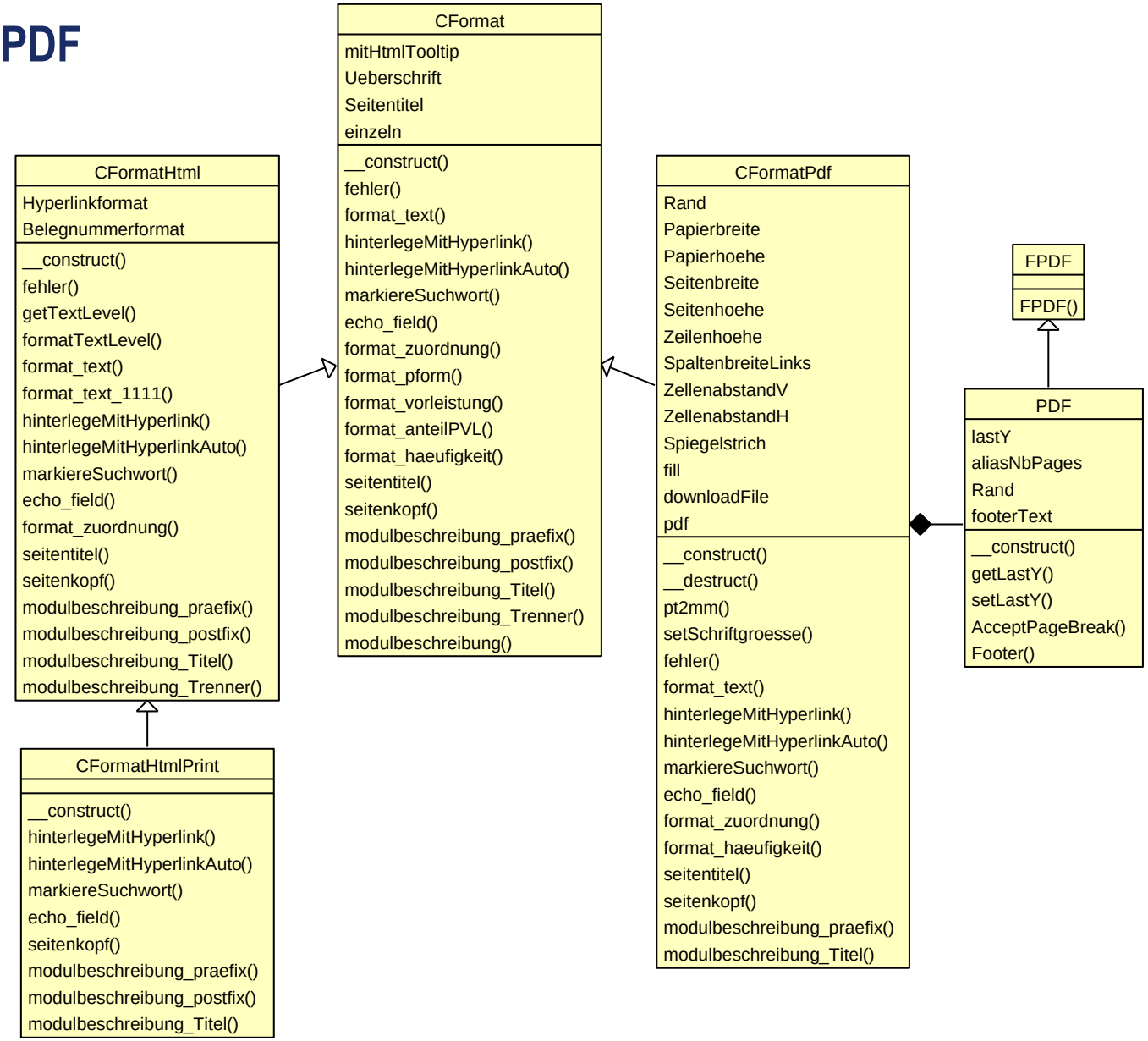
- Es gibt mehrere Bibliotheken zum Erzeugen von PDFs
  - ⇨ pdflib (http://www.pdflib.com)
    - ist bereits bei XAMPP dabei und kann als PHP-Modul aktiviert werden
    - für kommerzielle Anwendungen fallen Lizenzkosten an
  - ⇨ fpdflib (http://fpdf.org/)
    - eine PHP-Klasse, die Funktionen zum Erstellen von PDFs bietet
    - funktioniert ohne nennenswerte Installation
    - kostenlos und auch kommerziell frei einsetzbar

- PDF-Bibliotheken definieren ein spezielles DOM
  - ⇨ die Einarbeitung ist die größte Arbeit

```
<?php /* fpdf Tutorial 1*/
 require('fpdf.php');
 $pdf=new FPDF();
 $pdf->AddPage();
 $pdf->SetFont('Arial','B',16);
 $pdf->Cell(40,10,'Hello World!');
 $pdf->Output();
?>
```

# Multiview HTML / PDF

- Beispiel aus Modulhand-buch:
  - ⇨ HTML
  - ⇨ HTML für Druck
  - ⇨ PDF
- wartungs-freundlich durch Klassen-hierarchie



# Plattformunabhängigkeit ?

- ja: derselbe Interpreter für viele Betriebssysteme
- aber: jede Installation ist anders
  - ⇨ sehr viele Features abhängig von Konfigurationsparametern in PHP.INI
  - ⇨ und / oder auch von Compiler-Schaltern beim Build
  - ⇨ Abfragefunktionen in der Gruppe "PHP options & information"

besser wäre: in der Anwendung

- kein Problem, wenn der Interpreter nur eine Anwendung bedient
  - ⇨ dann kann man ihn passend konfigurieren (muss natürlich bei PHP-Update bewahrt werden)
  - ⇨ aber sehr nachteilig bei mehreren Anwendungen
- interessanter Weg zur Behinderung von Portabilität !

# Zusammenfassung

- Grundlagen
  - ⇨ Idee
  - ⇨ Historie
- Notation
  - ⇨ Grundstil C++
  - ⇨ Integration von HTML `<?php ... ?>`
  - ⇨ Variablennamen beginnen mit `$`
  - ⇨ Strings `.`, `"..."`, `'...'`, `echo <<<EOT`
  - ⇨ Arrays und deren Übergabe
  - ⇨ Umwandlung von Sonderzeichen
  - ⇨ Objektorientierung
  - ⇨ Ausnahmebehandlung

Jetzt können wir mächtige Skripts schreiben und laufen lassen
– aber was tun wir damit?

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3.3 Datenbankanbindung mit PHP

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Webserver und Datenbanken

- Man könnte Daten server-seitig in Dateien ablegen, besser ist aber fast immer eine Datenbank
  - ⇨ Shop- und Buchungssysteme, Content Management Systeme
- PHP bietet einfache Schnittstellen zu vielen Datenbanken
  - ⇨ PostgreSQL kostenlos unter PostgreSQL License
  - ⇨ MySQL kostenlos unter GPL; kostenpflichtige kommerzielle Lizenz
- Häufig im Bundle:
  Linux bzw. Windows + Apache + MySQL + PHP
  - ⇨ z.B. auch bei XAMPP

LAMP / WAMP

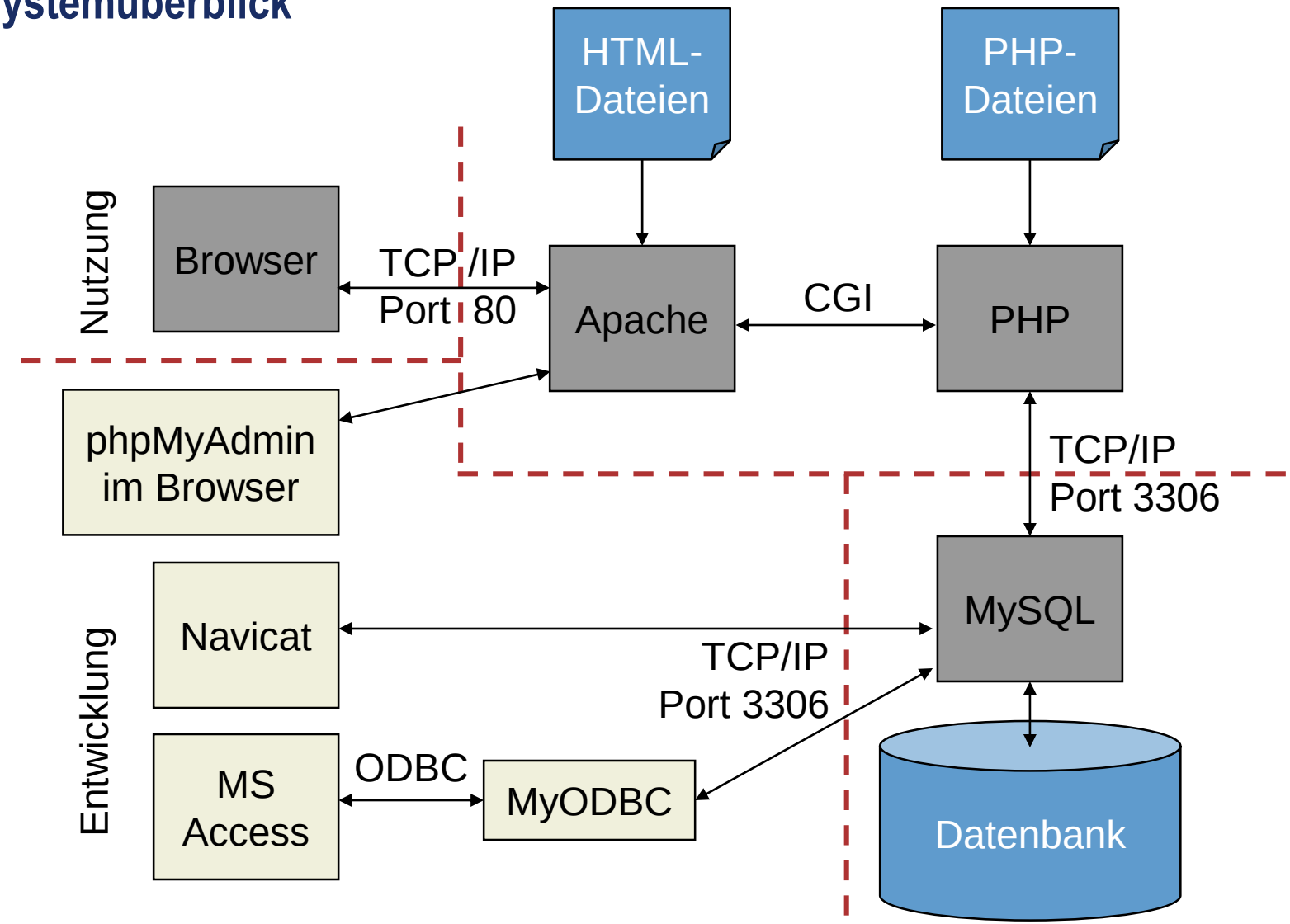
# Datenbank MySQL

- MySQL arbeitet wahlweise mit verschiedenen Storage Engines
  - ⇨ mit Transaktionen (Pro: Storage Engine InnoDB)
  - ⇨ ohne Transaktionen (Classic: Storage Engine MyISAM)

- die (PHP-)Anwendung selbst benötigt nur den Datenbank-Server mysqld.exe

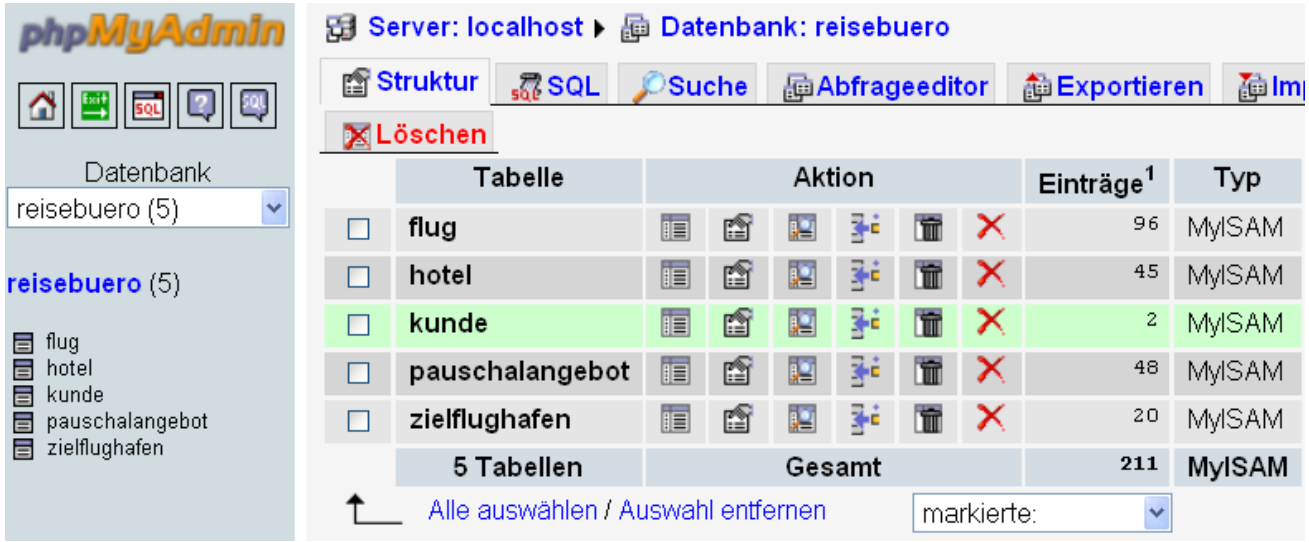
- ein Front-End ist erforderlich für DB-Entwicklung und -Administration
  - ⇨ z.B. Navicat für Windows, Mac, Linux (http://www.navicat.com/)
  - ⇨ als Webanwendung phpMyAdmin (http://localhost/phpmyadmin)
    - auch Remote verwendbar
    - wird mit XAMPP installiert
  - ⇨ sehr komfortabel durch QBE: Microsoft Access über MyODBC

# Systemüberblick



# Beispiel: Administration mit phpMyAdmin

- phpMyAdmin zur Bearbeitung einer MySQL-Datenbank





Zeichensatz: utf8
Kollation: utf8_unicode_ci
(vgl. http://dev.mysql.com/doc/refman/5.5/en/charset-unicode-sets.html)

# SQL - zur Erinnerung

- ganz primitiv: MySQL über die Kommandozeile
  - ⇨ mysql.exe ist das mitgelieferte Kommandozeilen-Front-End
- \mysql\bin\mysql.exe bzw. C:\xampp\mysql\bin\mysql.exe
  - ⇨ use Reisebuero

```
SELECT zimmerart, kategorie FROM hotel ORDER BY kategorie;
UPDATE hotel SET preis='150' WHERE preis='116';
DELETE FROM hotel WHERE ort='Alanya';
INSERT INTO zielflughafen (zielflughafen, land)
                          VALUES ('Rom', 'Italien');
```

  - ⇨ quit
- meist besser als PHP Algorithmen:
  - ⇨ Abfragen über mehrere Tabellen: INNER / LEFT / RIGHT JOIN
  - ⇨ Aggregatfunktionen: COUNT, MIN, MAX etc.
  - ⇨ Verschachtelte Abfragen durch Unterabfragen

# PHP-Schnittstellen zu MySQL

- MySQL - Extension
  - ⇨ ursprüngliche und immer noch häufig eingesetzte Anbindung
  - ⇨ rein prozedurale Schnittstelle
  - ⇨ Empfohlen für ältere MySQL-Versionen (< V 4.1.3)
- MySQLi - Extension (oft auch MySQL *improved*)
  - ⇨ Objektorientierte Schnittstelle (zusätzlich zur prozeduralen Schnittstelle)
  - ⇨ ab PHP5, für neuere MySQL-Versionen (≥ V 4.1.3) dringend empfohlen
  - ⇨ Diverse Verbesserungen: Prepared Statements, Multiple Statements, Transaktionen, verbessertes Debugging uvm.
- PHP Data Objects (PDO)
  - ⇨ Datenbank-Abstraktions-Schicht für PHP
  - ⇨ Einheitliche Schnittstelle unabhängig von der verwendeten Datenbank
  - ⇨ Austausch der Datenbank mit sehr geringem Aufwand, aber evtl. Einschränkung der verfügbaren Funktionalität durch Vereinheitlichung

Wir verwenden MySQLi!

# MySQLi Klassen

- Die MySQLi-Erweiterung enthält folgende Klassen

  ⇨ MySQLi
  - Aufbau und Verwaltung einer Verbindung zwischen PHP und dem MySQL-Server
  - Einstellung der Verbindungskonfiguration (z.B. SSL)

  ⇨ MySQLi_STMT
  - Verwaltung und Vorbereitung einer Datenbankabfrage ("prepared statement")
  - ermöglicht die Verknüpfung von PHP-Variablen mit Datenbankabfragen

  ⇨ MySQLi_Result
  - Verwaltung einer Menge von Ergebnissen, die als Antwort auf eine Query zurückgeliefert werden
  - Sollte nach der Verwendung wieder freigegeben werden

# MySQLi im Standardeinsatz

- TCP/IP-Verbindung aufbauen

```
$Connection = new MySQLi($host, $user, $pwd, $DB_name);
$Connection->set_charset("utf8");
```

  ⇨ viele (optionale) Parameter; liefert ein Objekt der Klasse `MySQLi`

- Ergebnistabelle abfragen oder SQL-Aktion ausführen

```
$Recordset = $Connection->query ($SQLabfrage);
```

  ⇨ liefert ein Objekt der Klasse `MySQLi_Result`

- nächste Zeile aus Ergebnistabelle in Array übertragen und auf einzelne Felder des Datensatzes zugreifen

```
while ($Record = $Recordset->fetch_assoc())
  $Wert = $Record['Feldname'];
```

- Speicher der Ergebnistabelle freigeben

```
$Recordset->free();
```

- TCP/IP-Verbindung schliessen

```
$Connection->close();
```

Fehlerbehandlung nicht vergessen !

# Beispiel: Verbindungsaufbau und Datenbankabfrage



```php
<?php
try {
    // MIME-Type der Antwort definieren (*vor* allem HTML):
    header("Content-type: text/html; charset=UTF-8");
    // alle möglichen Fehlermeldungen aktivieren:
    error_reporting (E_ALL);

    // Datenbank öffnen und abfragen:
    require_once 'pwd.php'; // Passwort & Co. einlesen
    $Connection = new MySQLi($host, $user, $pwd, "Reisebuero");

    // Verbindung prüfen:
    if (mysqli_connect_errno())
          throw new Exception("Connect failed: ".mysqli_connect_error());
    if (!$Connection->set_charset("utf8"))
          throw new Exception("Charset failed: ".$Connection->error);

    // SQL-Abfrage festlegen:
    $SQLabfrage = "SELECT Land FROM zielflughafen GROUP BY Land";
    $Recordset = $Connection->query ($SQLabfrage);
    if (!$Recordset)
          throw new Exception("Query failed: ".$Connection->error);
?>
```

PHP

SQL

# Beispiel: Generierung einer Auswahlliste


```
<!DOCTYPE html><html lang="de"><head> ... </head>
<body>
    <p>Bitte wählen Sie ein Land:</p>
    <form id="Auswahl" action="Result.php" method="post">
<?php
    $AnzahlRecords = $Recordset->num_rows;
    echo ("\t\t<p><select name=\"AuswahlLand\" size=\"$AnzahlRecords\">\n");
    while ($Record = $Recordset->fetch_assoc()) {
          echo ("\t\t\t<option>".htmlspecialchars($Record["Land"])."</option>\n");
    }
    $Recordset->free();
    $Connection->close();
    echo ("\t\t</select></p>\n");
?>
          <p><input type="submit" value="Flughäfen anzeigen"/></p>
    </form>
    <p><input type="button" value="Flughafen einfügen"
    onclick="window.location.href='Add.php'"/></p>
</body></html><?php
} catch (Exception $e) { echo $e->getMessage(); }
```

PHP

HTML, SQL und PHP sind stark vermischt ! Das erschwert die Wartung, aber solche Lösungen findet man häufig. Im nächsten Kapitel behandeln wir einen objektorientierten Ansatz.

HTML

# Spezialfall: Mehrere Datensätze im selben Formular

ähnlich zur "Übergabe mit Mehrfachauswahl" in Formularen !

- Üblicherweise verwendet man die Feldnamen aus der DB als Elementnamen im Formular
  - ⇨ Primärschlüssel bei Bedarf in <input type="hidden">
- bei mehreren Datensätzen Unterscheidung durch indizierte Namen
  - ⇨ HTML erlaubt [ ] im Attribut name; PHP macht daraus ein Array

// generieren:

```
$i = 0;
while ($Record = $Recordset->fetch_assoc()) {
    echo ("<input type=\"hidden\" name=\"id[$i]\" value=\"$Record['id']\">");
    echo ("<input type=\"text\" name=\"attr[$i]\" value=\"$Record['attr']\">");
    $i++;
}
```

// auswerten:

```
for ($i=0; $i<count($_POST['id']); $i++) {
    $id = $Connection->real_escape_string($_POST['id'][$i]);
    $attr = $Connection->real_escape_string($_POST['attr'][$i]);
    $SQL = "UPDATE table SET attr='$attr' WHERE id='$id';"
}
```

# Kritische Abschnitte sichern

Die Umsetzung unterscheidet sich stark für andere Datenbanken (Oracle, PostgreSQL) ! vgl. Isolation Level Serializable; Transaction Rollback ist unerwünscht

- Konsequenz der Parallel-verarbeitung unabhängiger HTTP-Requests
  - Zugriff auf gemeinsame Ressourcen (Dateien, Datensätze) muss gesichert sein
  - vgl. Mutex, Semaphore, Monitor
  - bei Datenbanken voneinander abhängige Mehrfachzugriffe mit Transaction und Lock einklammern
- eventuelle Fehler treten erst unter Last auf
  - schwer zu testen
  - Entwurf sorgfältig durchdenken

*z.B. Belegung beschränkter Plätze*

```
query("Begin Transaction;");
query("Lock Table belegt Write;");
$result = query(
  "Select count(*) From belegt;");
$anzahl = fetch_row($result)[0];
```

*hier darf kein anderer Prozess unterbrechen und ein Insert Into ausführen; anderer Prozess soll warten*

```
if ($anzahl < 16)
  query("Insert Into belegt ...");
else
  echo("ist schon voll");
query("Unlock Tables;");
query("Commit;");
```

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3.4 Systemarchitektur mit Seiten-Klassen

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Programmieren mit PHP

- Seit PHP 5 bietet PHP alles was man von einer modernen Programmiersprache erwartet:
  - ⇨ leider wird dies bei der Entwicklung häufig vergessen:
    - Funktions-Orientierung statt Objekt-Orientierung
    - undokumentierter Spaghetti-Code
    - Mischung von Algorithmen, Ausgabe, Datenbankzugriffen etc.
    - keine nennenswerte Ausnahmebehandlung
    - Verwendung von globalen Variablen
  - ⇨ dabei sind PHP-Programme häufig der Kern von großen Webauftritten, die in vielen Personenjahren entwickelt wurden
    - die Software wird weiterentwickelt
    - Wartung wird zunehmend wichtiger

Vergessen Sie nicht die ganz normalen (?!)
Regeln eines guten Software Designs
z.B. Geheimnisprinzip, starker Zusammenhalt, lose Kopplung...

# Programmierstil (I)

- Was halten Sie von folgendem Beispiel?

```
<?php
function Header($myTitle){
  echo "<html><head><title>";
  echo "$myTitle";
  echo "</title><body>";
}
function Footer(){
  echo "</body></html>";
}
/******* main ********/
  Header ("Hallo");
  echo "PHP ist toll!";
  Footer ();
?>
```

Stil:
"PHP mit HTML"

# Programmierstil (II)

- Was halten Sie von folgendem Beispiel?

```
<html>
<head>
    <title>Hallo</title>
</head>
<body>
  <?php echo "PHP ist toll!";?>
</body>
</html>
```

Stil:
"HTML mit PHP"

# Programmierstil (III)

- Es gibt verschiedene Möglichkeiten PHP und HTML zu kombinieren
  - ⇨ HTML mit eingebettetem PHP
    - ist übersichtlicher, wenn es vorwiegend um die HTML-Ausgabe geht
    - erlaubt die Erzeugung der HTML-Seite mit entsprechenden Tools. Die variablen Teile werden später als PHP-Abschnitte eingebaut.
  - ⇨ PHP mit eingebettetem HTML *geht auch objektorientiert*
    - ist übersichtlicher, wenn es vorwiegend um Berechnungen geht
    - also eher angebracht beim Auswerten von Daten, Datenbankoperationen, Algorithmen etc.
  - ⇨ die Heredoc-Notation kombiniert die Vorteile o.g. Ansätze
    - mehrzeiliger String
    - kein \ als Escape-Zeichen erforderlich vor "
    - Variablen werden durch ihren Wert substituiert
  - ⇨ Alternativ kann man mit "Templates" arbeiten
    - Die Webseite wird mit Platzhaltern erstellt und eingelesen `$inhalt = file_get_contents("template.html");`
    - anschließend werden die Platzhalter durch Inhalte ersetzt (`str_replace()`)

```
echo <<<EOT
<html lang="de">
<head> ...
  <title>$headline</title>
</head><body>
EOT;
```

# Ein Negativbeispiel: Typischer (!?) prozeduraler PHP Code

```
<?php
echo <<<EOT
  <div class="bestellung"> <h1> Bestellung</h1> <table class="left">
EOT;
$SQLabfrage = "SELECT * FROM pizza ORDER BY pizzanummer";
// Datenbank öffnen und abfragen:
$Connection1 = new MySQLi($host, $user, $pwd, "pizzaservice");
// check connection
if (mysqli_connect_errno()) { printf ("Connect failed: %s\n",
    mysqli_connect_error()); exit(); }
if (!$Connection1->set_charset("utf8"))
   printf ("Charset failed: %s\n", $Connection1->error);
$Recordset = $Connection1->query ($SQLabfrage);
if (!$Recordset){ printf("Query failed: %s\n", $Connection1->error); exit();}

$Record = $Recordset->fetch_assoc();
while ($Record) {
    $Bez = $Record["Bezeichnung"];
          echo ("\t<tr>\n");
          echo ("\t\t <td><img src=\"$Datei\" class=\"pizza\" title=\"Pizza\"
    onclick=\"InsertPizza(&quot;$Bez&quot;)\" ");
          $Record = $Recordset->fetch_assoc();
}
$Recordset->free(); $Connection1->close();
echo "</table>";
```

PHP

HTML

DB

Java Script

Geht das auch ordentlich???

# Bewertung des "Prozeduralen PHP Codes"

| Anforderung | Bewertung | Erfüllt? |
|---|---|---|
| Funktionserfüllung? | so weit ausprobiert... | 😐 |
| Wiederverwendbarkeit? | Copy & Paste | ☹ |
| Testbarkeit? | keine Klasse, kein Unit-Test | ☹ |
| Wartbarkeit? | unübersichtlich, unverständlich | ☹ |
| Sicherheit? | viele Stellen mit kritischen Zugriffen | ☹ |

| Entwurfsprinzipien | Erfüllt? |
|---|---|
| Schwache Kopplung? | ☹ |
| Starker Zusammenhalt? | ☹ |
| Geheimnisprinzip? Kapselung? | ☹ |

Es werden so ziemlich alle Prinzipien verletzt, die man in der Softwaretechnik in vielen Jahren entwickelt hat!

Das ist für einzelne Webseiten akzeptabel, aber für einen größeren oder professionellen Webauftritt undenkbar!

# Webseiten im Vergleich zu klassischen GUIs

- Wie finden wir geeignete Klassen in einer webbasierten Anwendung ?
  - ⇨ erinnern wir uns an ENA bzw. NZSE oder kennen Sie Java Swing?
- Eine HTML-Seite ist vergleichbar mit einem Fenster in Java Swing
  - ⇨ PHP Seiten-Klassen analog zu Java Fenster-Klassen
- Ein HTML-Block in einer Seite ist vergleichbar mit einem Java Panel
  - ⇨ PHP Block-Klassen analog zu Java Panel-Klassen
- Eine Seiten-Klasse hat im Rahmen einer webbasierten Anwendung typischerweise 3 Teilaufgaben (= Methoden) zu erfüllen:
  1. Auswertung der (z.B. von einem Formular) übermittelten Daten und Schreiben dieser Daten in die Datenbank
  2. Abfrage von Daten aus der Datenbank
  3. Generierung und Ausgabe einer HTML-Seite mit Hilfe der Daten aus 2.

processReceivedData()

getViewData()
generateView()

Diese Namen verwenden wir auch in unseren „Seiten-Klassen“

# Designklassen, Analyseklassen, GUI, Webseiten, Datenmodell

- Wie passt das alles zusammen? Ein Versuch
- ENA User Research
  - ⇨ Stereotypen, Anwendungsszenarien
- OOAD Analyse(klassen)
  - ⇨ GUI Analyse:
    Navigationsübersicht, Screendiagramme
  - ⇨ DB Analyse:
    Entity Relationship Diagram
- OOAD Design(klassen)
  - ⇨ ENA Java Swing:
    Screen => Fensterklassen, Panelklassen
  - ⇨ EWA Webtechnologien:
    Screen = Seite => Seitenklassen, Blockklassen
  - ⇨ DB Relationenmodell:
    SQL Data Definition, Objektrelationales Mapping

am Beispiel OBS

wer sind die Benutzer?
was brauchen sie?

Student

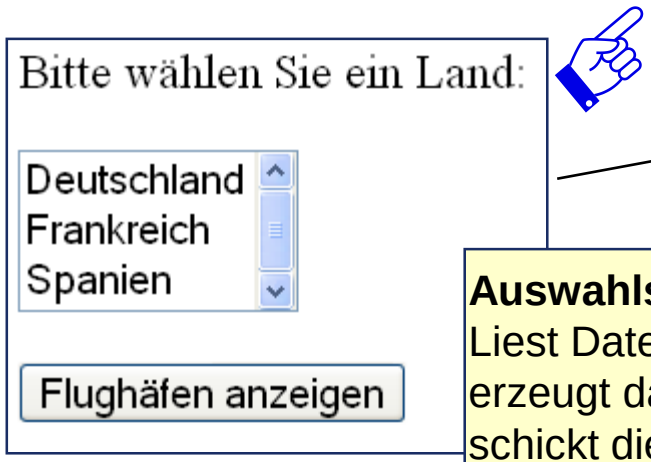
was soll er tun können?

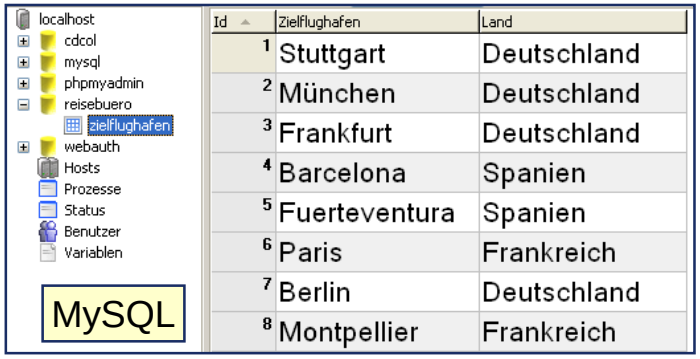
was muss man über ihn speichern?

Belegen, Termine, Noten

Student - belegt - Vorlesung

# Anwendungsbeispiel




**Auswahlseite ("Select"):**
Liest Daten aus der Datenbank,
erzeugt daraus eine Auswahlliste und
schickt die Auswahl an die Ergebnisseite

getViewData()
generateView()

| Zielflughafen | Land |
|---|---|
| Paris | Frankreich |
| Montpellier | Frankreich |
| Nantes | Frankreich |
| Cannes | Frankreich |

**Ergebnisseite ("Result"):**
Empfängt Nutzerauswahl,
liest (gefilterte) Daten aus der Datenbank
und erzeugt daraus die Ergebnistabelle

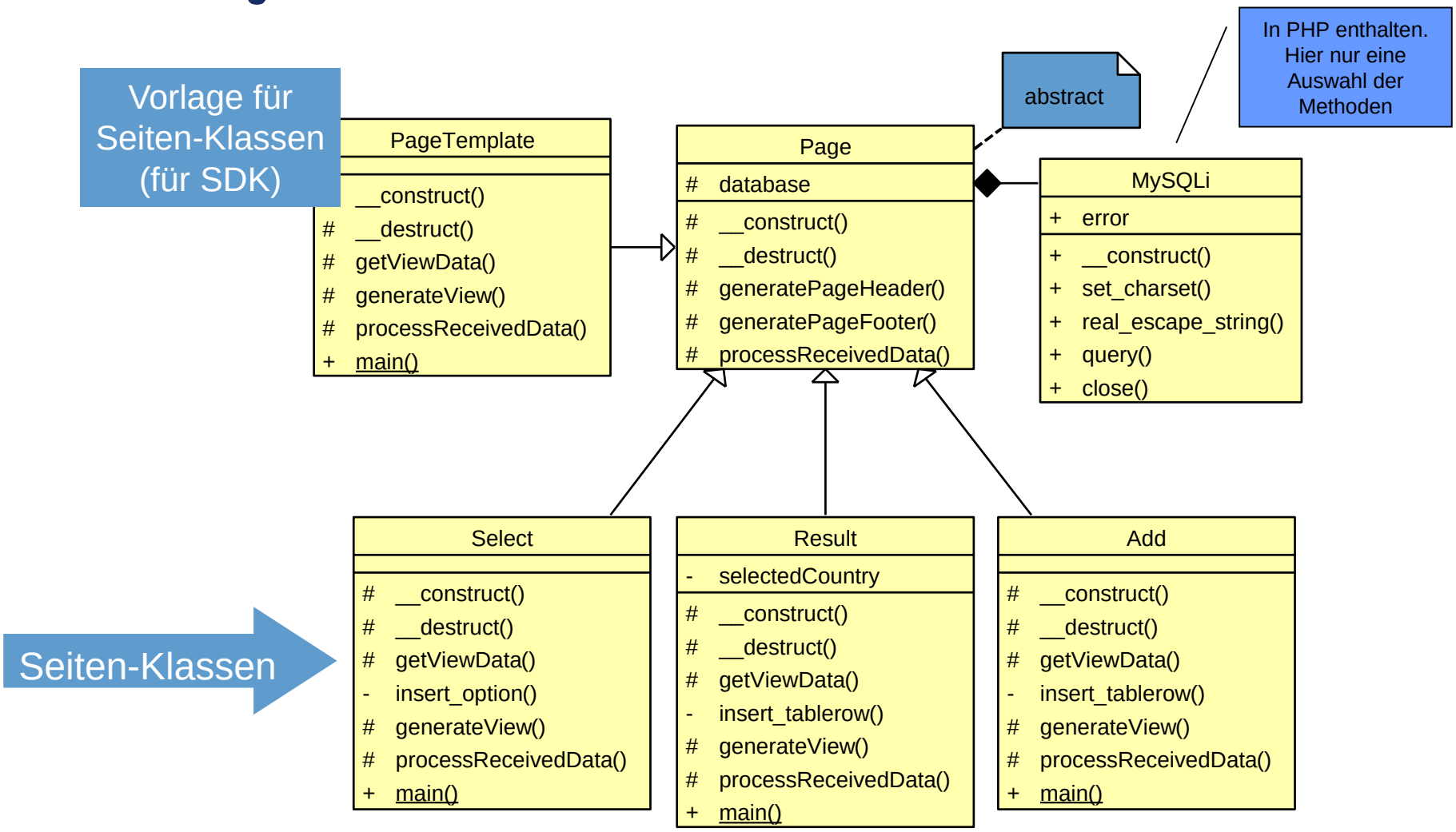
processReceivedData()
getViewData()
generateView()

# PHP-Klassen für HTML-Seiten

- Page (abstrakte Klasse)
  - verwaltet die Datenbankanbindung
  - Beinhaltet Hilfsfunktionen zur Erzeugung einer HTML-Seite
  - Vorverarbeitung in Methode processReceivedData()
- Select
  - Erzeugt eine HTML-Seite mit einem Auswahlfeld und sendet die Auswahl an Result
  - Beinhaltet die Hilfsfunktion insert_option() zum Einfügen von Elementen in eine Select-Box
  - Implementiert die Methoden getViewData(), generateView() und main()
  - processReceivedData() bleibt leer, da keine Daten empfangen werden
- Result (Add analog)
  - Erhält beim Aufruf ausgewählte Daten und erzeugt eine entsprechende HTML-Seite
  - Beinhaltet die Methoden getViewData(), generateView(), main() und processReceivedData()
  - Beinhaltet die Hilfsfunktion insert_tablerow() zum Einfügen von Zeilen in eine Tabelle,
  - Beinhaltet ein Attribut zur Übergabe der Daten zwischen getViewData() und generateView()

main() ist in php kein Schlüsselwort! Die Methode muss explizit beim Laden aufgerufen werden.

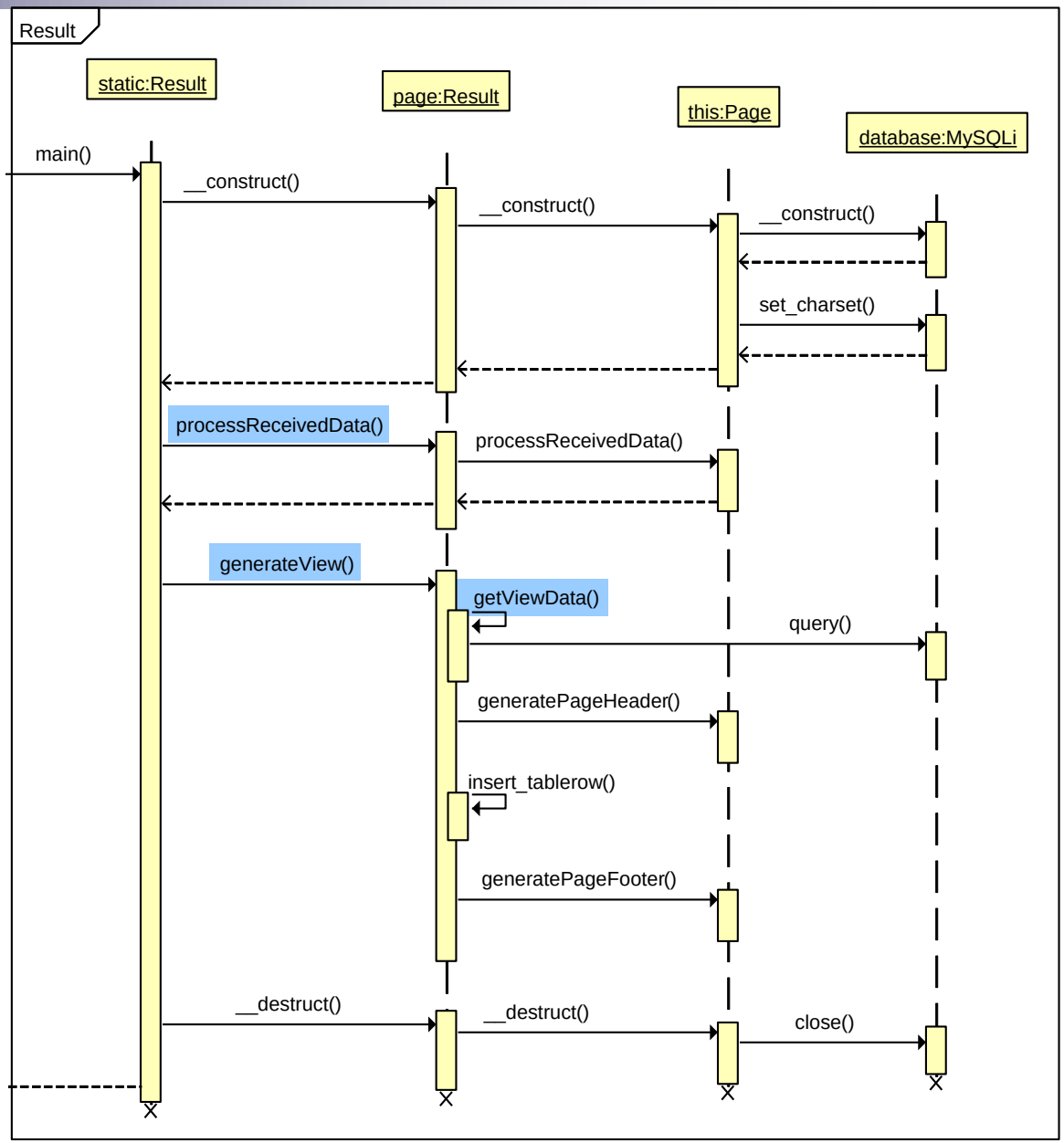
# Klassendiagramm




UML-Diagramme erstellt mit BOUML http://bouml.free.fr/

# Seqenzdiagramm

- Beim Laden der Seitenklasse Result wird die (statische) Methode main() ausgeführt
  - ⇨ das "Objekt" static repräsentiert hier statische Attribute und Methoden der Klasse Result
  - ⇨ diverse Aufrufe gehen an die von der Klasse Page geerbten Methoden



# Gemeinsame Basisklasse 'Page'


```php
<?php
abstract class Page
{
    protected $database = null;          // Referenz auf Datenbankobjekt

    protected function __construct()     // öffnet die Datenbank
    { ... }

    protected function __destruct()      // schließt die Datenbank
    { ... }

    protected function generatePageHeader($title = '')
    { ... }                              // generiert Anfang der HTML-Seite

    protected function generatePageFooter()
    { ... }                              // generiert Ende der HTML-Seite

    protected function processReceivedData()
    { ... }
}
```

# Seiten-Klasse 'Result': Klassenrahmen und Aufruf


```php
<?php
require_once './Page.php';
class Result extends Page
{
   private $selectedCountry;

   protected function __construct() {
      parent::__construct();
      $this->selectedCountry = 'xxx';   // selects nothing
   }

   protected function __destruct() {
      parent::__destruct();
   }
   
}
Result::main(); // nur main() wird direkt ausgeführt
```

hier werden die Methoden von den folgenden Folien eingefügt

# Seiten-Klasse 'Result': main() und processReceivedData()

```
public static function main() {
    try {
        $page = new Result();
        $page->processReceivedData();
        $page->generateView();
    }
    catch (Exception $e) {
        header("Content-type: text/plain; charset=UTF-8");
        echo $e->getMessage();
    }
}

protected function processReceivedData() {
    parent::processReceivedData();
    if (isset($_POST["AuswahlLand"])) {
        $this->selectedCountry = $_POST["AuswahlLand"];
    }
}
```

Hauptprogramm mit Fehlerbehandlung

Verarbeitung der Formulardaten

Die Klasse 'Select' hat das Formular generiert, das hier von 'Result' ausgewertet wird! ☹

# Seiten-Klasse 'Result': getViewData()

```
protected function getViewData() {
                                    // build SQL query from form parameters
    $countries = array();
    $selCountry = $this->database->real_escape_string($this->selectedCountry);
    $sql = "SELECT * FROM zielflughafen WHERE Land=\"$selCountry\"";
    $recordset = $this->database->query ($sql);
    if (!$recordset)
        throw new Exception("Fehler in Abfrage: ".$this->database->error);
                                    // read selected records into result array
    while ($record = $recordset->fetch_assoc()) {
        $airport = $record["Zielflughafen"];
        $country = $record["Land"];
        $countries[$airport] = $country;
    }
    $recordset->free();
    return $countries;
}
```

liest per SQL aus der Datenbank und liefert ein assoziatives Array

# Seiten-Klasse 'Result': generateView()

```
protected function generateView() {
    $countries = $this->getViewData();      // before first HTML output
    $this->generatePageHeader('Ergebnis');
    echo <<<EOT
    <h1>Ausgewählte Flughäfen:</h1>
    <table>
        <tr><th>Zielflughafen</th><th>Land</th></tr>
EOT;
    foreach($countries as $airport => $country) {
        $airport = htmlspecialchars($airport);
        $country = htmlspecialchars($country);
        $this->insert_tablerow("\t\t\t", $airport, $country);
    }
    echo <<<EOT
    </table>
    <p><input type="button" value="Neue Auswahl"
            onclick="window.location.href='Select.php'"/></p>
EOT;
    $this->generatePageFooter();
}
```

generiert HTML aus dem assoziativen Array

# Seiten-Klasse 'Result': insert_tablerow()

```
private function insert_tablerow($indent, $entry1="", $entry2="") {
    echo $indent."<tr>\n";
    echo $indent."\t<td>$entry1</td>\n";
    echo $indent."\t<td>$entry2</td>\n";
    echo $indent."</tr>\n";
}
```

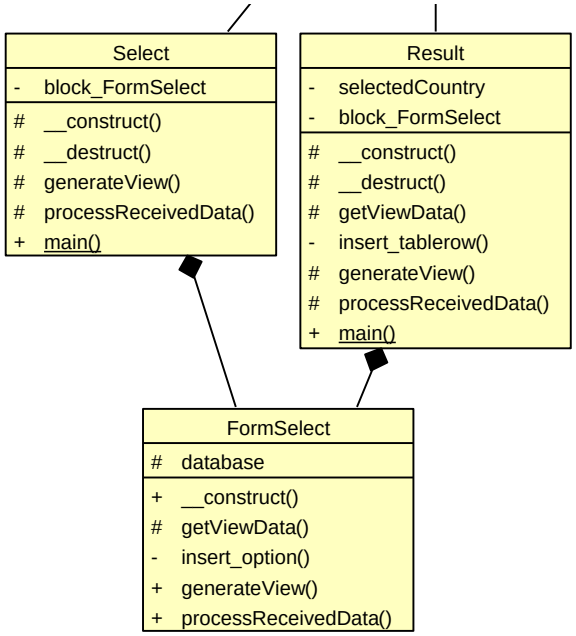
Hilfsfunktion für generateView()

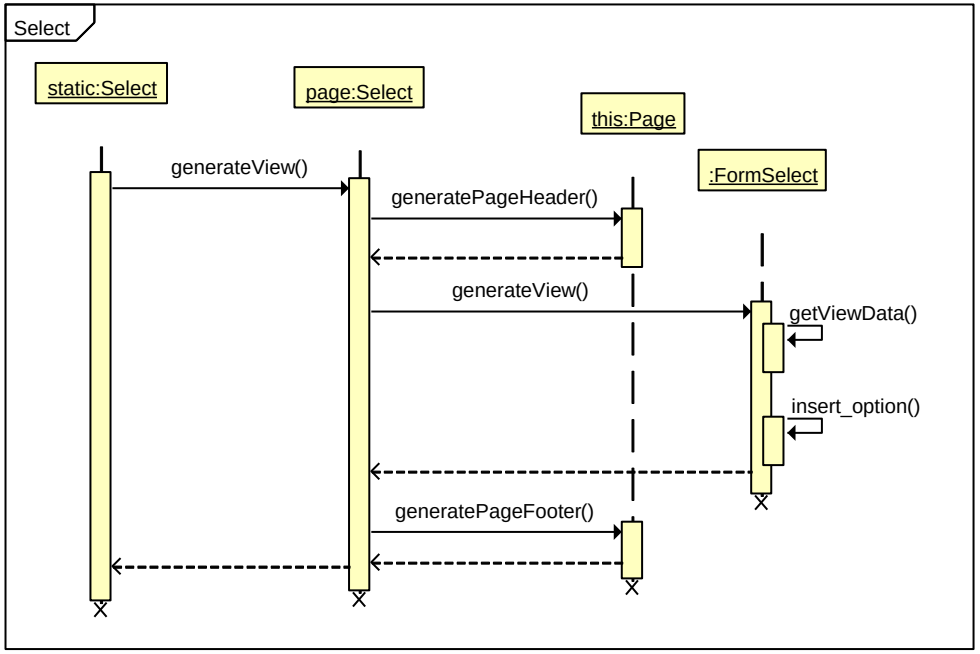
# Block-Klassen

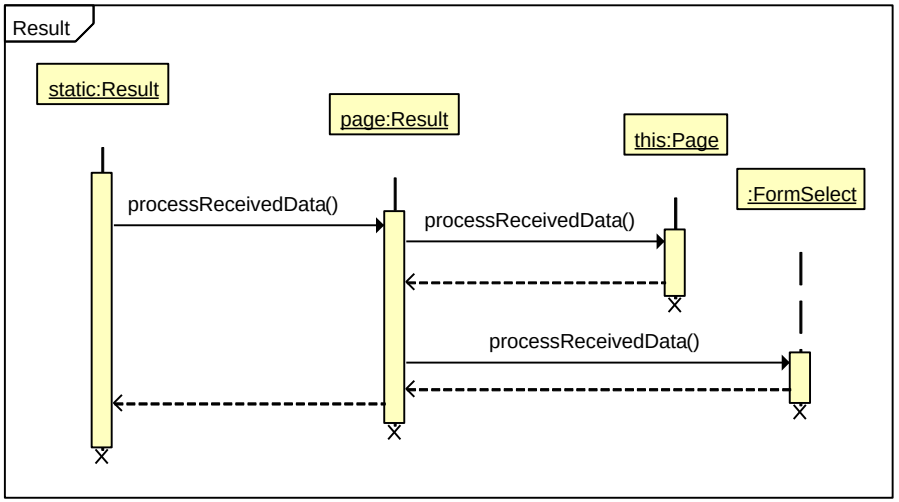
- einige Defizite der bisherigen Seiten-Klassen:
  - ⇨ gleiche bzw. ähnliche Strukturen auf verschiedenen Seiten werden mehrfach implementiert (im Beispiel die Tabelle der Flughäfen)
  - ⇨ bei komplexeren Seiten werden die Methoden getViewData() und generateView() unübersichtlich groß (tritt im trivialen Beispiel nicht auf)
  - ⇨ Generierung und Auswertung von Formularen finden oft auf verschiedenen Seiten statt (im Beispiel Select und Result)
- üblicher Lösungsansatz: Funktionalität auf weitere Methoden und/oder Klassen aufteilen
  - ⇨ spezialisiert für GUI-Objekte: Block-Klassen einführen
  - ⇨ Block-Klassen repräsentieren beliebige HTML Block Tags
  - ⇨ besonders nützlich für <form> Tags
    - Generierung und Auswertung des Formulars in derselben Klasse
    - eventuelle Änderungen oder Erweiterungen betreffen nur noch eine Klasse

# Block-Klasse FormSelect

- FormSelect wird Komponente von Select und Result
  - ⇨ Select ruft generateView()
  - ⇨ Result ruft processReceivedData()
- alle Teilaufgaben des Formulars sind nun gekapselt in FormSelect



# FormSelect->generateView() in class Select


```
class Select extends Page
{
    private $block_FormSelect;
    protected function __construct() {
        parent::__construct();
        $this->block_FormSelect = new FormSelect($this->database);
    }
    protected function generateView() {
        $this->generatePageHeader('Auswahl');
        echo <<<HERE
     <p>Bitte wählen Sie ein Land:</p>
HERE;
        $this->block_FormSelect->generateView();
        echo <<<HERE
    <p><input type="button" value="Flughafen einfügen"
        onclick="window.location.href='Add.php'"/></p>
HERE;
        $this->generatePageFooter();
    }
}
```

# FormSelect->processReceivedData() in class Result

```
class Result extends Page
{
    private $selectedCountry;
    private $block_FormSelect;

    protected function __construct() {
        parent::__construct();
        $this->selectedCountry = 'xxx';
        $this->block_FormSelect = new FormSelect($this->database);
    }

    protected function processReceivedData() {
        parent::processReceivedData();
        $this->block_FormSelect->processReceivedData(
            $this->selectedCountry);  // Parameter nach Bedarf
    }
}
```

# Bewertung des Ansatzes "Seiten- und Block-Klassen"

| Anforderung | Bewertung | Erfüllt? |
|---|---|---|
| Funktionserfüllung? | mit Unit-Tests automatisiert testbar | ☺ |
| Wiederverwendbarkeit? | Wiederholung von main() | 😐 |
| Testbarkeit? | Klassen erlauben Unit-Tests | ☺ |
| Wartbarkeit? | nach Einarbeitung ok, aber fehler-anfällig, weil Aufrufe in bestimmter Reihenfolge erfolgen müssen | 😐 |
| Sicherheit? | wenige Stellen mit kritischen Zugriffen | ☺ |

| Entwurfsprinzipien | Erfüllt? |
|---|---|
| Schwache Kopplung? | 😐 |
| Starker Zusammenhalt? | ☺ |
| Geheimnisprinzip? Kapselung? | ☺ |

Es werden schon viele Prinzipien erfüllt,
aber es geht noch besser !

# Verbleibende Defizite

- manche Dinge werden in jeder Seiten-Klasse wiederholt
  - ⇨ Methode main()
  - ⇨ Ausnahmebehandlung try ... catch
  - ⇨ Aufruf von generatePageHeader() und generatePageFooter()

das könnte ein Framework lösen

- jede Seiten- bzw. Block-Klasse greift auf die Datenbank zu
  - ⇨ und wenn man die Datenbank austauschen will oder muss ?
- jede Seiten- bzw. Block-Klasse generiert HTML
  - ⇨ und wenn man andere Clients bedienen will, z.B. mobile Geräte ?

man könnte Daten (model) und deren Darstellung (view) trennen

↳ **Model-View-Controller Framework**

# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3.5 Model-View-Controller Framework

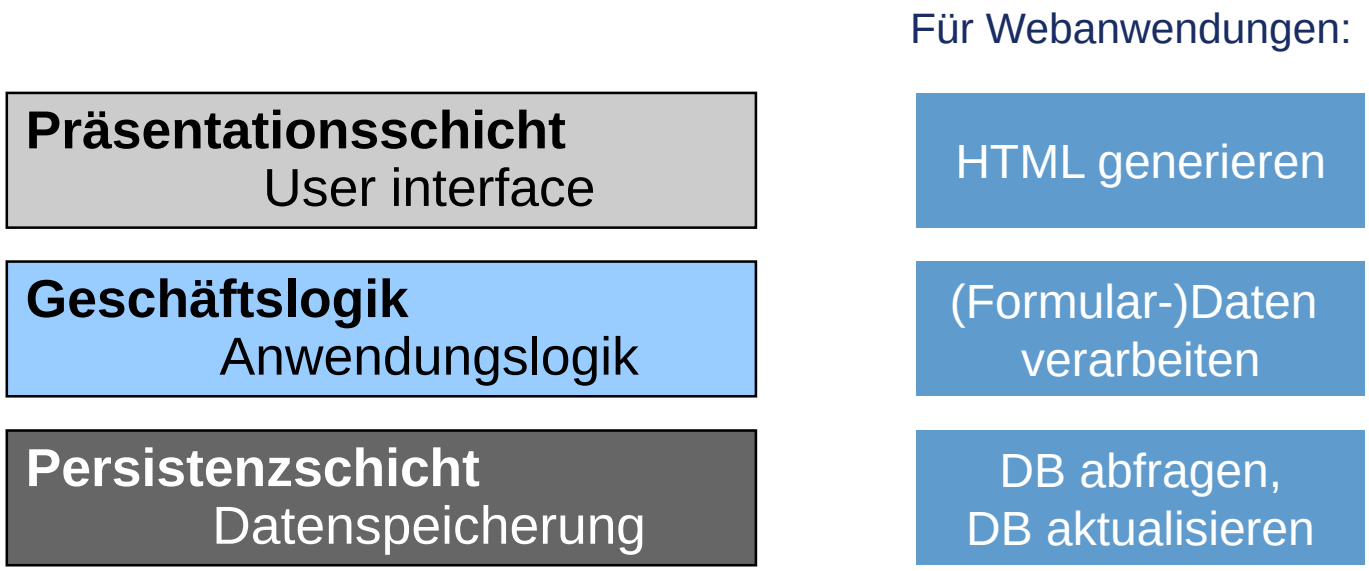
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Funktionsprinzip eines Frameworks

- Ziel: Vorgabe einer klaren Struktur für die Anwendung
  - ⇨ und Lösungen für Standardaufgaben:
    Datenhaltung, Sessionverwaltung, sichere Formulare
- bei Verwendung eines Frameworks schreibt der Entwickler kein "Hauptprogramm" mehr
  - ⇨ das Hauptprogramm steckt im Framework
  - ⇨ zentrale Steuerungslogik
- die Anwendung besteht nur noch aus einzelnen Teilen (i.a. Klassen)
  - ⇨ diese müssen bestimmte Schnittstellenkonventionen erfüllen
  - ⇨ und werden vom Framework zu gegebener Zeit aufgerufen
- Umkehrung der Steuerung (Inversion of control)
  - ⇨ traditionell: die Anwendung ruft Methoden aus Bibliotheken
  - ⇨ mit Framework: das Framework ruft Methoden der Anwendung
  - ⇨ "Hollywood-Prinzip": don't call us, we'll call you

# Drei-Schichten-Architektur

- Die "Standardtechniken" des Software-Engineering sind auch auf die Architektur einer Webanwendung anwendbar
  - ⇨ z.B. starker Zusammenhalt, schwache Kopplung, Kapselung, Auslagern von variablen Teilen usw.
  - ⇨ eine Aufteilung der Anwendung nach Schichten (z.B. Three-Tier Architecture) ist naheliegend



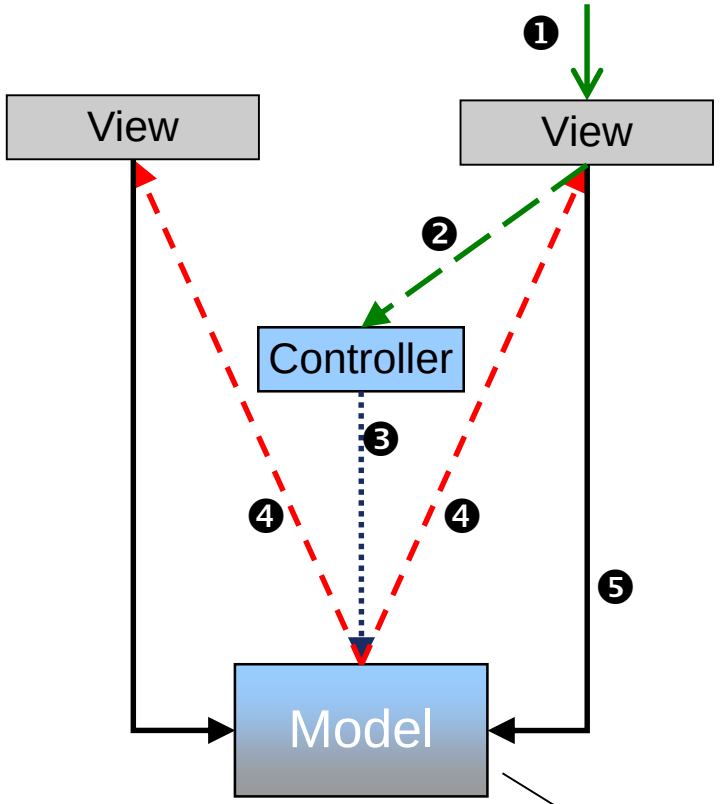
# Model-View-Controller – die Grundidee

- Trenne eine Anwendung in Model, View und Controller
  - ⇨ View(s)
    - kümmern sich "nur" um die Darstellung der Daten
    - reichen Eingaben weiter an den Controller
    - sind leicht austauschbar
  - ⇨ Controller
    - nimmt Eingaben an und stellt fest, was das für das Model bedeutet
  - ⇨ Model
    - enthält Anwendungslogik
    - kapselt Datenbankzugriffe
- Die Teile sind entkoppelt und leicht austauschbar
  - ⇨ durch eine Observer-Schnittstelle wird das Model vom Rest entkoppelt

Diese Aufteilung hat sich schon in vielen Anwendungen bewährt

# Model-View-Controller in lokaler Anwendung

vgl. Java Swing




❶ Ereignis an einem View

❷ Ereignis wird weiter-gegeben an Controller

❸ Controller veranlasst Änderungen im Model

❹ Model benachrichtigt Views über Änderungen

❺ Views holen Daten vom Model

Geschäftslogik wird im Model umgesetzt

Für Webanwendungen:

Client ruft URL auf / sendet Formular

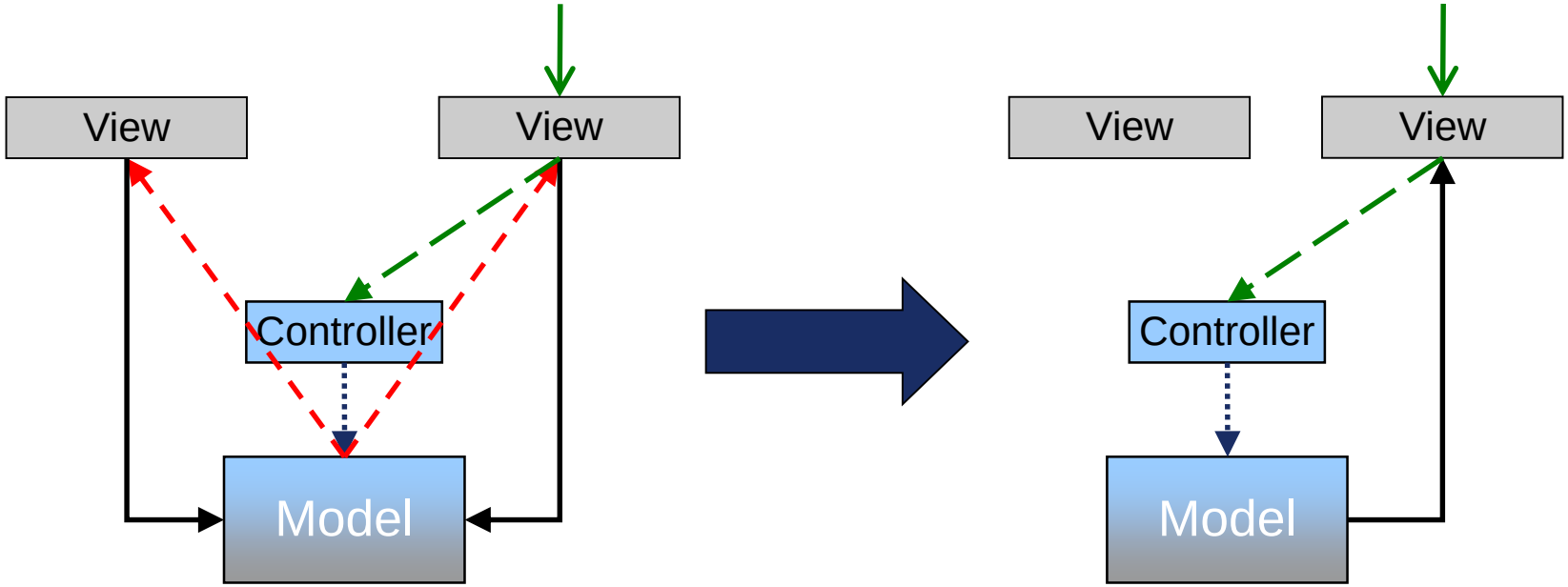
Formulardaten übermittlen

DB aktualisieren

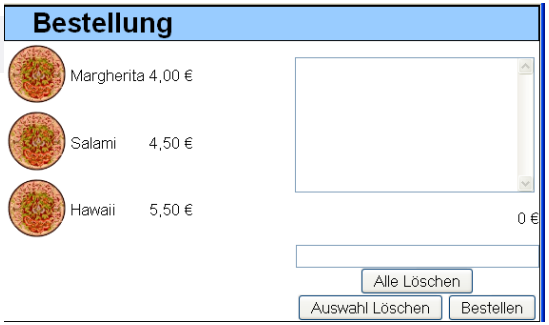
HTML-Antwort generieren; nur der aufrufende Client wird aktualisiert

# Model-View-Controller im Web

- Für Webanwendungen hat MVC einige Besonderheiten
  - ⇨ der View zeigt nicht (wie bei Desktop-Anwendungen) selbst die Seite an, sondern erzeugt lediglich den (HTML-)Code, den ein Browser anzeigt
  - ⇨ weitere Views werden in anderen Browserfenstern gezeigt
  - ⇨ die Benachrichtigung der anderen Views ist über HTTP nicht möglich
    - man hat sich daran gewöhnt, dass eine (bereits geladene) Webseite sich nicht automatisch aktualisiert; dazu muss man die Seite neu anfordern

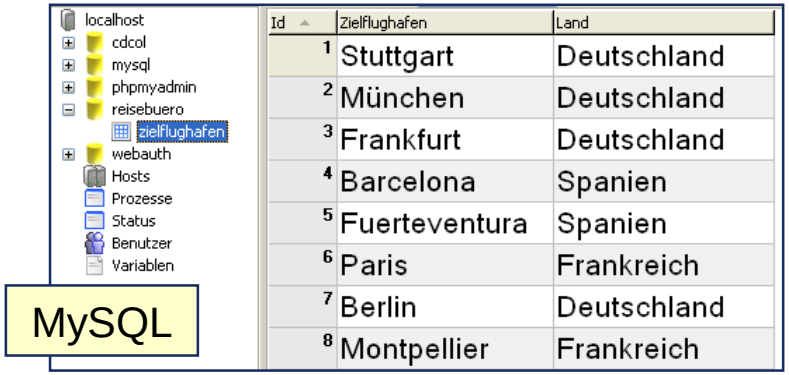


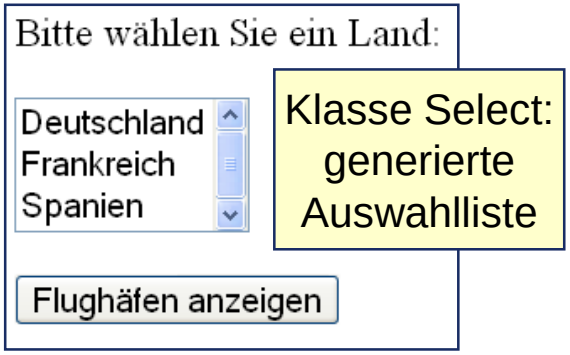
# Model-View-Controller am Beispiel Pizzaservice




- Pizzaservice – Ablauf einer Bestellung
  - ⇨ View(s)
    - es gibt einen View (zur Erzeugung der HTML-Seite) für die Bestellung und je einen View für die Statusanzeige für Fahrer, Kunde und Bäcker
    - das abgeschickte Formular wird an den Controller geschickt
  - ⇨ Controller
    - nimmt Daten an und stellt fest, was diese Daten (derzeit) für das Model bedeuten
    - aktualisiert das Model gemäß der übergebenen Daten
    - die neue Bestellung wird zur Abspeicherung an das Model übergeben
  - ⇨ Model
    - bietet Methoden zum Erledigen der üblichen Aufgaben des Pizzaservices (z.B. Einfügen und Löschen einer Bestellung samt Daten, Ändern des Status, Abfragen von Informationen)
    - speichert übergebene Bestelldaten und liefert Daten an den View

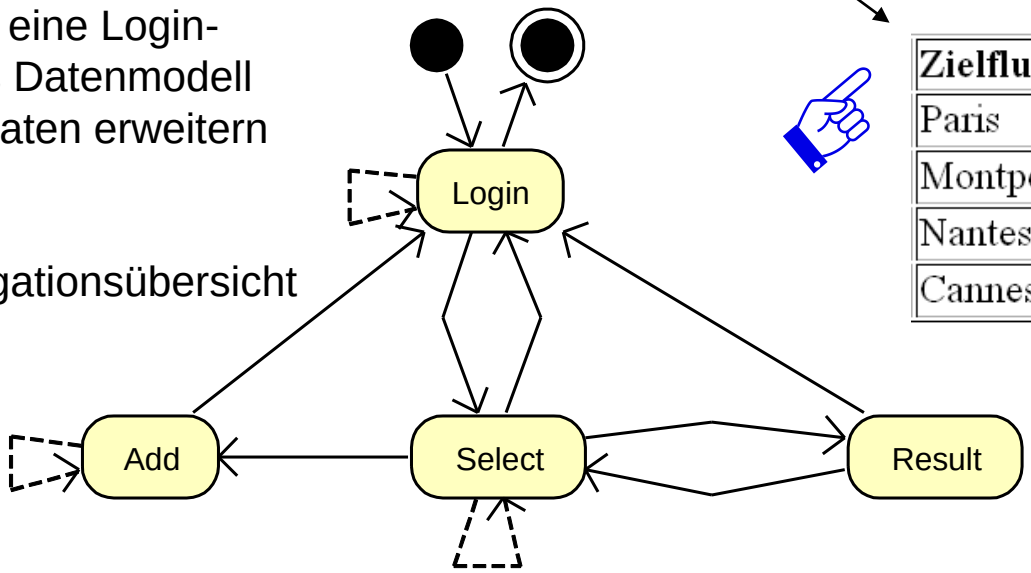
# Das bekannte Anwendungsbeispiel nun realisiert als MVC Framework



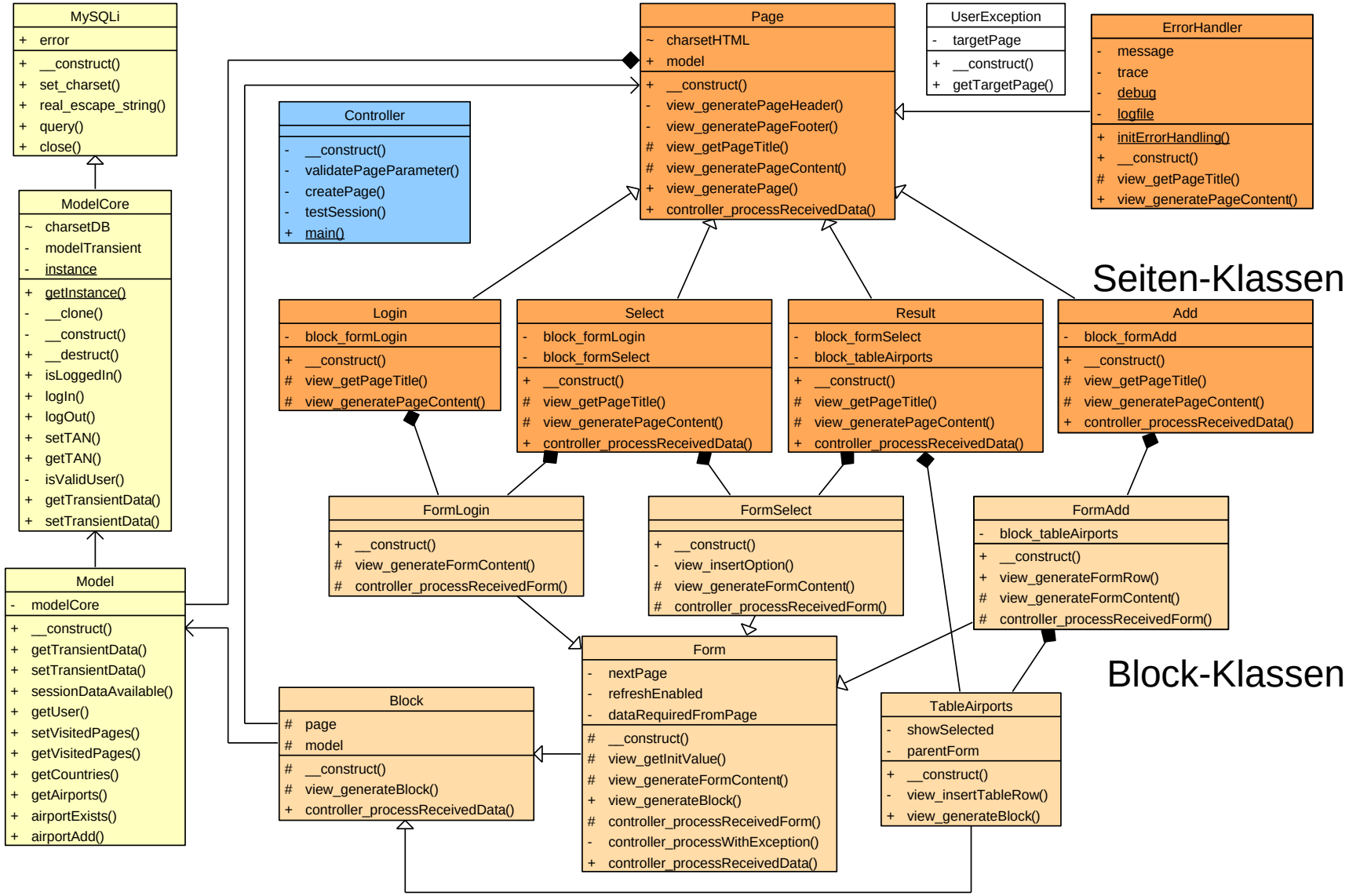

zusätzlich bekommt das Bespiel noch eine Login-Seite um das Datenmodell um Sessiondaten erweitern zu können

Navigationsübersicht

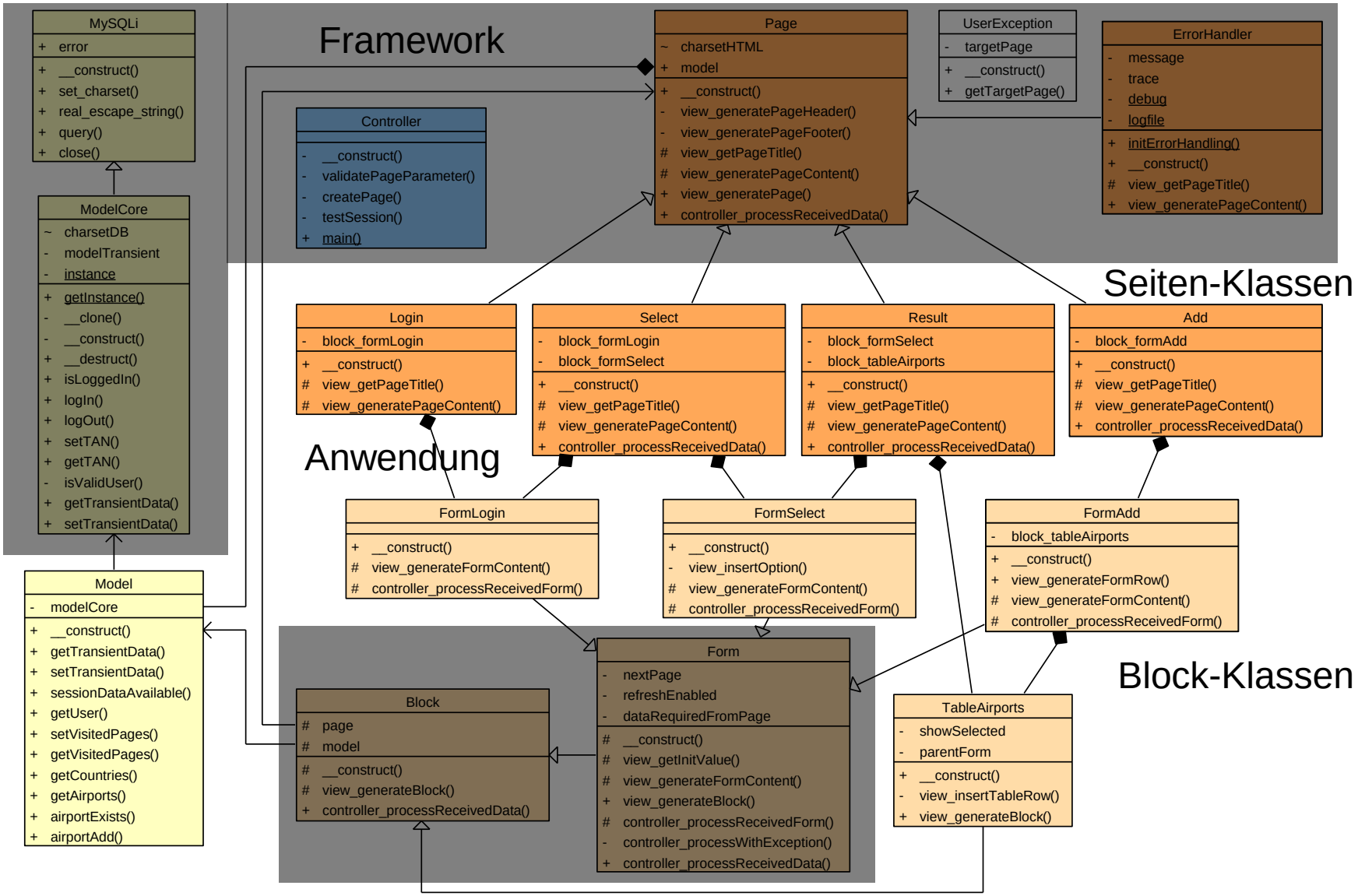





# Klassendiagramm des Anwendungsbeispiels

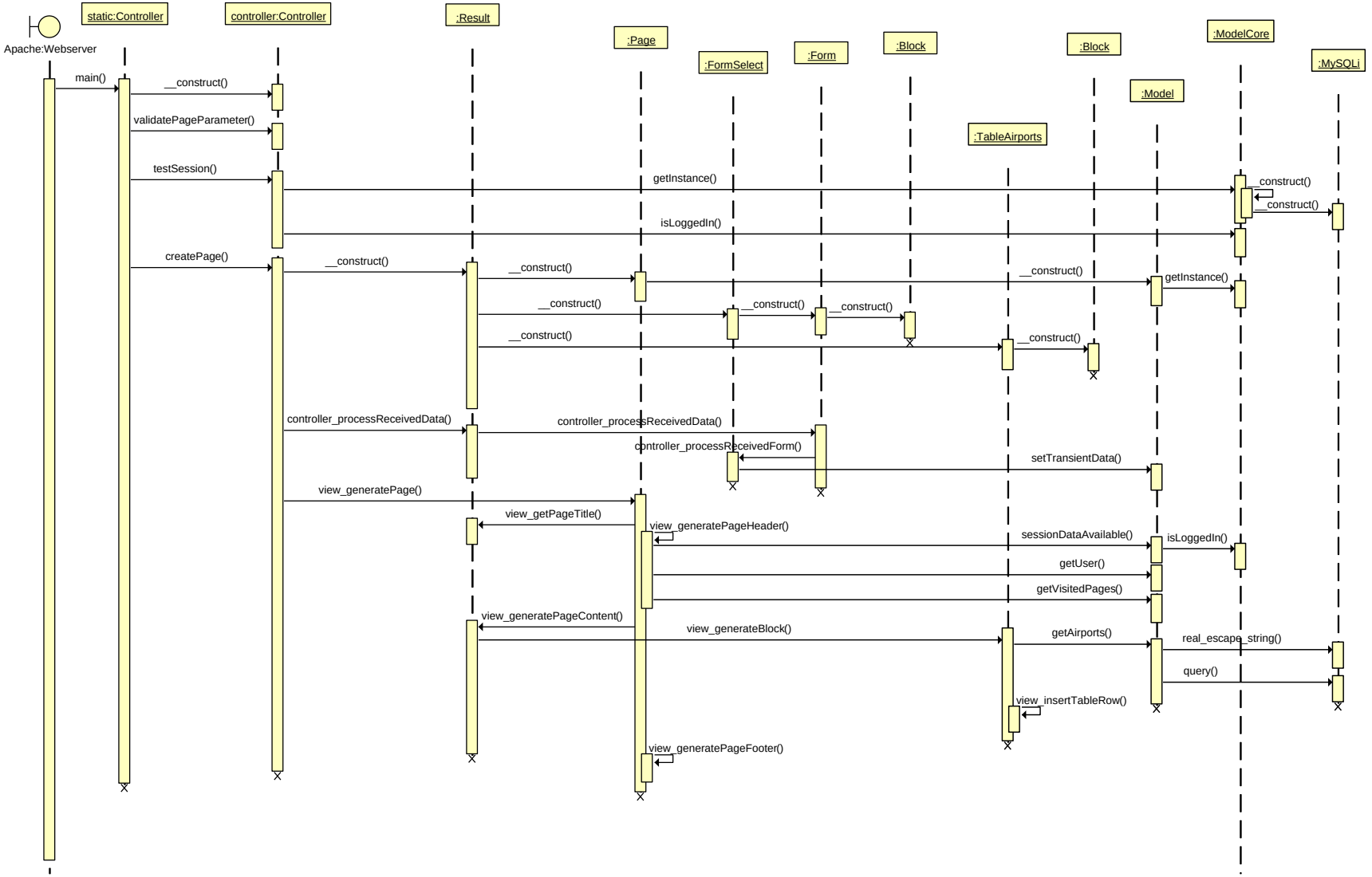


# Aufteilung in Framework und Anwendung



# Sequenzdiagramm (Überblick)

selbst das ist nur ein Ausschnitt ...



# Controller

- zentraler Einstieg in das Skript
  - ⇨ Aufruf im Browser: index.php?page=Add
- kapselt main() und die Fehlerbehandlung
- instanziiert eine Seiten-Klasse abhängig vom Parameter page
- weitere Anteile des Controllers sind verteilt in den Methoden processReceivedData() der Seiten- und Block-Klassen
  - ⇨ ggf. Vorverarbeitung von übermittelten Daten


Controller.php

```
final class Controller {
  private function createPage($pageClassName) {
    require_once "./pages/$pageClassName.php";
    $page = new $pageClassName();
    $page->controller_processReceivedData();
    $page->view_generatePage();
  }
  public static function main($debug = false) {
    try {
      mb_internal_encoding(Page::charsetHTML);
      $controller = new Controller($debug);
      $page = $controller->validatePageParameter();
      if ($controller->sessionIsActive($page))
        $controller->createPage($page);
    }
    catch (Exception $e) {
      // output error message as a HTML page
      $errorHandler = new ErrorHandler($e);
      $errorHandler->view_generatePage();
    }
  }
}

Controller::main();
Controller::main(true);
```

hier noch ohne UserException


index.php
indexTest.php

# Model: Persistenz und Verfügbarkeit

Speicherort

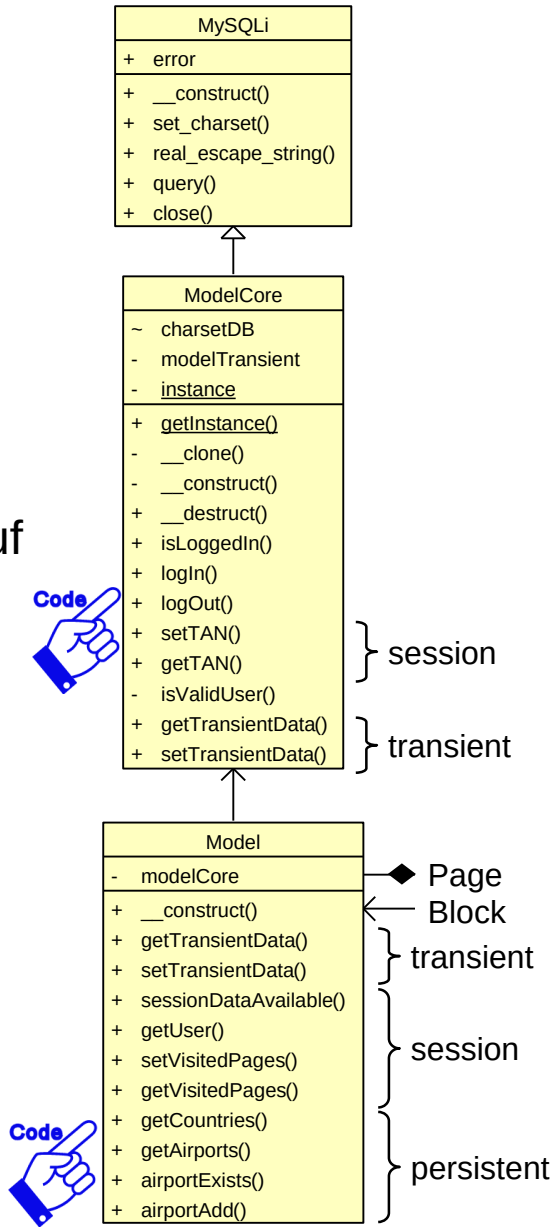
- Man kann das Datenmodell hinsichtlich Persistenz und Verfügbarkeit in drei Bereiche unterteilen:

- Flüchtige lokale Daten (transient data) — Objektattribute
  - ⇨ gehen verloren nach Ausführung des aktuellen Skripts
  - ⇨ privat für den aktuellen Benutzer
  - ⇨ z.B. Suchparameter einer Abfrage
- Sitzungsdaten (session data) — Session Variable
  - ⇨ begrenzte Haltbarkeit für die aktuelle Sitzung
  - ⇨ privat für den aktuellen Benutzer
  - ⇨ z.B. Kundennummer, Inhalt des Warenkorbs
- Dauerhafte globale Daten (persistent data) — Datenbank
  - ⇨ unbegrenzte Haltbarkeit
  - ⇨ öffentlich für alle Sitzungen aller Benutzer
    - Beschränkung ggf. in der Anwendung oder der Datenbank
  - ⇨ z.B. Liste der Kunden, Liste aller Bestellungen

# Model: das Datenmodell im Detail

- ModelCore ist realisiert als Singleton
  - ⇨ stellt sicher, das es systemweit nur eine Instanz gibt
  - ⇨ erbt die Datenbankschnittstelle via MySQLi
  - ⇨ verwaltet Login/Logout und die Session Variablen
  - ⇨ kapselt die transienten Daten
- Model ist eine "normale" Klasse mit einer Referenz auf das ModelCore Objekt

  wg. privatem Konstruktor
  - ⇨ von einem Singleton kann man schlecht erben
  - ⇨ nur Model soll Methoden von MySQLi aufrufen
  - ⇨ wird instanziiert vom Konstruktor der Basisklasse Page
    - alternativ können seitenspezifische Unterklassen von Model implementiert und in Konstruktoren von Seiten-Klassen instanziiert werden
  - ⇨ eine Referenz auf das aktuelle Model wird über die Konstruktoren an die Klasse Block durchgereicht
    - damit ist das Model in allen Seiten- und Block-Klassen verfügbar



# Model: Weiterverarbeitung von Abfrageergebnissen

- Select-Abfragen über MySQLi liefern ein MySQLi_Result Objekt
  - ⇨ Variante 1
    - MySQLi_Result wird in Model::getCountries mittels while-Schleife in ein Array übertragen
    - in FormSelect::view_generateFormContent wird das Array mittels foreach-Schleife in HTML übertragen
    - Vorteil: MySQLi_Result ist in Model gekapselt
    - Nachteil: zwei Schleifen (Mehraufwand für Entwickler und Prozessor)
  - ⇨ Variante 2
    - Model::getAirports liefert ein MySQLi_Result Objekt ab
    - in TableAirports::view_generateBlock wird das MySQLi_Result mittels while-Schleife in HTML übertragen
    - Vorteil: nur eine Schleife
    - Nachteil: View wird abhängig von der verwendeten Datenbank (hier MySQLi)
- Kompromiss
  - ⇨ Kapselung von MySQLi_Result in einer neuen Klasse mit Iterator-Methode und automatischem Aufruf von MySQLi_Result::free() beim letzten Datensatz

# Model: Objektrelationales Mapping

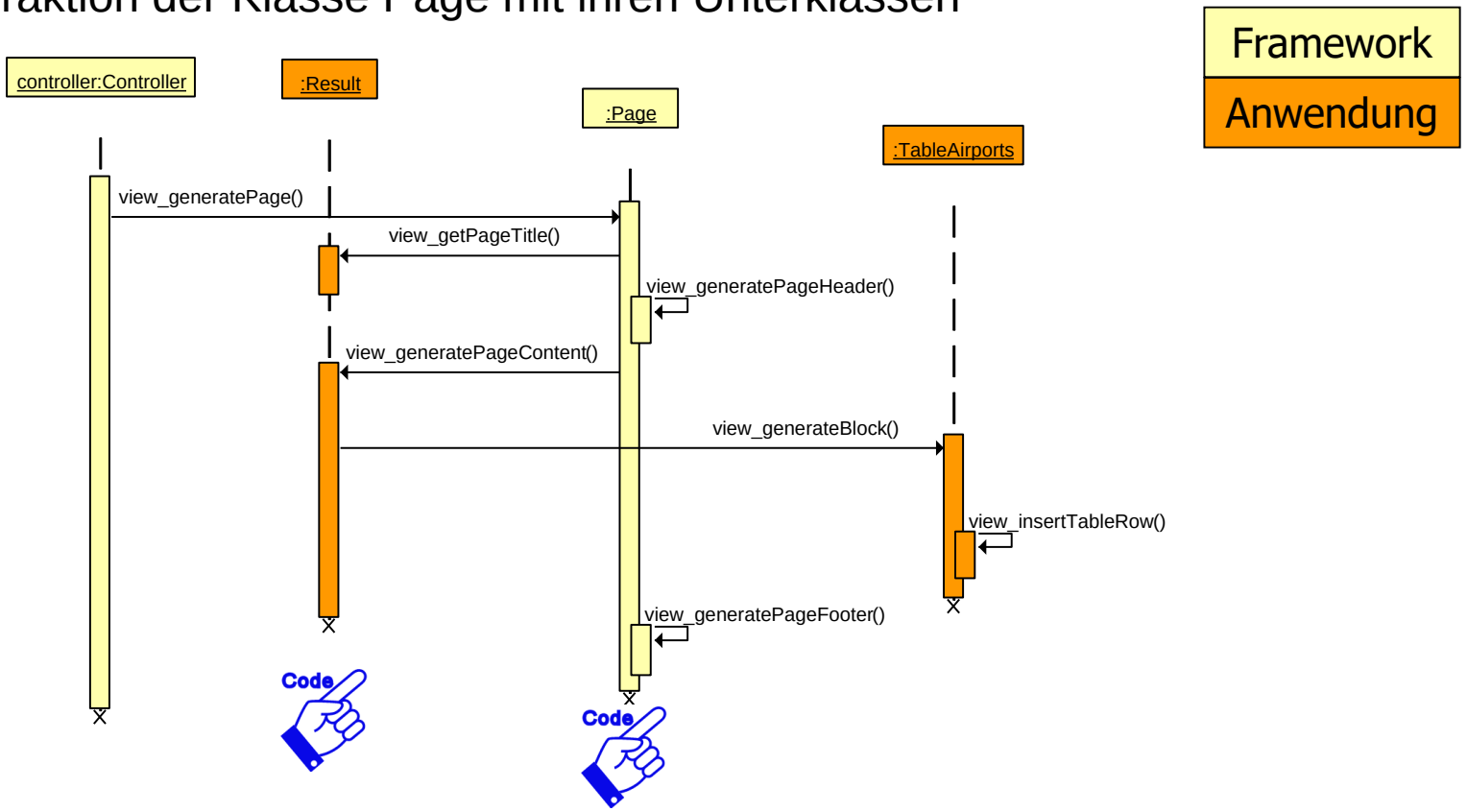
das verfolgen wir nicht weiter

- bisher werden alle Daten einheitlich über "generische" Klassen verwaltet
  - ⇨ MySQLi_Result, Array, Assoziatives Array, mysqli_result::fetch_object
  - ⇨ wenn man relational denken kann, genügt das meistens auch
- die unterschiedliche Semantik der Daten kann man aber auch besser in die objektorientierte Welt abbilden
  - ⇨ pro Datenbank-Tabelle (bzw. Select-Abfrage / Database View)
    - eine Klasse für die Datensätze (ein Datensatz wird ein Objekt dieser Klasse)
    - eine Containerklasse für die Tabelle (speichert mehrere Datensatzobjekte)
  - ⇨ die Attribute der Datensatzklasse entsprechen den Spalten der Tabelle
    - Fremdschlüssel werden zu Objektreferenzen
  - ⇨ die Methoden sind immer die gleichen
    - für die Datensatzklasse: new, get, set, ...
    - für die Containerklasse: insert, delete, iterate, ...
- "Objektrelationales Mapping" oder ORM
  - ⇨ diese Klassen werden automatisch aus einer Beschreibung des Datenmodells generiert (z.B. XML)
  - ⇨ dann muss "nur noch" die Umsetzung der Geschäftslogik im Model manuell implementiert werden

dieses Thema wird vertieft im Wahlpflichtmodul 30.2520 Java EE Datenbankanwendungs-entwicklung

# View: aufgeteilt auf Seiten-Klassen und Block-Klassen

- Der View wird implementiert mittels Seiten- und Block-Klassen
  - ⇨ wie im bisherigen Ansatz
- Das Framework-Prinzip "Inversion of Control" findet sich in der Interaktion der Klasse Page mit ihren Unterklassen



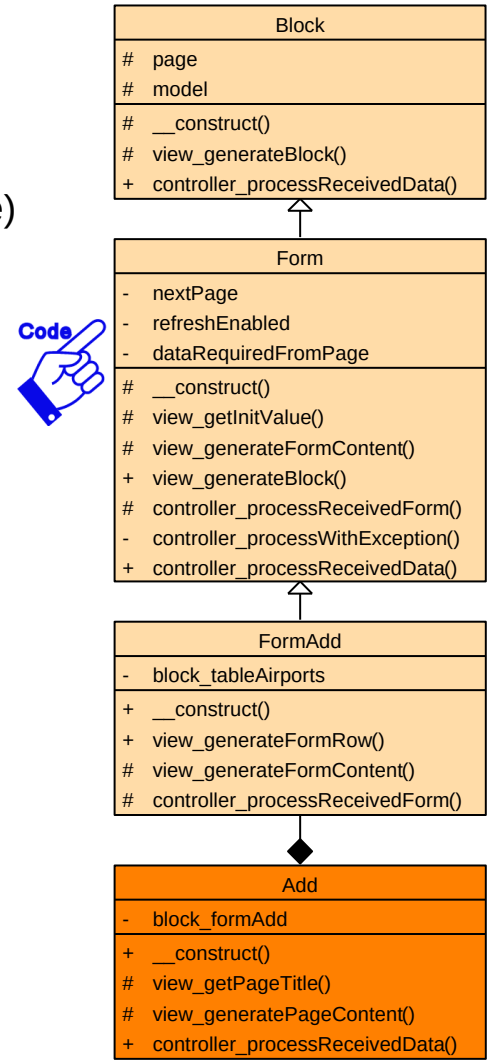
# Hochschule Darmstadt
# Fachbereich Informatik

## 3.3.6 Standardprobleme und Lösungen mit einem Framework

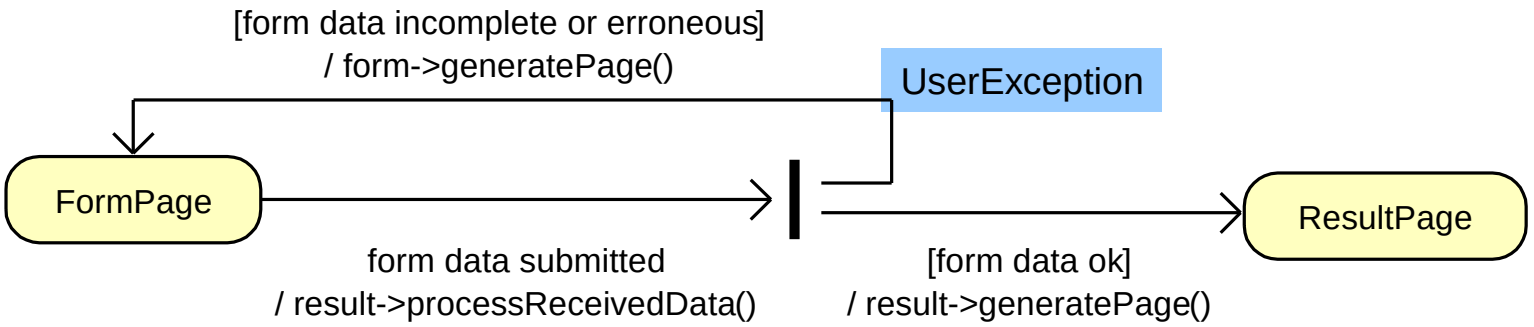
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# View-Controller: Formular sichern gegen wiederholte Verarbeitung

- Basisklasse Form generiert das <form>-Tag
  - ⇨ mit <input type="hidden" name="TAN">
  - ⇨ mit <input type="hidden" name="formPage"> (⇨ nächste Folie)
- refreshEnabled wählt GET bzw. POST
  - ⇨ false / POST: abgesichert mittels TAN gegen Reload der Seite und Zurück-Button des Browsers
  - ⇨ true / GET: Benutzer kann Lesezeichen auf Suchergebnis setzen; keine TAN
- wiederholte Auswertung wird verhindert durch TAN
  - ⇨ Form::view_generateBlock erzeugt eine TAN als Zufallszahl
    - speichert die TAN als Session Variable im Server
    - schreibt die TAN verborgen ins Formular (⇨ Browser)
  - ⇨ Form::controller_processReceivedData vergleicht die vom Browser übermittelte und die gespeicherte TAN
  - ⇨ FormSelect::controller_processReceivedForm wird nur aufgerufen, wenn beide TANs übereinstimmen
  - ⇨ alle abgeleiteten Form-Klassen erben dieses Verhalten



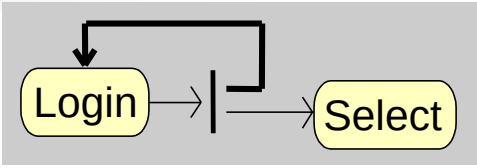
# View-Controller: Formular sichern gegen Eingabefehler (1)

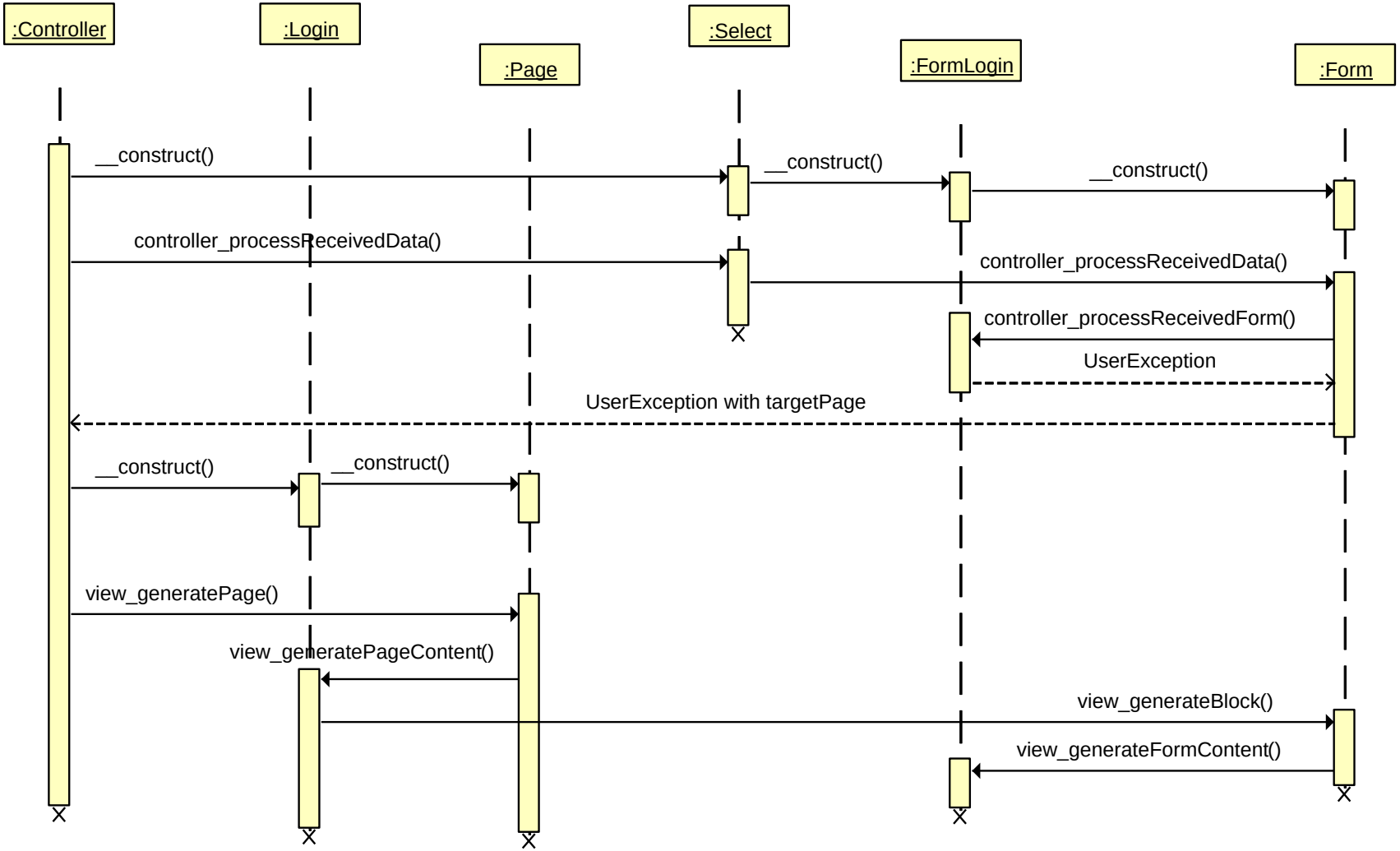



- Ziel: bei unvollständiger oder fehlerhafter Eingabe soll das Formular erneut gezeigt werden, inkl. bereits eingegebener Daten
  - Form::view_generateBlock verbirgt den Namen der FormPage im Formular
- der Browser sendet das ausgefüllte Formular immer an die ResultPage
  - Form::controller_processReceivedData sichert die übermittelten Daten transient im Model
  - FormXxx::controller_processReceivedForm prüft und verarbeitet die übermittelten Daten und wirft im Fehlerfall eine UserException
- ohne UserException:
  - Controller::createPage ruft auf ResultPage::view_generatePage
- mit UserException:
  - Form::controller_processWithException fügt zur UserException den Namen der FormPage hinzu
  - Controller::createPage fängt die UserException ein, erzeugt erneut eine FormPage und ruft auf FormPage::view_generatePage (errmsg)
  - FormXxx::view_generateFormContent initialisiert seine Formularelemente aus dem Model per Form::view_getInitValue

*nur dies hat die Anwendung zu tun; den Rest macht das Framework*

# View-Controller: Formular sichern gegen Eingabefehler (2)

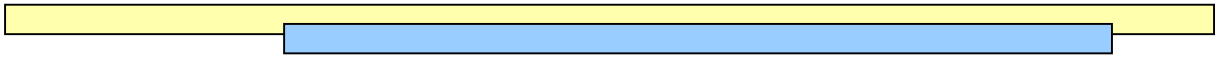


# View-Controller: Login und Sessionverwaltung

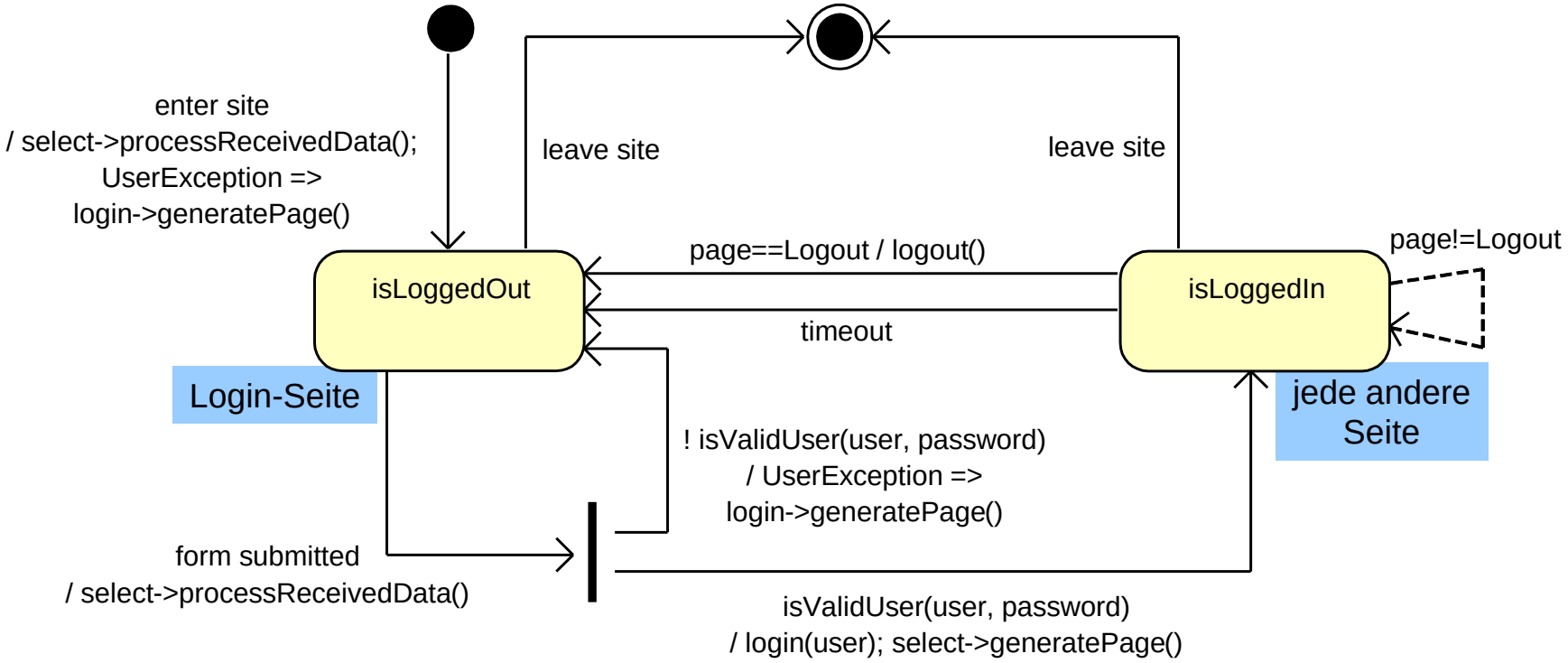
siehe Abschnitt „Sessionverwaltung“

- Man muss unterscheiden zwischen Session und Login/Logout
  - ⇨ Session: eine Folge von Skript-Aufrufen mit eigenen persistenten Daten



  - ⇨ Login ... Logout: ein Benutzer ist authentifiziert; das ist typischerweise ein Teilabschnitt einer Session
- ModelCore::__construct startet bzw. restauriert die Session mit Hilfe der PHP-Funktion session_start()
- innerhalb der Session kann man sich einloggen
  - ⇨ unbekannter Benutzername oder falsches Passwort führen zu einer UserException im Login-Formular
  - ⇨ wenn sich ein Benutzer per Login authentifiziert hat, wird sein Benutzername in $_SESSION['user'] gespeichert
    - die SessionID wird zur Sicherheit ausgetauscht
  - ⇨ die Existenz von $_SESSION['user'] zeigt an, dass ein Benutzer eingeloggt ist
  - ⇨ beim Logout wird $_SESSION['user'] gelöscht
    - die Session selbst bleibt aktiv für das nächste Login
    - das TAN-Verfahren in Klasse Form sichert gegen Reload von Accountdaten aus History
- solange kein Benutzer eingeloggt ist, wird nur die Login-Seite gezeigt

# View-Controller: Login ... Logout als Zustandsdiagramm



# Fehlerbehandlung: Initialisierung

Statische Elemente der Klasse:
- Auswahl Debug / Release
- Fehlerprüfungen aktivieren
- Umlenken in Log-Datei
- PHP Fehler in Exceptions umwandeln


```php
class UserException extends Exception {  // user vs. system
  public function __construct($message, $targetPage = "")
}                       // $targetPage may point to prior page

final class ErrorHandler extends Page {
  private static $debug = true;
```


```php
  private static $logfile = "./framework/log.txt";
  public static function initErrorHandling($debug) {
    self::$debug = $debug;

    // activate all available error messages and redirect them to log file:
    ini_set("error_reporting", E_ALL);            // report anything
    set_error_handler("error_handler");           // register to convert errors to Exceptions
    if (!self::$debug) {                          // but in production environment
      ini_set("display_errors", 0);               // do not show messages directly
      ini_set("log_errors", 1);                   // but log them
      ini_set("error_log", self::$logfile);       // into this file
    }
  }
}
function error_handler($errno, $errstr, $errfile, $errline ) {        // global function
  // convert old PHP errors to modern ErrorExceptions
  // this PHP callback function is registered by ErrorHandler::initErrorHandling
  throw new ErrorException($errstr, 0, $errno, $errfile, $errline);
}
ErrorHandler::initErrorHandling($debug);          // called in Controller::__construct
```

# Fehlerbehandlung: Meldung anzeigen

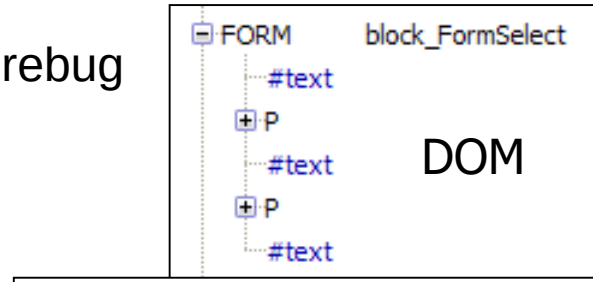
Klasse erbt von Page
- zeigt Meldung und verfolgt die Aufrufe zurück
- schreibt in Log-Datei
- volle Anzeige im Debug; sonst Details nur in Log
- Ausgabe von validem HTML

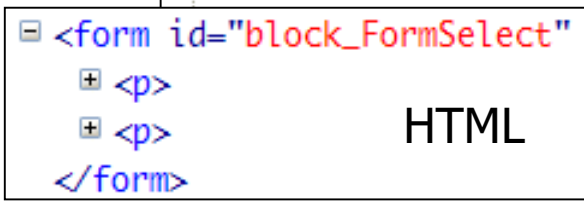
```
final class ErrorHandler extends Page {
  private $message = "";
  private $trace   = "";

  public function __construct(Exception $e) {
    parent::__construct(false);
    $this->message = $e->getMessage();
    if (!$e instanceof UserException) {        // this is a system error: show trace
      $this->trace = str_replace("\n", "\r\n", $e->getTraceAsString());
      if (!self::$debug) {       // write message to log file only; hide it from user
        file_put_contents(self::$logfile,
            "\r\n$this->message\r\n$this->trace\r\n\r\n", FILE_APPEND);
        $this->message = "Es ist ein Fehler aufgetreten.";
        $this->trace = "";
      }
    }
  }

  public function view_generatePageContent() {
    echo "<h1>Fehler</h1>\n";
    echo "<p>$this->message</p>\n";
    if ($this->trace)
      echo "<pre>$this->trace</pre>\n";
  }
}
```

Eine anwendungsorientierte Log-Datei ist extrem wichtig, damit der Betreiber der Seite eventuell auftretende Probleme überhaupt mitbekommt

# Wartung: Finde die richtige Klasse

- Beginne am Graphical User Interface – an der Webseite
  - ⇨ identifiziere den Ort des Problems bzw. die zu erweiternde Stelle
- Rufe die Seite im Browser auf
  - ⇨ finde die Seiten-Klasse über die URL
    http://www.xyz.de/index.php?page=Result
- inspiziere den Block mittels rechtem Mausklick
  - ⇨ verwende Mozilla Add-ons DOM Inspector oder Firebug
  - ⇨ finde die Block-Klasse über die HTML id:
    <form id="block_FormSelect" ...




- das erfordert strenge Namenskonventionen:
  - ⇨ page-Parameter = Klassenname = Dateiname
  - ⇨ block id = Klassenname = Dateiname

# Typische Funktionen von Web-Frameworks

- Vorgabe der Systemarchitektur
  - ⇨ meist Model-View-Controller
  - ⇨ ereignisorientierte Programmierung
- Anbindung von Datenbanken
  - ⇨ ORM Object-Relational Mapping
  - ⇨ Datenbankabstraktionsschicht
- Entwicklerkomfort
  - ⇨ Authentifizierung der Benutzer
  - ⇨ Validierung von Eingaben
  - ⇨ Generierung von HTML-Seiten mittels Schablonen ("Templates")
  - ⇨ Vermeidung wiederholter Neugenerierung von Seiten ("Caching")
  - ⇨ Unterstützung für Ajax
  - ⇨ Integrierte Entwicklungsumgebung

Das haben praktisch alle Web-Frameworks gemeinsam

Hier unterscheiden sich die Frameworks wirklich

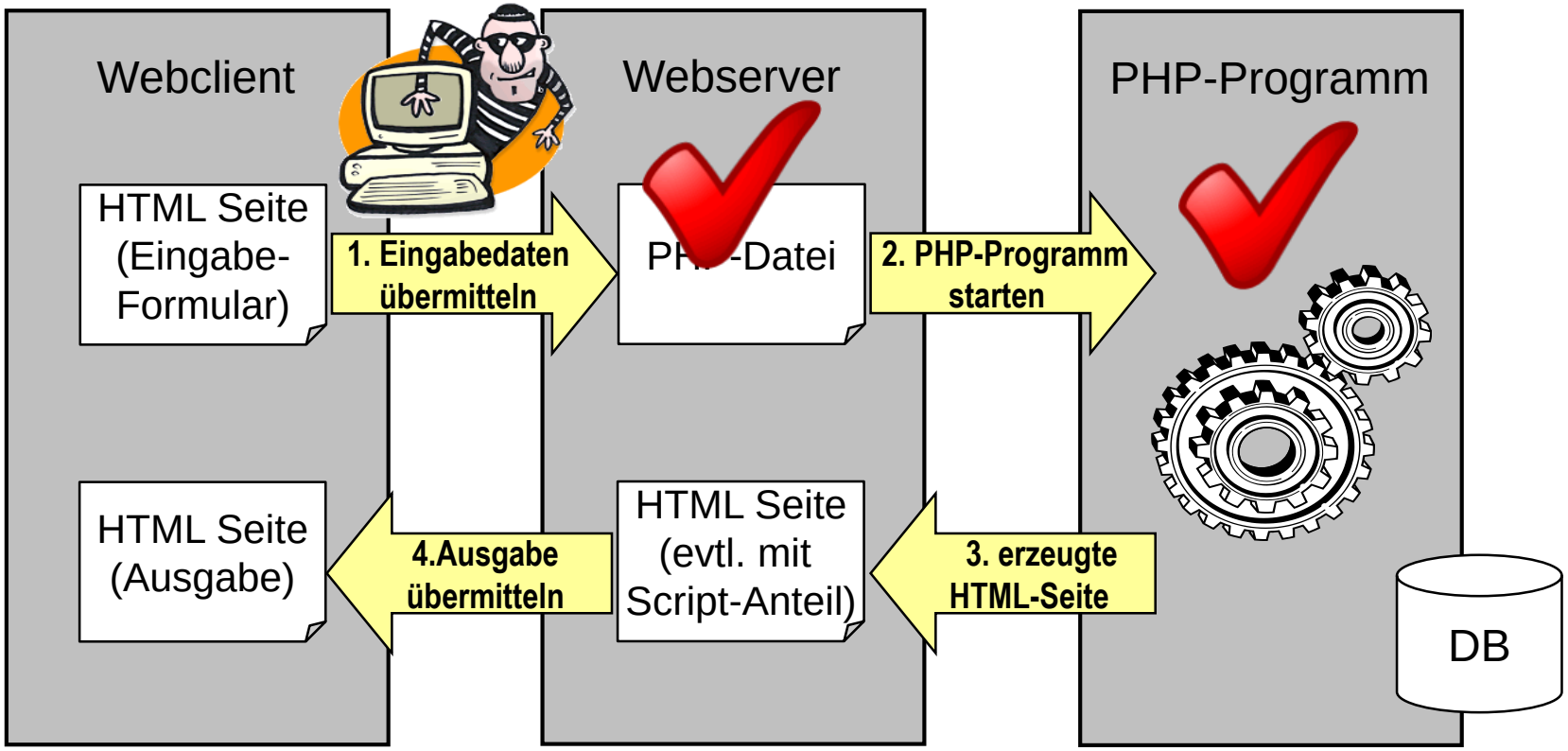
# Freie und kommerzielle Frameworks

- Kleine Auswahl - ohne Anspruch auf Vollständigkeit:
  - ⇨ Symfony
    - orientiert an Rails, reichhaltige Features
  - ⇨ Zend Framework
    - Komponentenorientiert, objektorientiert, mächtig, mit vielen Freiheitsgraden
  - ⇨ CakePHP
    - Rails Adaption
  - ⇨ Prado
    - orientiert an ASP.NET, ereignisorientiert

Referenzmodell: "Konvention vor Konfiguration" vgl. Ruby on Rails

Welches Framework Sie auch verwenden – wenn Sie MVC und ORM verstanden haben, fällt Ihnen die Einarbeitung leichter

# Übersicht




- HTML
- CSS
- ECMA-Script
- DOM
- AJAX

- HTTP

- Server-Konfiguration

- CGI
- PHP
- MySQLi
- Seitenklassen
- Frameworks

# Zusammenfassung: Zeichenkodierung – wo überall festlegen?

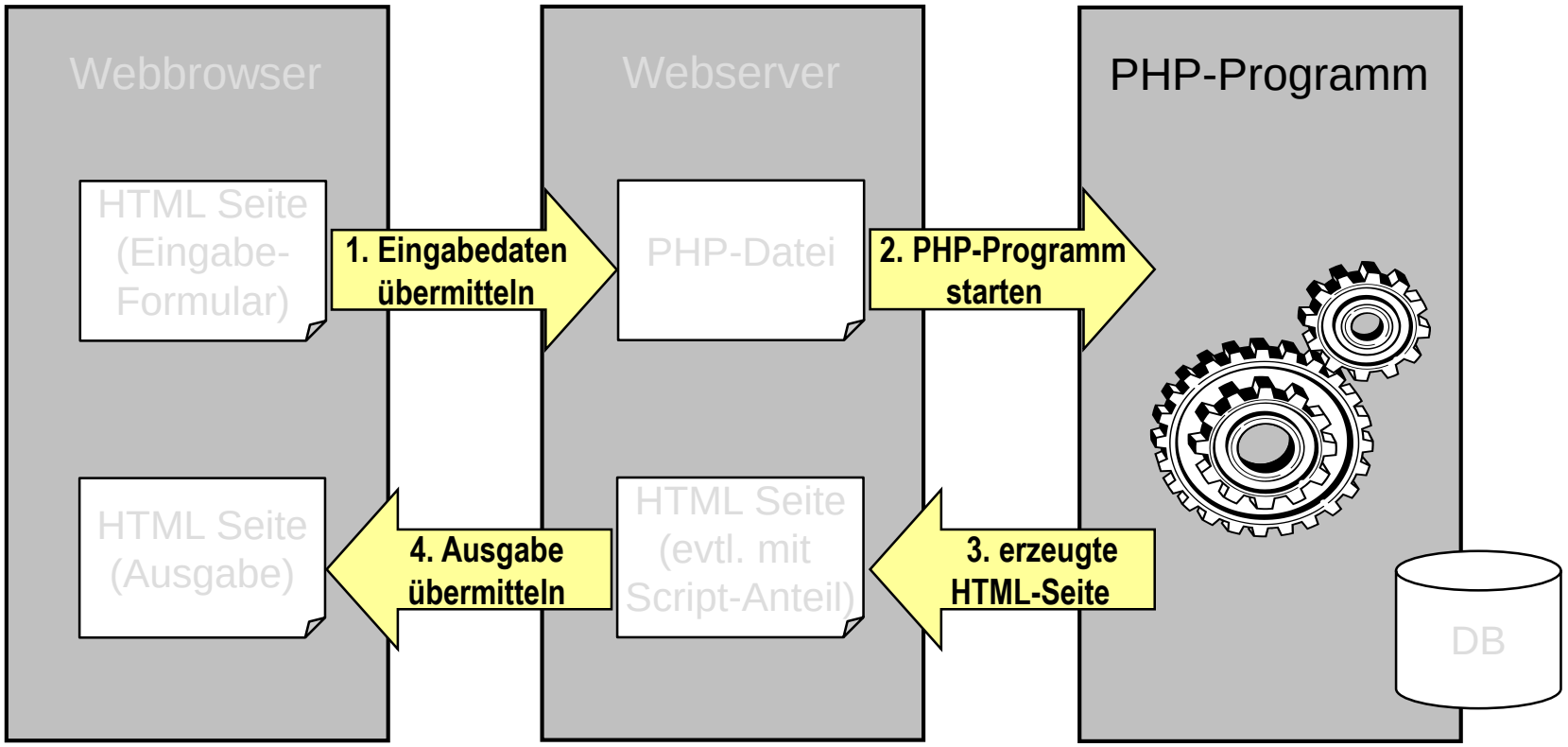
- Dateikodierung im Editor einstellen auf UTF-8 ohne BOM
- HTTP Header per PHP für die Übermittlung Server → Client
  - ⇨ header("Content-type: text/html; charset=UTF-8");
- HTML (und ggf. XML) Header für die Interpretation durch den Browser
  - ⇨ <meta charset="UTF-8" />
  - ⇨ nur falls XML: <?xml version="1.0" encoding="UTF-8" ?>
- im HTML Formular für die Übermittlung Client → Server
  - ⇨ <form action="..." method="post" accept-charset="UTF-8">
- ECMAScript XMLHttpRequest sendet und erwartet standardmäßig UTF-8
  - ⇨ *keine besonderen Maßnahmen erforderlich*
- PHP Strings sind standardmäßig Singlebyte Strings
  - ⇨ Kodierung für Multibyte Strings setzen: mb_internal_encoding("UTF-8");
  - ⇨ spezielle Funktionen für multi byte strings verwenden: mb_...()
- beim Anlegen der Datenbank per SQL
  - ⇨ CREATE DATABASE `db` DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci;
  - ⇨ CREATE TABLE `table` (...) DEFAULT CHARSET=utf8;
- nach Verbinden mit der DB für die Verbindung PHP ←→ MySQL
  - ⇨ $mysqli->set_charset("utf8");

# Hochschule Darmstadt
# Fachbereich Informatik

## 4. Zwischen Webclient und Webserver

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Zwischen Webclient und Webserver




- HTTP
- Sessions

# Hochschule Darmstadt
# Fachbereich Informatik

## 4.1 HTTP

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Begriffe: Client / Server, Pull / Push

- Server bietet Dienstleistung an
  - ⇨ Dienste sind z.B. WWW, FTP, Mail
  - ⇨ nimmt Anfragen entgegen, führt Anweisungen aus, liefert Daten
- Client nimmt Dienstleistung in Anspruch
  - ⇨ initiiert dazu eine Verbindung mit dem Server
  - ⇨ meistens "dichter am Benutzer"
- Pull — dafür ist HTTP gedacht
  - ⇨ Aktivität geht vom Client aus, Server reagiert nur
  - ⇨ "klassischer" Stil der Kommunikation im Internet
- Push — hat sich im Internet bisher nicht durchgesetzt
  - ⇨ Server übermittelt von sich aus (ungefragt) Daten
  - ⇨ Broadcast Systeme, Channels, Abonnements

# HyperText Transfer Protocol (HTTP)

- einfaches Protokoll speziell für die Übertragung von Hypertext-Dokumenten über das Internet
  - ⇨ regelt die Kommunikation zwischen WWW-Client und -Server

- Request / Response Verfahren über eine TCP-Verbindung
  - ⇨ mehrere Nachrichten über dieselbe Verbindung
  - ⇨ einfacher Zugriffsschutz über IP-Adresse oder Passwort

- reines ASCII, Klartext, unverschlüsselt, zeilenorientiert
  - ⇨ Browser ⇨ Server:
    `GETSPhttp://www.xyz.de/datei.htmlSPHTTP/1.1CRLF`
  - ⇨ Server ⇨ Browser:
    `HTTP/1.1SP200SPOKCRLF`
    `Content-Type: text/htmlCRLFCRLF`
    `<!DOCTYPE html><html lang="de">...</html>`

# HTTP-Nachrichten

Details unter
http://www.w3.org/Protocols/rfc2616/rfc2616.html

| Browser ⇨ Server | Server ⇨ Browser |
|---|---|
| **Request-Line:** Methode, URI, Version | **Response-Line:** Statusinformation |
| **General-Header:** (Cache-Control, Proxy-Info, Datum) | |
| **Request-Header:** Anforderungs- und Client-Information | **Response-Header:** Antwort- und Server-Information |
| **Entity-Header:** Information über Entity-Body (Komprimierung, Modifikationsdatum, Sprache, Länge) | |
| **Entity-Body:** Nutzinhalt (HTML-Datei) | |

# Request-Line (Methode, URI, Version)

Browser
⇨
Server

- HTTP Request-Methoden:
  Methoden sind die "Grundfunktionen" für Client-Requests
  - ⇨ in Nachrichten übermittelt, mit Zusatzinformationen ergänzt
- GET: Web-Seite lesen
  - ⇨ auch: wenige Daten an CGI-Prozess übermitteln, vorzugsweise für Abfrage von Daten
  - ⇨ HEAD: liefert dieselben HTTP-Header, aber ohne Web-Seite
- POST: Daten an CGI-Prozess übermitteln
  - ⇨ vorzugsweise für Speichern von Daten
  - ⇨ auch Datei-Upload an CGI-Prozess
- PUT: Web-Seite schreiben / ersetzen
- DELETE: Web-Seite löschen
- OPTIONS (Server-Fähigkeiten), TRACE (loop-back)

GETSP/index.htmlSPHTTP/1.1CRLF

# Response-Line (Version, Status, Reason)

Server ⇨ Browser

Status für Menschen

- HTTP Statusmeldungen
  Antwort vom Server in Response-Line:
  3-stellige Zahl (für Browser) und Klartext (für Benutzer)

  - ⇨ Information 100-101
  - ⇨ Erfolg 200-206
    - Anfrage erfolgreich bearbeitet
  - ⇨ Umleitung 300-307
    - der Client muss anderswo anfragen
  - ⇨ Fehler des Clients 400-417
    - nicht gefunden, Zugriffsverletzung, Authentifikation erforderlich
  - ⇨ Fehler des Servers 500-505
    - interner Fehler, Service nicht verfügbar

**HTTP/1.1**SP**200**SP**OK**CRLF

# HTTP Header (1)



- Header im Allgemeinen...
  - ⇨ dienen dem Austausch von Hilfsinformationen zwischen Client und Server
  - ⇨ bestehen aus Namen und Wert, getrennt durch Doppelpunkt, abgeschlossen mit Zeilenende
    `Content-type: text/html`CRLF

- General Header
  - ⇨ Date
    - liefert den Zeitpunkt der Anforderung oder Antwort
      z.B. Sun, 01 Apr 2000 08:05:37 GMT
  - ⇨ Pragma
    - no-cache: Antwort nicht aus Cache, sondern vom Server holen

`Pragma:no-cache`CRLF

# HTTP Header (2)



- Request Header
  - ⇨ If-Modified-Since
    - bei GET: fordert nur ein aktualisiertes Dokument an

```
If-Modified-Since: Tue, 07 Apr 2004 23:24:25 GMT
```

  - ⇨ Referer
    - von welchem Dokument wurde auf das angeforderte verwiesen ?
  - ⇨ User-Agent
    - Informationen über den Browser (z.B. Mozilla/... für Netscape)

```
User-Agent: Mozilla/4.1
```

# HTTP Header (3)



- Response Header
  - ⇨ Server
    - Information über den Server z.B.: Apache/1.3.17 (Win32)
  - ⇨ WWW-Authenticate
    - verlangt vom Client eine Authentifizierung

# HTTP Header (4)



- Entity Header
  - ⇨ Content-Type
    - Typ des Nutzinhalts, z.B. Content-Type: text/html
  - ⇨ Content-Length
    - Länge des Inhalts in Bytes
  - ⇨ Content-Encoding
    - bei komprimierten Dokumenten, z.B. gzip
  - ⇨ Last-Modified
    - letzte Änderung des übertragenen Inhalts für Caching

# Beispiel

Browser ⇨ Server

(Request)

```
GET /index.html HTTP/1.1
Host: www.xyz.de
User-Agent: Mozilla/4.0
Accept: image/gif, image/jpeg, */*
Connection: Keep-Alive
```

Server ⇨ Browser

(Response)

```
HTTP/1.1 200 OK
Date: Thu, 15 Jul 2004 19:20:21 GMT
Server: Apache/1.3.5 (Unix)
Accept-Ranges: bytes
Content-length: 42
Connection: close
Content-type: text/html

<h1>Antwort</h1>
<p>Jetzt kommt's</p>
```

# Fazit

- HTTP ist ein einfaches Frage-Antwort-Protokoll
  - ⇨ zur Übertragung von Daten in einem Netzwerk
  - ⇨ Es wird hauptsächlich im WWW zur Kommunikation zwischen Webserver und Webclient eingesetzt
  - ⇨ HTTP setzt auf einem Transportprotokoll auf (im Normalfall TCP)
- Jede HTTP Aktion ist ein unabhängiger Vorgang
  - ⇨ HTTP ist ein zustandsloses Protokoll; es enthält keinen Bezug auf vorhergehende Aktionen
  - ⇨ aufeinanderfolgende Request/Response-Aktionen sind unabhängig
  - ⇨ einen Bezug zwischen Aktionen muss die Anwendung bei Bedarf herstellen (z.B. SessionIDs für einen Warenkorb)
  - ⇨ Angriffe (z.B. automatische Login-Versuche) sind schwer zu erkennen

Wenn Sie HTTP mal "live" sehen wollen:
Firefox Addon "Live HTTP headers" installieren

# Hochschule Darmstadt
# Fachbereich Informatik

## 4.2 Sessionverwaltung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Beispiel (Prinzip)

Ob BOL tatsächlich diese Seite so erzeugt, ist nicht bekannt

## Beispiel: Warenkorb (bei BOL)

Mit PHP erzeugte HTML-Seite


Eintrag aus einer Datenbank

erste Überprüfung der Eingabe mit ECMA_Script

Layout mit CSS

Aber woher weiß der Webserver welcher Warenkorb zu welchem Kunden gehört?

# Zusammenhängende Webseiten - Beispiel

- Formular-Folge mit mehreren Seiten
  (z.B. Shop, Warenkorb, Bestellung,…)
- Was passiert, wenn die URL parallel in mehreren Browsern geöffnet wird?
- Woher weiß der Webserver / ein CGI-Skript, dass die Aufrufe zusammen gehören?

# Die Problematik

- Jede HTTP Aktion ist ein unabhängiger Vorgang
  - ⇨ HTTP ist ein zustandsloses Protokoll; es enthält keinen Bezug auf vorhergehende Aktionen
  - ⇨ aufeinanderfolgende Request/Response-Aktionen sind unabhängig
  - ⇨ Wie kann man einen Bezug zwischen Aktionen herstellen?

- jeder Aufruf eines CGI-Skripts ist ein neuer Aufruf eines selbständigen Programms
  - ⇨ idealerweise in einem eigenen Prozess
  - ⇨ CGI-Skripten können keine Daten über globale Variable austauschen oder in solchen retten
  - ⇨ nur persistente Speicher sind brauchbar: Dateien / Datenbanken
  - ⇨ Wie kann man Daten zwischen Aufrufen austauschen?
  - ⇨ …der Bezug kann nur über den Aufruf hergestellt werden!

# Grundidee

- Verwende einen Identifier, der zwischen Client, Webserver und CGI-Anwendung ausgetauscht wird und simuliere damit eine feste Verbindung !
  - ⇨ Erzeugung durch den Webserver / die CGI-Anwendung
  - ⇨ Speicherung beim Client oder (temporär) als Parameter in der URL
  - ⇨ Zusätzliche Übertragung der ID bei jedem Aufruf dieser URL
- Sinnvolle Unterstützung
  - ⇨ Webserver:
    - Methoden zum Erstellen, Löschen und Verwalten solcher IDs
    - Methoden zum Verwalten der Daten einer Verbindung
  - ⇨ Webclient:
    - Abspeicherung und Übertragung der IDs

# Was ist eine Session ?

- Eine zeitweise bestehende Verbindung zwischen einem Server und einem Client
- Zusammenhängende Ausführung mehrerer Aktionen, die dieser Session zugeordnet sind
  - ⇨ z.B. Ausfüllen mehrerer Formulare mit jeweils zugehöriger Rückantwort
  - ⇨ involvierte CGI-Skripte müssen auf Sessiondaten zugreifen können (SessionID ⇨ User, User-bezogene Daten)
- Eröffnung durch einen HTTP-Request
  - ⇨ typischerweise "Login"
- Beenden durch einen HTTP-Request
  - ⇨ typischerweise "Logout"
  - ⇨ könnte vom Benutzer vergessen werden
- mehrere Sessions können gleichzeitig offen sein

persistent bis zum Ende der Session; Zugriff beschränkt auf diese Session

Datensicherheit ist eine separate Anforderung

# SessionID


- wird vom Server beim Login generiert und beim Logout gelöscht
  - ⇨ eindeutig — soll verschiedene Benutzer unterscheiden
  - ⇨ zufällig — soll nicht erraten werden können
  - ⇨ kryptisch — verdeckt das Bildungsgesetz
  - ⇨ mit Erstellungs- oder Verfallszeitpunkt — falls ein Benutzer sich nicht abmeldet

- wird zwischen Server und Client hin- und hergereicht
  - ⇨ in HTML-Datei (Formulardaten, href) oder HTTP-Header (Cookie, URL)

in jedem Fall leicht manipulierbar

- wird im Server bei jeder Seite verwendet, um den Benutzer zu identifizieren

# HTTP Cookie

Server speichert Daten clientseitig
(nicht nur für Sessionverwaltung)

"Cookie"

- HTTP Cookies sind (evtl. dauerhaft) vom Webclient gespeicherte Daten
  - ⇨ sind einer bestimmten Website zugeordnet
  - ⇨ können von Client und Server gelesen und geschrieben werden
  - ⇨ Der Webclient sendet die Daten bei Zugriffen auf eine Webseite an den zugehörigen Webserver
- Cookies werden gesetzt
  - ⇨ per Meta-Tag `<meta http-equiv="set-cookie" content= "Keks=Wert;expires=friday, 31-dec-09 23:59:59 gmt;"/>`
  - ⇨ per HTTP-Header (gemäß RFC 2109 / RFC 2965)
    ```
    Set-cookie: Keks=Wert; domain=h-da.de;
      expires=friday, 31-dec-09 23:59:59 gmt;
    ```
  - ⇨ oder auch mit JavaScript über HTMLDocument:
    ```
    document.cookie = "nochnKeks=xy";
    ```

# Cookies unter PHP

natürlich nicht nur für Sessionverwaltung

- Funktionsdeklaration
  - ⇨ `int setcookie (name [, value [, expire [, path [, domain [,secure]]]]])`
- Cookie setzen
  - ⇨ `setcookie ("SessionID", $Wert, time()+3600);`
  - ⇨ verfällt nach 1 Stunde
- Cookie löschen
  - ⇨ `setcookie ("SessionID", "", time()-3600);`
  - ⇨ mit denselben Parametern wie beim Setzen !
  - ⇨ Verfallszeit in der Vergangenheit bewirkt sofortiges Löschen
- Cookie-Wert auslesen
  - ⇨ `if (isset($_COOKIE["SessionID"])) $Wert = $_COOKIE["SessionID"];`

Serverzeit

Per HTTP-Header!
Vor der ersten Ausgabe

# Cookies beschränken und teilen

URL, die das Cookie setzt, z.B.: https://obs.fbi.h-da.de/obs/index.php

- beschränken auf Domäne
  - ⇨ default: Server Name der URL obs.fbi.h-da.de
  - ⇨ erweitern für alle Subdomänen .fbi.h-da.de ⇨ www.fbi.h-da.de etc.
- beschränken auf Pfad (unterhalb der Domäne)
  - ⇨ default: Pfad der setzenden URL /obs/
  - ⇨ alle Pfade der Domäne /
  - ⇨ anderer Pfad /mhb/
- Cookies von Drittanbietern (third-party cookies)
  - ⇨ Beispiel: eine Webseite holt ein Werbebanner von einer anderen Domäne; das Werbebanner kann ein Cookie setzen
  - ⇨ solche Cookies können i.a. im Browser separat gesperrt werden
- Cookie "Sharing" über kooperierende Top Level Domains
  - ⇨ jeder Server setzt sein eigenes Cookie, jedoch mit demselben Wert
  - ⇨ die Server übergeben sich den Wert als Parameter von Redirects

# SessionID mit Cookies übertragen

- Der Webserver übergibt dem Webclient eine SessionID zur Abspeicherung als Cookie
  - ⇨ normalerweise per HTTP-Header
    ```
    Set-cookie: Session=081512345678; domain=h-da.de;
     expires=friday, 31-dec-09 23:59:59 gmt;
    ```
- Der Webclient liefert das Cookie ab sofort bei allen Aufrufen dieser Webseite mit
  - ⇨ CGI kann Umgebungsvariable HTTP_COOKIE abfragen
    ```
    HTTP_COOKIE=Session=081512345678; nochnKeks=xy
    ```
  - ⇨ der Webserver / CGI "weiß" welcher Client zugreift
- Problem: Benutzer kann Cookies abschalten
  - ⇨ Anwendung kann sich also nicht darauf verlassen und sollte – wenn möglich – auch ohne Cookies funktionieren

# SessionID ohne Cookies

nur verfügbar mit
session.use_only_cookies=Off

- optimal bei HTML-Formularen
  - ⇨ Verstecktes Formularelement mit generierter ID
  - ⇨ 
```
<input type="hidden"
  name="SessionID" value="081512345678">
```
  - ⇨ Client schickt dieses per GET sichtbar oder per POST unsichtbar zurück

- bei formular-losen HTML-Dateien über Verweisziel
  - ⇨ Server generiert URL mit angehängtem Parameter
```
<a href="naechsteSeite.htm?SessionID=081512345678">
```
  - ⇨ Client schickt dieses per GET sichtbar zurück
  - ⇨ gefährlich: Besucher der Seite könnte die SessionID versehentlich per Link an andere weitergeben. Der Empfänger kann damit die Session öffnen!

ID Erzeugen mit PHP:
$SID=md5 (uniqid(mt_rand()));

# PHP Sessionverwaltung

php.ini

- generiert und übermittelt SessionID

per Cookie oder URL

- sichert und restauriert persistente Variable

in Datei oder DB

- Session eröffnen (Login) bzw. restaurieren (Folgeseiten)
  - ⇨ am Anfang des PHP-Programms: `session_start();`
- persistente Variable registrieren
  - ⇨ `$_SESSION["Zustand"] = 5;`
  - ⇨ am Programmende werden persistente Variable autom. gesichert
- danach Zugriff auf persistente Variable
  - ⇨ `$myID = $_SESSION["Zustand"];`
- Session beenden (Logout)
  - ⇨ `session_destroy();`

`session_id()` liefert die aktuelle SessionID als String

außerdem bei Bedarf:
`$_SESSION = array();` // löscht Daten
`setcookie(...);` // Cookie löschen

# PHP Sessionverwaltung mit Cookies – ein Problem (und eine Lösung)

- Beim Öffnen einer Session wird festgelegt, wie lange die Session gültig bleibt
  - ⇨ default: so lange der Browser geöffnet bleibt
  - ⇨ Änderung auf eine Lebensdauer: vor `session_start()` - über `session_set_cookie_params(int $lifetime_in_sec)`
- Aber Vorsicht
  - ⇨ auch wenn die Lebenszeit relativ angegeben ist - der Wert für das Cookie wird aus der Uhrzeit des Webservers und der Lebenszeit berechnet und auf dem Client absolut gesetzt!
  - ⇨ Wenn die Uhrzeit auf dem Client falsch gesetzt ist, kann es sein, dass das Cookie sofort ungültig ist bzw. länger gilt als gewünscht
- Lösung
  - ⇨ Das Cookie wird ohne Einschränkung ausgeliefert und übertragen
  - ⇨ Der Server entscheidet selbst, ob eine Session noch gültig ist
  - ⇨ Konfigurierbar in der PHP.ini über *session.gc_maxlifetime*

# Sessions und Login


- beim (erfolgreichen) Login

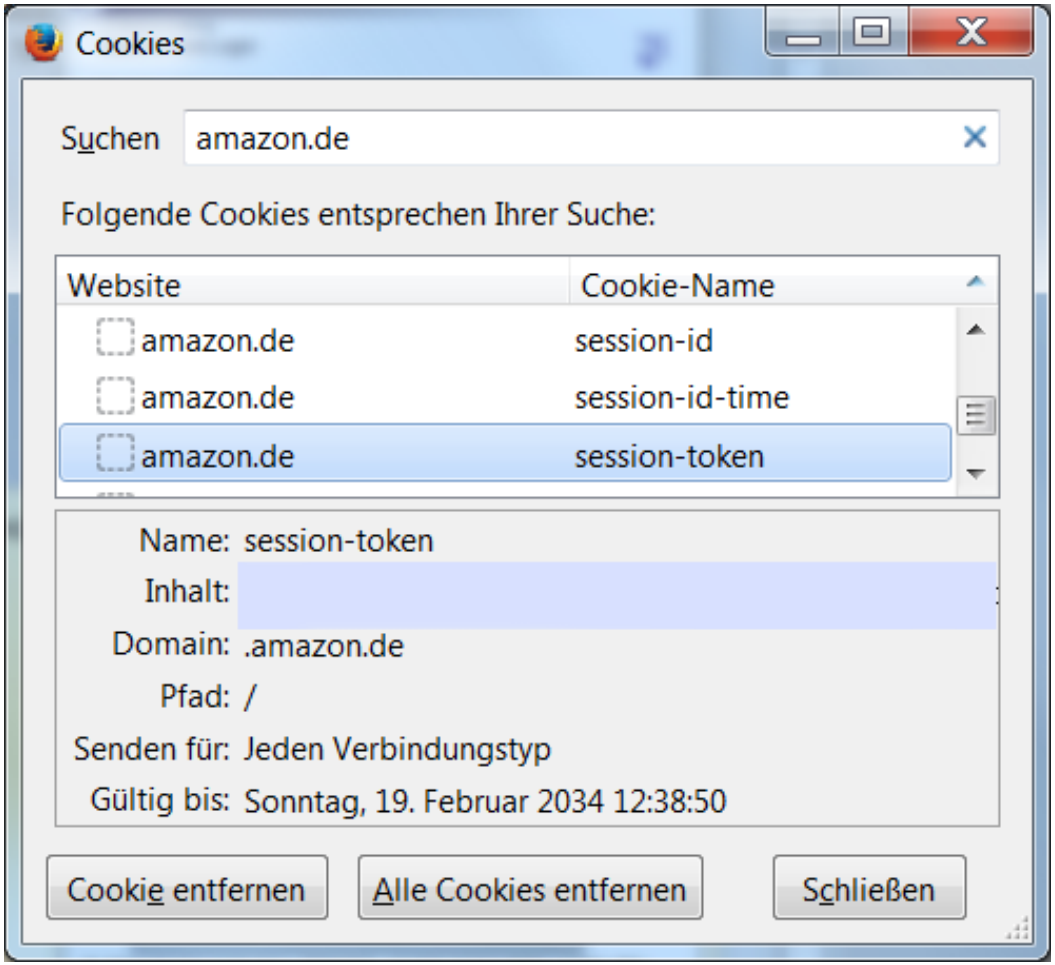
```
session_start();
if ... isset($_POST['Username']) && preg_match(...) {
   $_SESSION["User"] = $_POST['Username'];
   $_SESSION['LastAccess'] = time();
}
```

- beim Zugriff

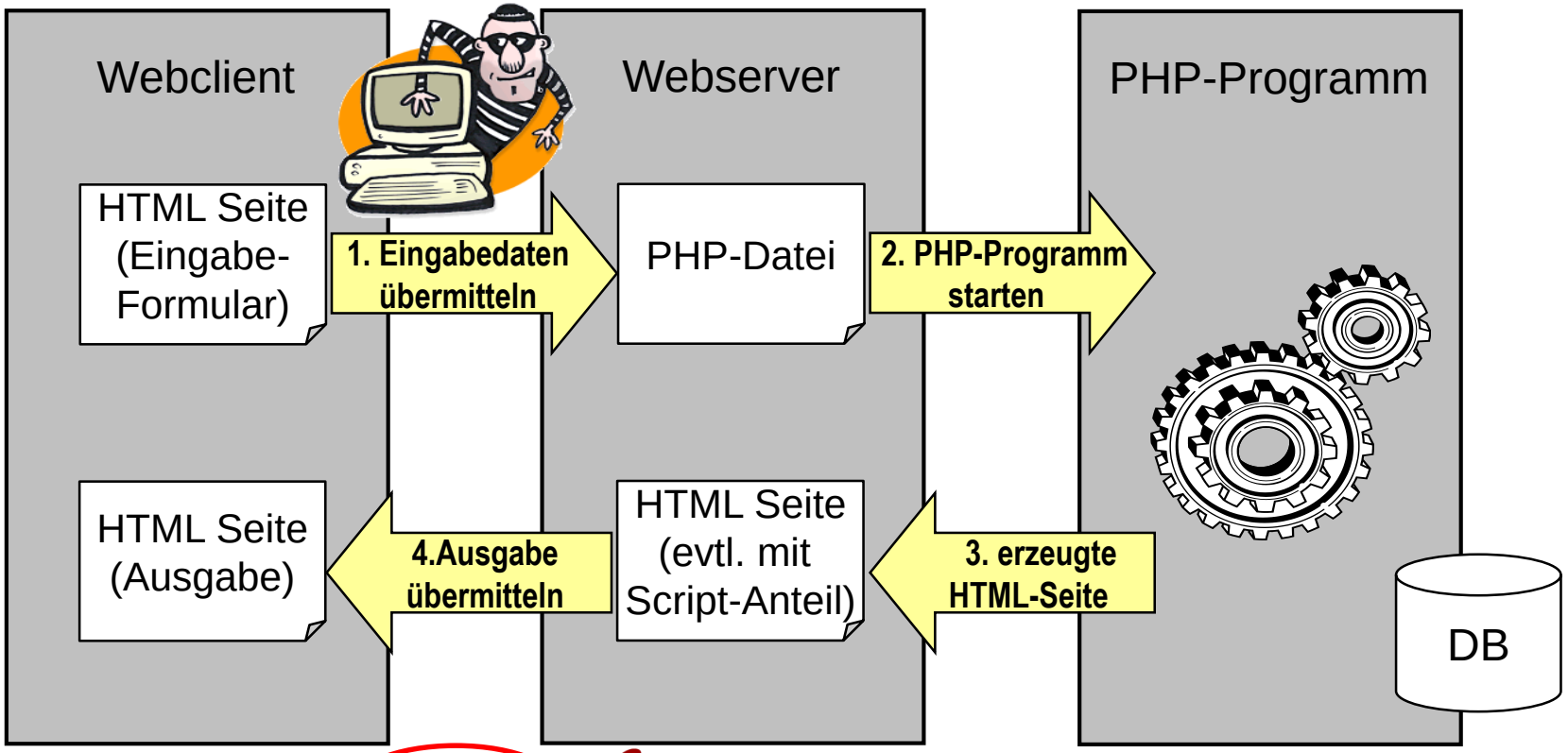
```
session_start();
if (isset($_SESSION['LastAccess']) &&
    time()-$_SESSION['LastAccess']<SessionTimeoutSec)) {
   $User = $_SESSION["User"];
   $_SESSION['LastAccess'] = time(); // Inaktivitätsdauer = 0
   $SQLabfrage = ... WHERE User ="$User";
}
```

# Sessions und Cookies im Browser

- Browser speichern eine Vielzahl von Cookies
  - ⇨ darunter auch session-id‘s uvm.
  - ⇨ das ermöglicht kundenspezifische Empfehlungen „Diese DVD könnte Sie interessieren...“
  - ⇨ aber auch eine erschreckende Transparenz z. B. Werbung auf anderen Websites, die genau das bietet, was Sie gestern gesucht haben...



# Übersicht




- HTML
- CSS
- ECMA-Script
- DOM
- AJAX

- HTTP

- Server-Konfiguration

- CGI
- PHP
- MySQLi
- Sessions

# Hochschule Darmstadt
# Fachbereich Informatik

## 5. Sicherheit

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

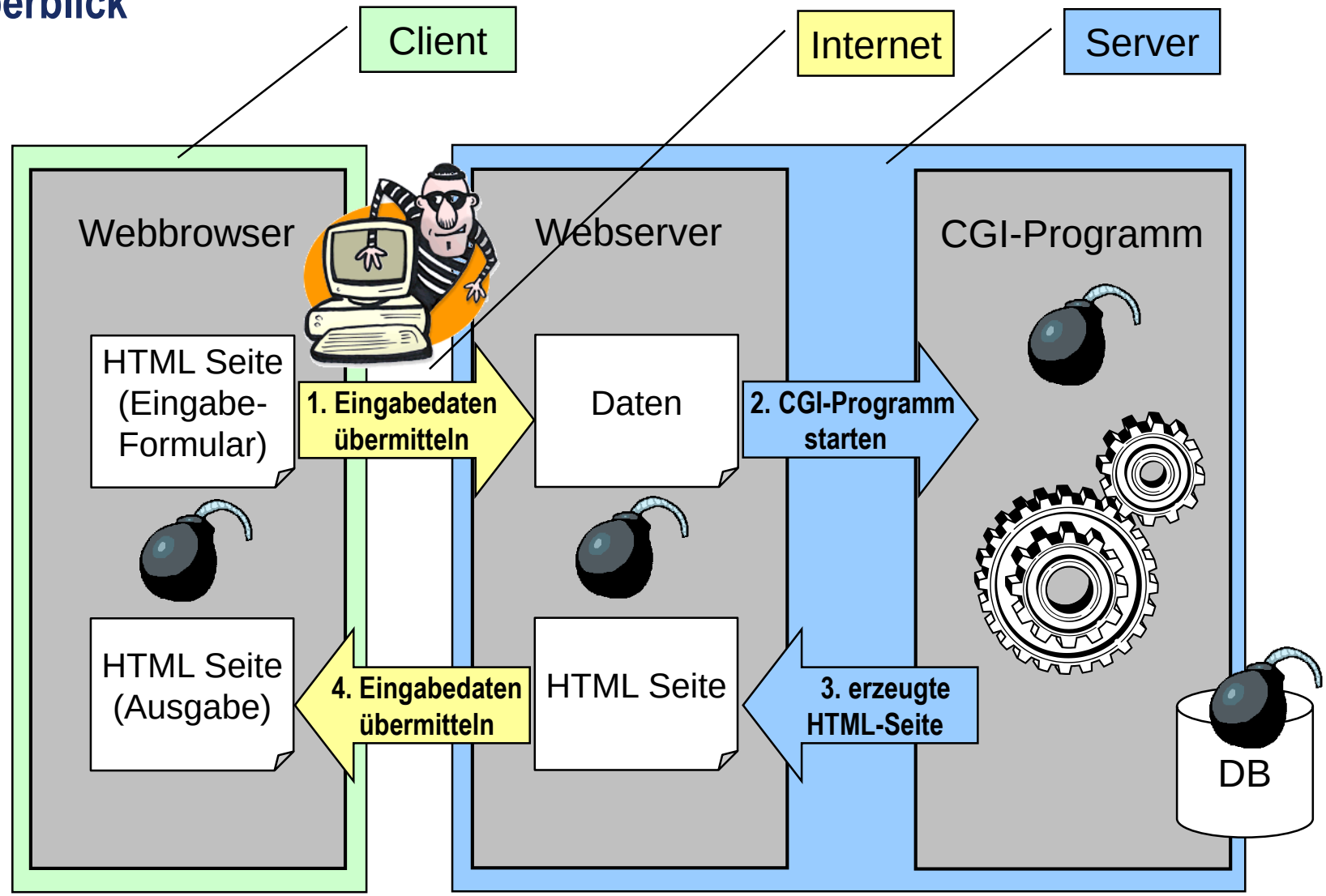
# Vorbemerkung

- Dieser Abschnitt bietet lediglich einen Einstieg in das Thema
  - ⇨ es werden ständig neue Sicherheitslücken und Angriffsformen entdeckt
  - ⇨ es gibt große Unterschiede zwischen den Betriebssystemen, Browsern, Webservern etc.
  - ⇨ der Sicherheitsbedarf hängt von der Anwendung ab

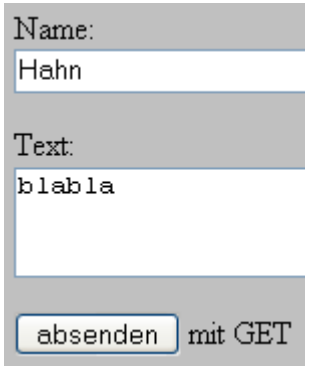
- Für den Betrieb einer professionellen Webanwendung ist Sicherheit ein Dauerthema
  - ⇨ wenn Sie das nicht leisten können, mieten Sie sich einen Server mit einem entsprechenden Update-Service für Betriebssystem, Apache, PHP, DB etc.
  - ⇨ dann müssen Sie "nur" noch kontinuierlich Sicherheitslücken in Ihrer Anwendung beseitigen

**Sie sollen hier für das Thema Sicherheit sensibilisiert werden !**

# Überblick


Client
Internet
Server
Webbrowser
Webserver
CGI-Programm
HTML Seite (Eingabe-Formular)
1. Eingabedaten übermitteln
Daten
2. CGI-Programm starten
HTML Seite (Ausgabe)
4. Eingabedaten übermitteln
HTML Seite
3. erzeugte HTML-Seite
DB

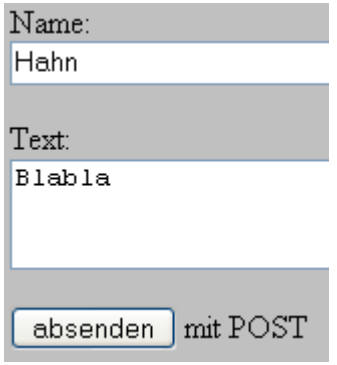
# Datenübertragung mit GET




```
<form action="21_FormularEcho.php" method="get">
    <p>Name:<br><input maxlength="40" size="40" name="Name" ></p>
    <p>Text:<br><textarea name="Text" rows="2" cols="40"></textarea></p>
    <p><input type="submit" value="absenden"> mit GET</p>
</form>
```

- überträgt die Daten für jeden sichtbar als URL:
  - ⇨ http://localhost/xxx.php?Name=Hahn&Text=blabla
- Parameter und Werte können ohne jegliche Tools verändert werden
  - ⇨ unerwartete Parameter (fehlende oder zusätzliche)
  - ⇨ unerwartete Werte

# Datenübertragung mit POST




```
<form action="21_FormularEcho.php" method="post">
    <p>Name:<br><input maxlength="40" size="40" name="Name" value="c"></p>
    <p>Text:<br><textarea name="Text" rows="2" cols="40">d</textarea></p>
    <p><input type="submit" value="absenden"> mit POST</p>
</form>
```

- Parameter werden im HTTP Body übertragen
- Formularfelder sind über den HTML-Code zugänglich
- Mit etwas HTML-Kenntnissen können Werte und Parameter manipuliert werden
  - ⇨ unerwartete Parameter (fehlende oder zusätzliche)
  - ⇨ unerwartete Werte

# Daten aus Formularen

jeden Parameter strengstens kontrollieren !

- die Parameter müssen nicht im entferntesten dem erwarteten Format entsprechen !
  - ⇨ vielleicht hat sie ein Hacker von Hand gemacht...
  - ⇨ ein erwarteter Parameter fehlt / ein nicht erwarteter wird übermittelt
  - ⇨ ein Parameter-Name enthält Sonderzeichen

- folgende Annahmen sind alle falsch:
  - ⇨ der QUERY_STRING passt in den Speicher
  - ⇨ der QUERY_STRING erfüllt die HTTP-Spezifikation
  - ⇨ QUERY_STRING-Felder entsprechen dem Formular
  - ⇨ der Wert einer Auswahlliste ist einer der Listeneinträge
  - ⇨ ein Eingabefeld sendet maximal so viele Zeichen, wie in maxlength festgelegt

➔ Apache

➔ PHP

➔ Program-mierer

# Formularparameter grundsätzlich validieren

ereg vermeiden; ist nicht binary safe und deprecated

- Wo ?
  - ⇨ serverseitig zur Sicherheit
  - ⇨ clientseitig (JavaScript) nur als Benutzerkomfort
    - der Benutzer könnte JavaScript abgeschaltet haben; ein Hacker könnte dies gezielt tun

- Problem der Vollständigkeit
  - ⇨ die gültigen Zeichenfolgen spezifizieren (z.B. über regulären Ausdruck)
  - ⇨ nicht die ungültigen, sonst würden vergessene Fälle akzeptiert

- Achtung
  - ⇨ Der PHP-Befehl `ereg()` (zur Ersetzung mit regulären Ausdrücken) ist nicht binärsicher, d.h. er sucht nur bis zur ersten '\0' nach Übereinstimmungen mit einem regulären Ausdruck

# Angriffe allgemein

- Es gibt viele Installationen mit Standardkonfiguration (Webserver, Datenbank, CGI, PHP,… vgl. XAMPP)
  - ⇨ Webserver geben großzügig Auskunft über die Versionen
  - ⇨ Default-Passwörter sind bekannt
  - ⇨ installierte Skripte sind teilweise bekannt
  - ⇨ Quellcode ist verfügbar
  - ⇨ Sicherheitslücken können gesucht und erprobt werden

- Angriffsziele (Auswahl):
  - ⇨ Ausführen von Befehlen
  - ⇨ Auslesen von Daten
  - ⇨ Transaktionen unter fremden Namen
  - ⇨ Lahmlegen eines Internetauftritts

# Angriffe auf den Webserver

- Auslesen von Daten auf dem Server
  - ⇨ Geheime Daten
  - ⇨ Passwörter (.passwd oder aus Datenbank)
  - ⇨ …
- Lahmlegen des Servers
  - ⇨ z.B. durch Überlastung mit Requests ("Denial of Service Attack")
  - ⇨ Löschen von Dateien
  - ⇨ Passwörter nicht im Klartext speichern: Hash verwenden

**gewissenhaft und restriktiv konfigurieren!**

# Code-Injektion

- Große Gefahr durch Ausführung von Systembefehlen (oder anderen Programmen) mit Parametern aus einem Formular: (durch unerwünschte Befehle innerhalb des Parameters)
  - ⇨ 2. Unix-Befehl mit `;` angehängt
    Beispiel: Suche nach Unix-Benutzer löscht alle Dateien
    Eingabe per Formular: gesuchter Name, z.B. `james`
    Implementierung: `passthru('finger '.$_POST['name']);`
    Angriff: `james; rm -rf /`
  - ⇨ String als PHP-Code ausführen mit `eval("PHP-Code")`
    Beispiel: Berechnung von arithmetischen Ausdrücken zeigt Passwortdatei an
    Eingabe per Formular: ein Ausdruck, z.B. `(3*4)+5`
    Implementierung: `eval("echo {$_POST['name']};")`
    Angriff: `file_get_contents('/etc/passwd')`
- Systemaufrufe in CGI Skripten sollten möglichst vermieden werden
  - ⇨ Inhalt eventuell mit `escapeshellarg()` etc. sichern

passthru gibt einen String zur Ausführung an das System und gibt das Ergebnis zurück

alle Parameter, die an unumgänglichen Systemaufrufen beteiligt sind, besonders sorgfältig kontrollieren!

# Zugriff auf Dateien im Server

- Anwendung: Dateiverwaltung (z.B. Fotoalbum) via Web

  Link auf Bild: `<a href="fetch.php?img=237.jpg">vergrößern</a>`

  Angriff: `../../../etc/passwd`

- Problem: Manipulierte Dateinamen können andere Dateien liefern bzw. abrufen als beabsichtigt

- Dateinamen aus Formularen, PATH_INFO und anderen Quellen sind verdächtig
  - ⇨ auch Dateinamen, die aus solchen Bestandteilen gebildet werden
  - ⇨ evtl. im Server gegen eine Liste von erlaubten Dateinamen prüfen
- fest im Skript codierte Dateinamen sind unproblematisch
  - ⇨ zumindest den Pfad fest kodieren
- in Dateinamen keine .. und keine shell-Steuerzeichen zulassen
  - ⇨ besser: in Dateinamen nur a..z, A..Z, 0..9, -, _ zulassen

# HTTP und HTTPS

- HTTP ist unverschlüsselt
  - ⇨ kann mit Sniffer (z.B. Wireshark) abgehört werden
  - ⇨ Passwort und persönliche Daten im Klartext
- HTTPS ist verschlüsselt mit SSL
  - ⇨ erfordert (gekauftes) Zertifikat und Zertifizierungshierarchie
  - ⇨ selbstgemachte Zertifikate sind nach Installation genauso sicher
  - ⇨ Angriffsmöglichkeit für "man in the middle" während Installation
- Konsequenz für Entwickler:
  - ⇨ jede Website mit Benutzerverwaltung sollte HTTPS verwenden
- Konsequenz für Anwender:
  - ⇨ nicht dasselbe Passwort für verschlüsselte und unverschlüsselte Verbindungen verwenden

# Zusätzliche Formularparameter

- PHP kann Formularparameter als einfache globale Variable bereitstellen (ab PHP 5.4 nicht mehr verfügbar)
- PHP interpretiert nicht initialisierte Variablen als false

- Angriff und Abwehr:
  - ⇨ Entwickler wertet nicht initialisierte Variable aus
  - ⇨ Hacker fügt weiteres Formularelement ein und initialisiert so die Variable

  - ⇨ Entwickler sollte jede Variable initialisieren
    - Warnung aktivieren: error_reporting(E_ALL);
  - ⇨ Entwickler sollte besser über $_GET oder $_POST auf Formularparameter zugreifen
    - zur Unterscheidung von Hilfsvariablen

```
if (...)
   $ok = true;

if ($ok)
   ...
```

```
<input type="hidden"
 name="ok" value="1">
```

# Angriff auf die Datenbank: SQL-Injektion

- Formularparameter werden oft in SQL-Anweisungen integriert
- Angriff: Hacker sendet einen Parameterstring, der die korrekte SQL-Anweisung verändert oder eine weitere anfügt
- Abwehr
  - ⇨ Entwickler muss jeden Parameter auf Gültigkeit prüfen (z.B. regul. Ausdr.)
  - ⇨ Entwickler muss SQL-Sonderzeichen ersetzen mit `mysqli::real_escape_string()`

```
$sql="SELECT * FROM student WHERE "
  ."matnr='$matnr' AND pwd='$pwd';"
```

**Password:** **1' OR '1=1**

# Zugriff auf Datensätze fremder Benutzer

- Jeder Seitenabruf innerhalb einer Session muss einem bestimmten Benutzer zugeordnet werden
  - ⇨ z.B. über Matrikelnummer oder Kundennummer
- Angriff
  - ⇨ Hacker könnte Benutzeridentifikation fälschen, falls im Klartext transferiert
- Abwehr:
  - ⇨ Entwickler: Benutzeridentifikation nicht aus (versteckten) Formular- oder URL-Parametern entnehmen
  - ⇨ Entwickler: Login identifiziert den Benutzer erstmalig; auf Folgeseiten erfolgt dies über SessionID und Session-Variable
    - Benutzerdaten werden bei Bedarf über Session-Variable weitergegeben

# Session-Hijacking: Einbruch in fremde Session

- Basis: Ausspähen der SessionID
  - ⇨ Benutzer ist eingeloggt, SessionID wird mit jeder Seite und Seitenanforderung übertragen
  - ⇨ Hacker ersetzt eigene SessionID durch fremde
- Angriff:
  - ⇨ Abhören des Netzverkehrs mit Sniffer
  - ⇨ Blick oder Foto über die Schulter des Benutzers
- Abwehr
  - ⇨ Verschlüsselte Verbindung (HTTPS) verwenden
  - ⇨ Benutzer sollte sich nicht beobachten lassen
  - ⇨ Benutzer sollte Cookies nicht abschalten (damit PHP die SessionID nicht an die URL anhängt)
  - ⇨ Benutzer sollte nicht eingeloggt bleiben und weggehen
  - ⇨ Entwickler sollte lange und kryptische SessionID generieren
  - ⇨ Entwickler kann zusätzlich HTTP_REFERER und REMOTE_ADDR prüfen

# Einbruch in fremde Anwendungsfälle

- Basis
  - ⇨ Es gibt Benutzer mit verschiedenen Rollen
    z.B. OBS mit Sekretariat, Dozenten, Admin
- Angriff
  - ⇨ Benutzer einer eingeschränkten Rolle ruft Anwendungsfall einer mächtigeren Rolle auf
    - z.B. durch Manipulation der GET-Parameter
- Abwehr
  - ⇨ rollenspezifisch eingeschränktes Menü ist komfortabel für den Benutzer, genügt aber nicht für die Sicherheit
  - ⇨ Entwickler muss jeden Seitenaufruf rollenspezifisch kontrollieren (zwecks Übersichtlichkeit tabellarisch implementieren)

# Browser History

- Der Browser speichert die zuletzt aufgerufenen URLs einschließlich Parametern, egal ob GET oder POST verwendet wurde
  - ⇨ also auch ein Login mit Benutzername und Passwort
- Angriff
  - ⇨ nach dem Ausloggen eines Benutzers kann der nächste Benutzer aus der Browser History das Login seines Vorgängers erneut aufrufen
  - ⇨ insbesondere an öffentlichen Surfstationen (Internet Café, Web Kiosk)
  - ⇨ auf diese Weise hat einmal ein Student im OBS seinem Vorgänger am Web Kiosk im Foyer des FbI dessen Plätze gelöscht und sie dann schnell selbst belegt ...
- Abwehr
  - ⇨ Benutzer: Browser nach Gebrauch schließen und History löschen
    - Problem: Web Kioske lassen das meistens nicht zu
  - ⇨ Entwickler: POST-Daten bei erneuter Übertragung nicht verarbeiten
    - z.B. mit Hilfe des TAN-Verfahrens, siehe Folie "View-Controller: sichere Formulare" im Abschnitt "Model-View-Controller Framework"

# Unerwünschte Skripte auf einem Client ausführen

- Basis:
  - ⇨ Eine Website (z.B. ein Forum oder Auktionshaus) prüft eingestellte Beiträge nicht gründlich und erzeugt aus den fremd-eingestellten Texten HTML-Seiten, die von Benutzern abgerufen werden
  - ⇨ "Sorglose Surfer" (z.B. auf der Suche nach PHP-Tipps in diversen Foren und Blogs) sind gefährdet
- Angriff: Cross-Site-Scripting (XSS)
  - ⇨ Ein Hacker stellt "Texte" mit Javascript-Anteilen ein (z.B. eine Login-Maske, welche die Eingaben weiterleitet)
  - ⇨ Die Skripte werden auf dem Client des Benutzers ausgeführt
- Abwehr:
  - ⇨ Benutzer: Javascript abschalten (dann funktionieren aber viele Webseiten nicht mehr vernünftig)
  - ⇨ Entwickler: Unbedingt alle Ausgaben so kodieren, dass HTML-Code als normaler Text erscheint – und keinesfalls interpretiert wird
    - `htmlspecialchars()` aufrufen

# Unerwünschte Skripte auf einem Server ausführen

- Basis
  - ⇨ Ein Webauftritt integriert einen Teil eines anderen Webauftritts
    Beispiel: Auf www.fbi.h-da.de könnte eine Karte mit Wegbeschreibung von einem Kartenanbieter integriert sein
- Angriff (Variante des Cross-Site-Scripting):
  - ⇨ Jemand hackt sich in das Netzwerk der Hochschule, fängt die Anfragen an den Kartenanbieter ab und ersetzt sie durch "erweiterte" Karten ("man in the middle")
  - ⇨ Gefahr für Server, wenn der gelieferte Code ausgeführt wird
  - ⇨ Gefahr für Client, falls der gelieferte Code JavaScript enthält
- Abwehr:
  - ⇨ Den Code nicht ausführen, sondern nur ausliefern:
    Entwickler kann `readfile("URL")` (liefert String) statt `require` bzw. `include("URL")` verwenden
  - ⇨ Admin kann Öffnen von URLs in PHP.INI generell sperren:
    `allow_url_fopen = 0`

dann geht readfile("URL") auch nicht mehr!

# Ausspähen des Datenbank-Passworts

- Basis
  - ⇨ Das Passwort zum Zugriff von PHP auf die Datenbank steht im PHP-Quellcode im Klartext
- Angriff:
  - ⇨ direkter Abruf der Passwort-Datei wenn URL bekannt
  - ⇨ durch Fehler im Webserver oder im PHP-Interpreter könnte eine PHP-Datei gezeigt statt ausgeführt werden
- Abwehr:
  - ⇨ Entwickler: Passwort in eigene Datei in geschütztem Verzeichnis
  - ⇨ Entwickler: Passwort in .php-Datei, nicht etwa .txt oder .html
  - ⇨ Entwickler: Passwort-Datei bei Open Source-Veröffentlichung rausnehmen (oder bei Installation abfragen)
  - ⇨ Admin: Datenbank nur vom localhost ansprechbar
    - nur möglich, wenn keine weiteren Datenbank-Clients existieren

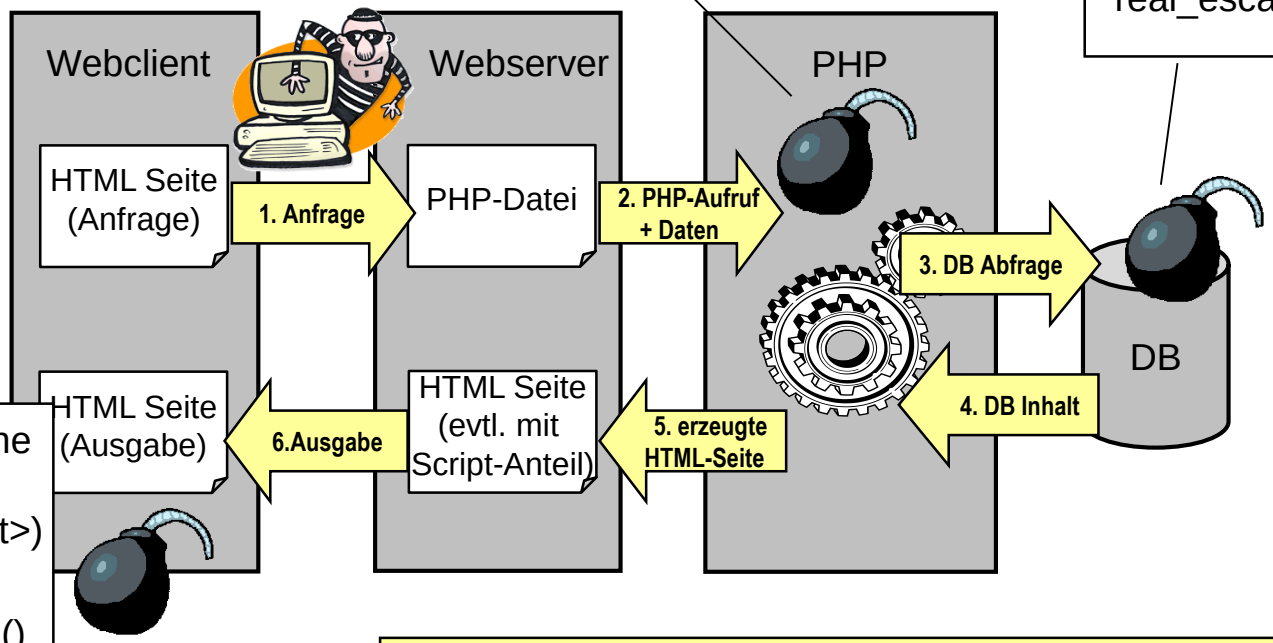
# Infos für Hacker aus Fehlermeldungen

- Fehlermeldungen sind wichtig, auch im Produktivbetrieb
  - ⇨ aber dann bitte in einer Log-Datei und nicht auf dem Bildschirm
- Fehlermeldungen aus Bibliotheken nicht dem Benutzer zeigen
  - ⇨ versteht er meist ohnehin nicht
  - ⇨ enthalten evtl. wertvolle Hinweise für Hacker
  - ⇨ Entwickler muss dem Benutzer anwendungsbezogene Meldungen zeigen
  - ⇨ Entwickler muss systembezogene Meldungen in Log-Datei schreiben
    - `ini_set("error_reporting", E_ALL);` // alles melden
      `ini_set("display_errors", 0);` // aber nicht ausgeben
      `ini_set("log_errors", 1);` // sondern loggen
      `ini_set("error_log", "./mylog.txt");` // in diese Datei
    - Vorsicht: `mysqli::error()` zeigt u.U. DB-Passwort in Fehlermeldung

# Pflichtmaßnahmen in PHP zur Sicherheit

Ausschluss von ausführbaren Befehlen
Übergebene Daten genau überprüfen;
Vergleich mit Listen / reg. Ausdrücken

Für MySQL "gefährliche" Zeichenfolgen umwandeln:
mysqli::
real_escape_string(...)



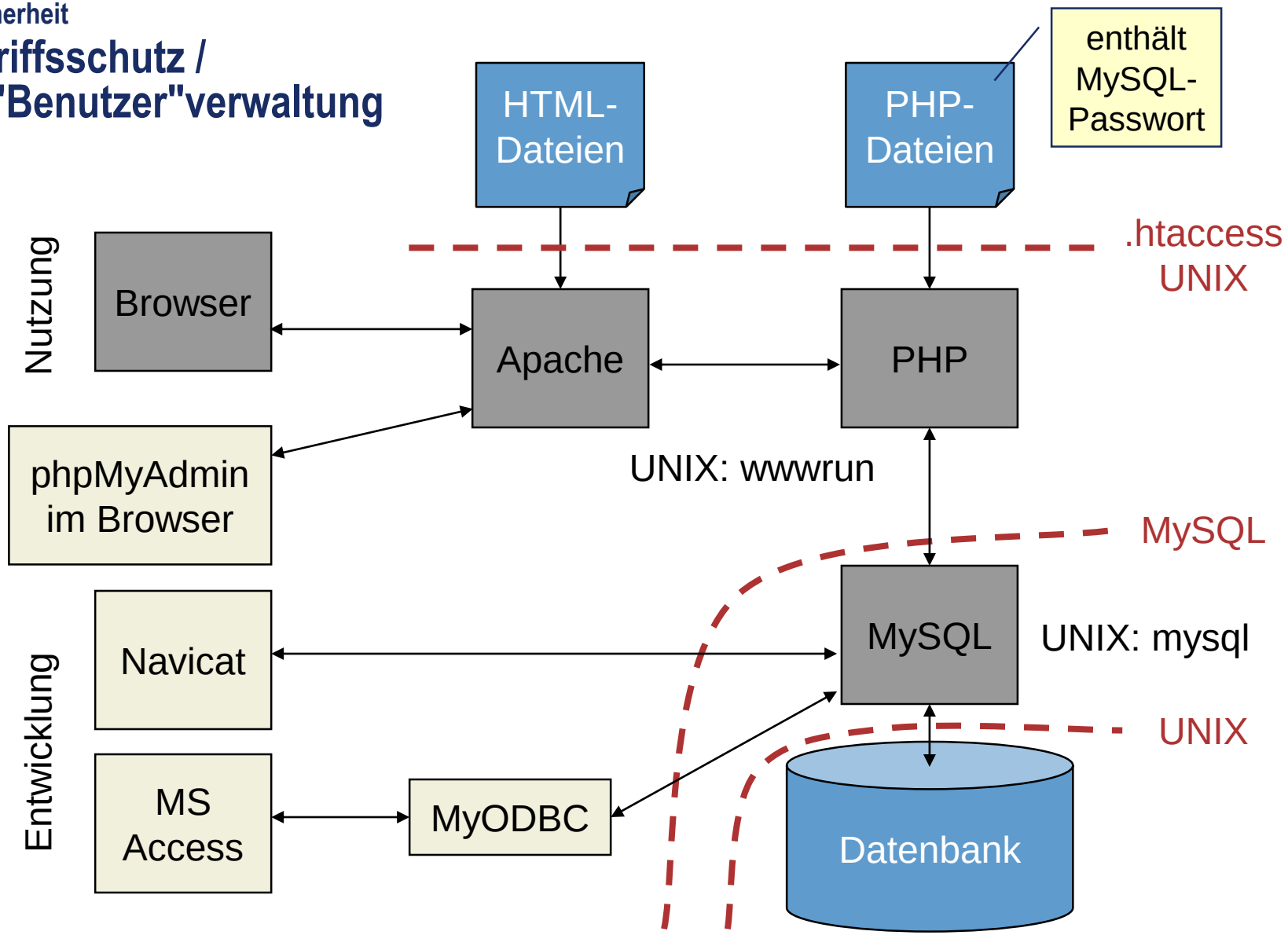

Für Client gefährliche Zeichenfolgen (z.B. XSS mit <script>) umwandeln:

html_special_chars()

außerdem HTML-konform und international!

Tipp: Das "HTML_Quickform2"-Paket von PEAR wird benutzt, um HTML-Formulare komfortabel aus PHP zu erzeugen. Optional erzeugt es auch eine Überprüfungsroutine (validation), die serverseitig und auch clientseitig (Javascript) funktioniert.

# Zugriffsschutz / "Benutzer"verwaltung



# Interessante Links

- Bundesamt für Sicherheit in der Informationstechnik (BSI)
  https://www.bsi.bund.de
  - ⇨ Lagebericht zur IT-Sicherheit in Deutschland
  - ⇨ IT Grundschutz
  - ⇨ uvm.

- OWASP : Open Web Application Security Project
  https://www.owasp.org
  - ⇨ Eine Projekt zur Vermittlung von Know-How im Bereich IT-Sicherheit
  - ⇨ Eine Anwendung, die (absichtlich) Sicherheitslücken aufweist, welche in verschiedenen Lektionen gefunden werden sollen

# Hochschule Darmstadt
# Fachbereich Informatik

## 6. Professionelle Webentwicklung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Motivation

- Professionelle Webentwicklung bedeutet
  - ⇨ dass Sie eine Webanwendung "professionell" entwickeln
    d.h. beruflich als Fachmann bzw. Fachfrau
  - ⇨ dass in der Regel jemand von Ihrem Arbeitsergebnis "lebt" und für eine adäquate Qualität bezahlt
  - ⇨ dass je nach Qualitätsanforderungen (z.B. Wartbarkeit, Testbarkeit, Robustheit, Verteilbarkeit, Sicherheit etc.)
    - ein entsprechendes Vorgehen gewählt und
    - entsprechende Ergebnisse erarbeitet werden
  - ⇨ dass Sie ein bewusstes Vorgehen haben

Auch für webbasierte Anwendungen gilt alles, was Sie über Softwaretechnik, Software Ergonomie und GUIs gelernt haben!

# Web Engineering

- Web Engineering
  - ⇨ ist ein Zweig des Software Engineering, der sich mit der Entwicklung von Webanwendungen beschäftigt
  - ⇨ ist wichtig für komplexe Websites wie Portalsysteme, Shops etc.
  - ⇨ passt die klassischen Methoden der Softwaretechnik für den gesamten Lebenszyklus einer Webanwendung an – und berücksichtigt dabei die Besonderheiten von Webanwendungen
- Besonderheiten bei Webapplikationen
  - ⇨ hoher Gestaltungsanteil
  - ⇨ Benutzbarkeit ist extrem wichtig
  - ⇨ Kunde/Auftraggeber weiß oft nicht (genau), was er will

Es entstehen spezielle Vorgehensmodelle und spezielle Arbeitstechniken für die einzelnen Phasen der Entwicklung!

# "Do It Yourself" oder "Off the Shelf"-Entwicklung?

- Es gibt für typische Websites fertige Lösungen (Off-the-Shelf)
  - ⇨ Homepage-Pakete bei vielen Internet-Providern
  - ⇨ mit branchenspezifischen Vorlagen und konfigurierbarem Design
  - ⇨ mit konfigurierbaren Elementen wie Feedback-Formular, Foto-Galerie, Kontaktformular, Gästebuch, Shops, Maps usw.
  - ⇨ mit eigener Domain und Emailadresse
  - ⇨ gegen geringe monatliche Kosten
- Den Betrieb übernimmt der Provider
  - ⇨ Hardware, Aktualisierung und Wartung von Webserver & Co.
  - ⇨ Unterstützung für neue Clients (z.B. Smartphones)
- Derartige Homepage-Pakete sind oft sehr gute Lösungen
  - ⇨ für Webauftritte ohne exotische Anforderungen
  - ⇨ die Pflege der Inhalte ist einfach (nach dem Aufsetzen des Auftritts) und erfordert keine Spezialisten wie bei einer "Do It Yourself"-Entwicklung

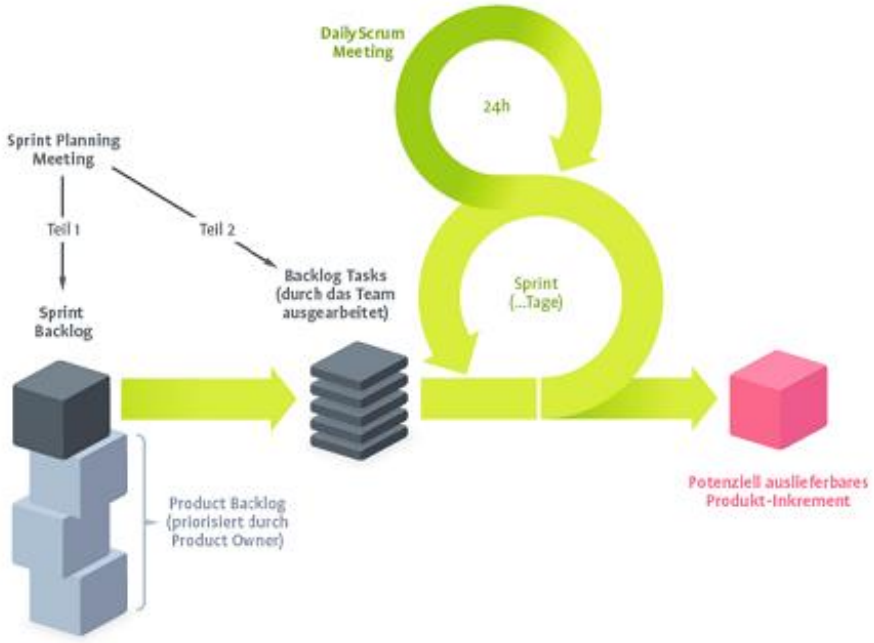
# Hochschule Darmstadt
# Fachbereich Informatik

## 6.1 Vorgehensmodelle in der Webentwicklung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Web Engineering: Vorgehensmodell Scrum

- Iterative und inkrementelle Entwicklung mit hoher Kundenbeteiligung
- Prozess zur Durchführung der Entwicklung in kurzen iterativen Zyklen ("Sprints").
- Fokussiert auf die Entwicklung und sollte in anderes Vorgehensmodell eingebettet werden
- Einsatzfähige Teilergebnisse für den Auftraggeber schon nach relativ kurzer Zeit




Adaption von "Agile Software Development with Scrum" von K. Schwaber, M. Beedle
http://slides.liip.ch/about-liip/slide_18.html

# Hochschule Darmstadt
# Fachbereich Informatik

## 6.2 Arbeitstechniken in der Webentwicklung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

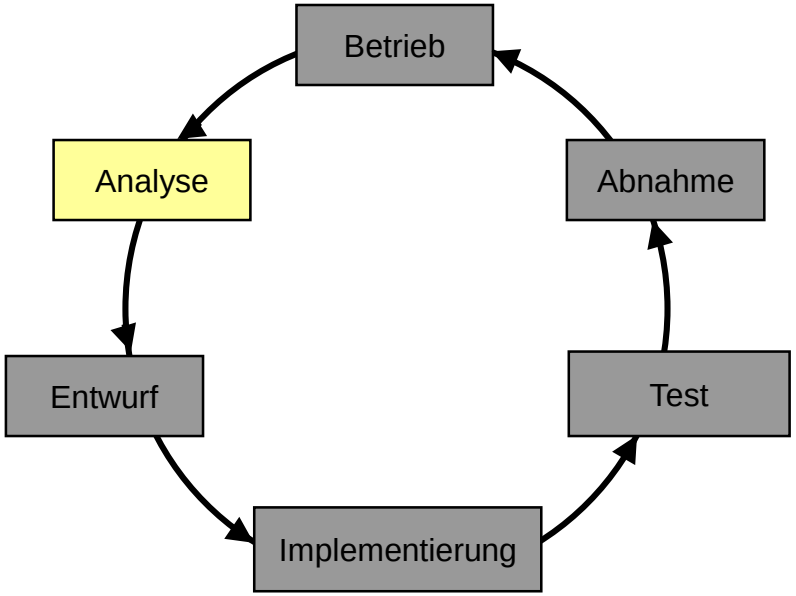
# Anpassung von Arbeitstechniken (Beispiele)

| Phase | Arbeitstechnik (Beispiel) |
|---|---|
| ■ Anforderungsanalyse | ■ Story Cards statt Use Cases |
| ■ Architektur / Design | ■ Spezielle Architekturen<br>■ Zusätzliche Modelle für Navigation, Präsentation, Hypertext etc. |
| ■ Implementierung | ■ Spezielle Programmiersprachen und Implementierungsrichtlinien |
| ■ Test | ■ Automatisierte Tests, automatisierte GUI-Tests |
| ■ Wartung | ■ Erzeugung von Dokumentationen aus Code und Testfällen |

Im folgenden werden einige Beispiele für die Phasen präsentiert!
Einige davon verwenden PHP.
In anderen Sprachen gibt es aber ähnliche Lösungen.

# Hochschule Darmstadt
# Fachbereich Informatik

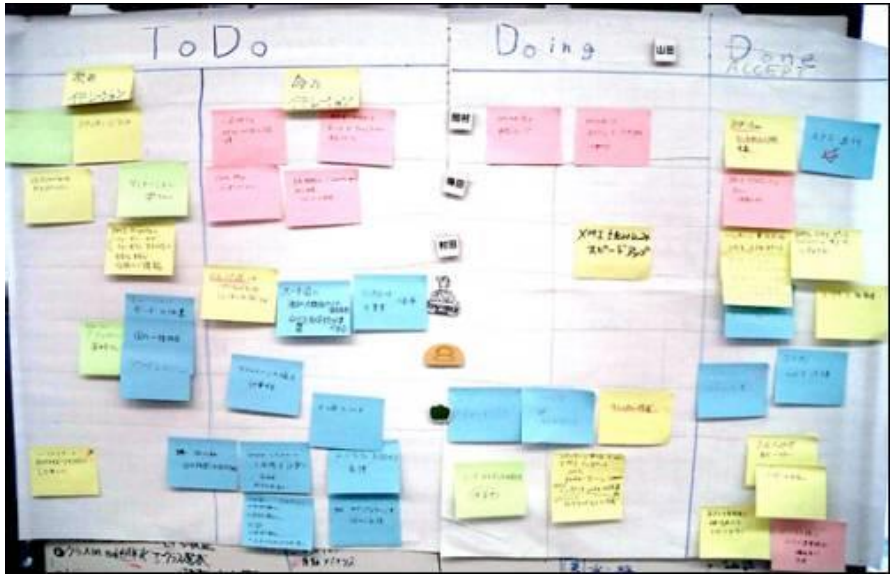
## 6.2.1 Arbeitstechniken in der Analyse



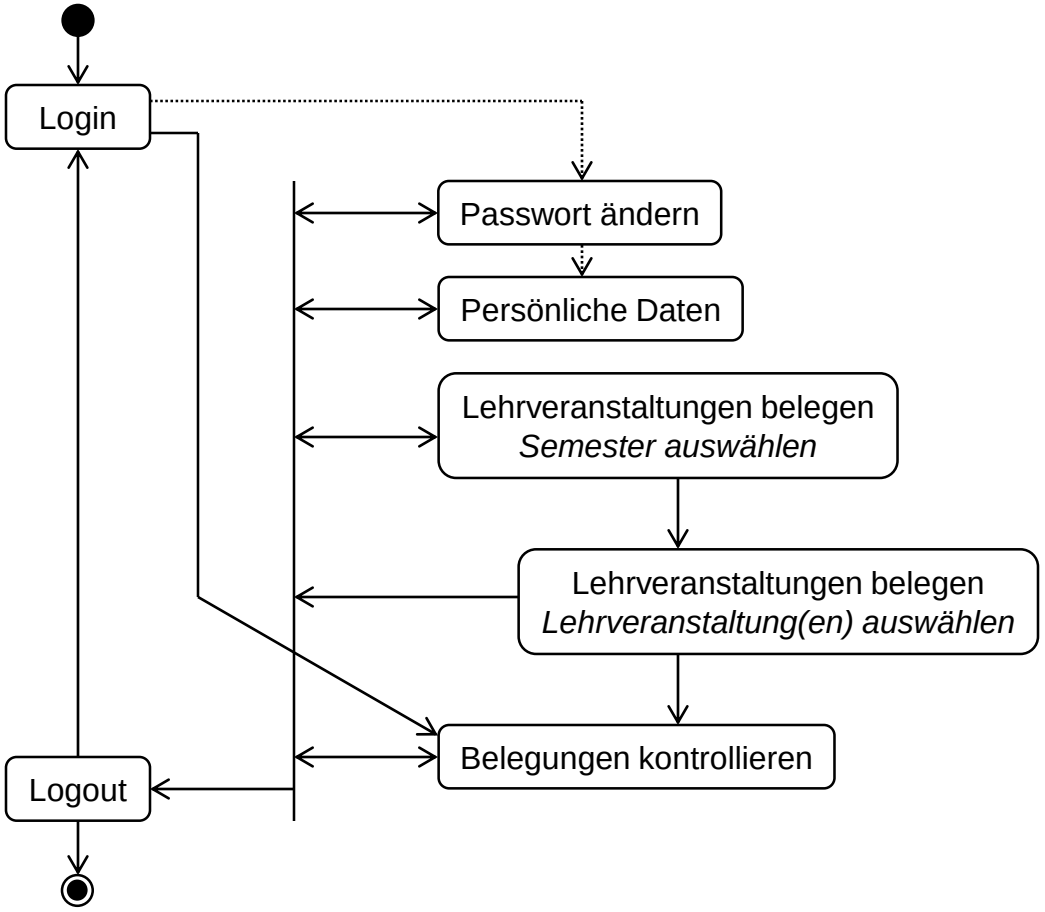
# Erfassung und Verwaltung von Anforderungen

Das kennen Sie schon !
vgl. Praktikum Aufgabe 1

- Story Cards
  - ⇨ enthalten je eine User Story – d.h. eine Funktionalität, die sich der Auftraggeber wünscht
  - ⇨ z.B. "Warenkorb füllen":
    Der Kunde klickt auf die gewünschten Pizzasymbole und die Pizzen werden in den Warenkorb übernommen. Der aktuelle Preis wird sofort angezeigt.
  - ⇨ erlauben schnelle und unkomplizierte Erfassung von Anforderungen
  - ⇨ Die Karten werden zur Projektplanung und Verfolgung eingesetzt (Priorisierung und Aufwandsschätzung)



# Navigationsübersicht als Zustandsdiagramm




**Zustände**

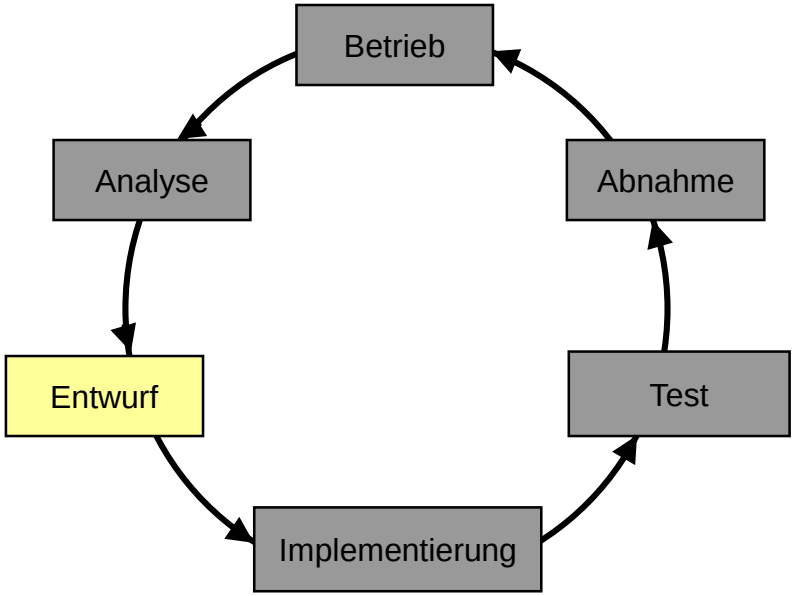
generierte HTML-Seiten

**Zustands-übergänge**

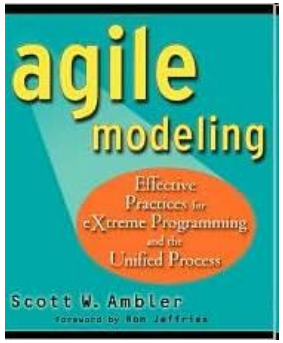
URL-Aufruf / Formular senden
↓
PHP-Programm ausführen

# Hochschule Darmstadt
# Fachbereich Informatik

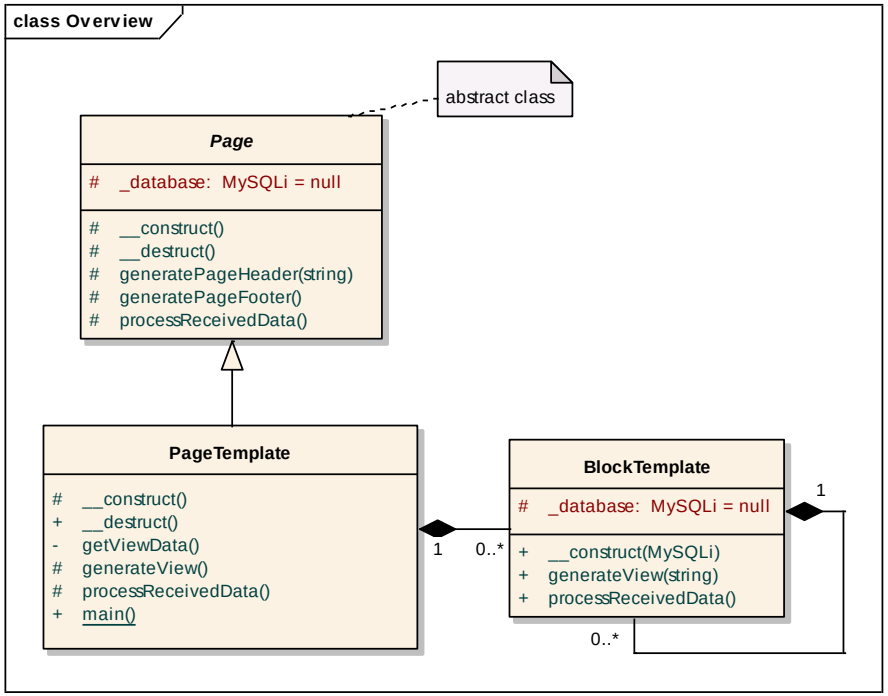
## 6.2.2 Arbeitstechniken im Entwurf



# Modellierung in Webprojekten



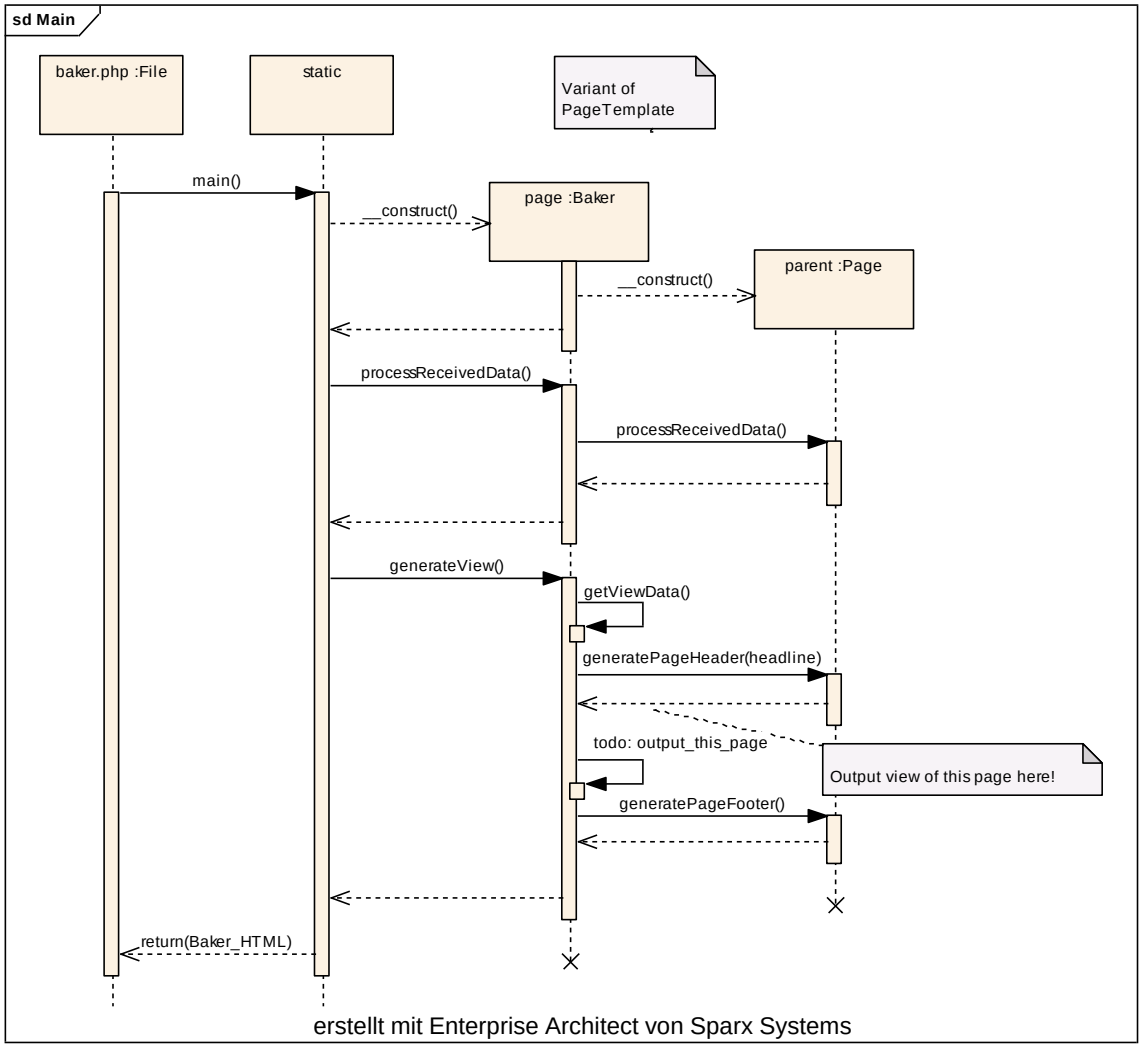

- Auch Webprojekte haben ein Konzept – und manchmal ist es sogar dokumentiert !
- Moderne Case-Tools können auch mit PHP-Code umgehen
  - ⇨ Entwurf und Dokumentation von Klassen
  - ⇨ Unterstützung der UML
  - ⇨ Import aus vorhandenen PHP-Dateien
  - ⇨ Export von Coderahmen




erstellt mit Enterprise Architect von Sparx Systems

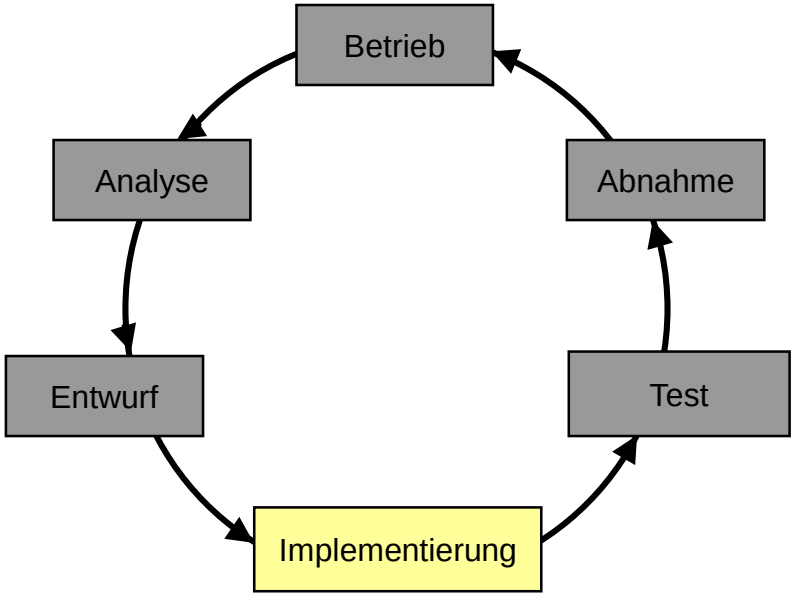
# Modellierung in Webprojekten (II)

- Die Interaktion der Klassen kann modelliert und dokumentiert werden
- "Test" der Klassen als Trocken-Übung vor der Implementierung
- Grundlage für Kommunikation im Projekt



# Hochschule Darmstadt
# Fachbereich Informatik

## 6.2.3 Arbeitstechniken in der Implementierung



# Programmierrichtlinien

https://framework.zend.com/manual/1.11/de/coding-standard.html

- Eine Auswahl von Regeln im Zend-Standard:
  - ⇨ Fehlervermeidung
    - In jeder `switch` Anweisungen muss ein `default:` enthalten sein
    - `if` für Einzelanweisungen (ohne `{}`) ist unerwünscht
    - In PHP-Dateien wird ein schließendes PHP-Tag `?>` am Dateiende weggelassen (Vermeidet unabsichtlichen "Beginn" der Ausgabe)
  - ⇨ Verbesserung der Lesbarkeit
    - Strenge Regeln für Einrückungstiefe, Leerzeichen, Klammerungen etc.
    - Leerzeichen statt Tabs, Zeilenlänge von 80 Zeichen
    - Vorgaben für Kommentare (Ort, Häufigkeit, Schlüsselworte)
    - Namenskonventionen für Klassennamen, Methoden, Attribute

"Coding Standards sind in jedem Entwicklungsprojekt wichtig, aber sie sind speziell dann wichtig wenn viele Entwickler an dem gleichen Projekt arbeiten. Coding Standards helfen sicherzustellen, dass der Code von hoher Qualität ist, weniger Fehler hat, und einfach zu warten ist."
(Zend Coding Standard)

# Überprüfung von Programmierrichtlinien

- Programmierrichtlinien werden kontrolliert
  - ⇨ z.B. mit PHP_CodeSniffer
    - zeigt Verstöße gegen Richtlinien an!
    - Aufruf: `phpcs my_source.php` (prüft PEAR-Standard)
    - oder z.B. `phpcs --standard=zend mysource.php` (Zend-Standard)
  - ⇨ Anfangs ist die Erfüllung der Richtlinien eher nervig, aber nach einer Gewöhnungsphase geht es schnell und die Wartbarkeit wird besser

```
----------------------------------------------------------------------
FOUND 24 ERROR(S) AND 4 WARNING(S) AFFECTING 22 LINE(S)
----------------------------------------------------------------------
  1 | ERROR   | End of line character is invalid; expected "\n" but found "\r\n"
  2 | ERROR   | You must use "/**" style comments for a file comment
  3 | ERROR   | Space before opening parenthesis of function call prohibited
  5 | ERROR   | Space before opening parenthesis of function call prohibited
 30 | WARNING | Inline control structures are discouraged
 31 | ERROR   | Spaces must be used to indent lines; tabs are not allowed
 32 | WARNING | Inline control structures are discouraged
 33 | ERROR   | Spaces must be used to indent lines; tabs are not allowed
 49 | ERROR   | Expected "foreach (...) {\n"; found "foreach(...) {\n"
 50 | ERROR   | Line indented incorrectly; expected at least 4 spaces, found 1
 50 | ERROR   | Spaces must be used to indent lines; tabs are not allowed
```

# Dokumentation im Code

- Erzeugung einer Dokumentation aus Kommentaren im Quellcode
  - ⇨ es müssen bestimmte Schlüsselworte und Regeln beachtet werden
    - Kommentare stehen über Klassen, Methoden und Attributen
    - am Anfang einer Datei beschreibt ein Kommentar den Inhalt der Datei usw.
  - ⇨ z.B. mit PHP Documentor
    - `phpdoc -f myfile -t ./doc` bzw. `phpdoc -d mydir -t ./doc`
  - ⇨ Doxygen bietet ähnliche Funktionalität, aber mit einem GUI

```
/*
*
* Initialisierung der bestellten Pizza mit Status & ID der Speisekarte
* Die ID wird erst beim Speichern gesetzt, liefert vorher UNDEF
*
* @param $pizzaID PizzaID der Pizza in der Speisekarte
* @param $status  Bestell_Status der Pizza
*
* @return Boolean true, wenn alles geklappt
public function initialize($pizzaNr, $kunde
...
}
```

Ausgabe als HTML, XML, RTF uvm.

**initialize ( $ pizzaNr, $ kundenNr, $ status )**

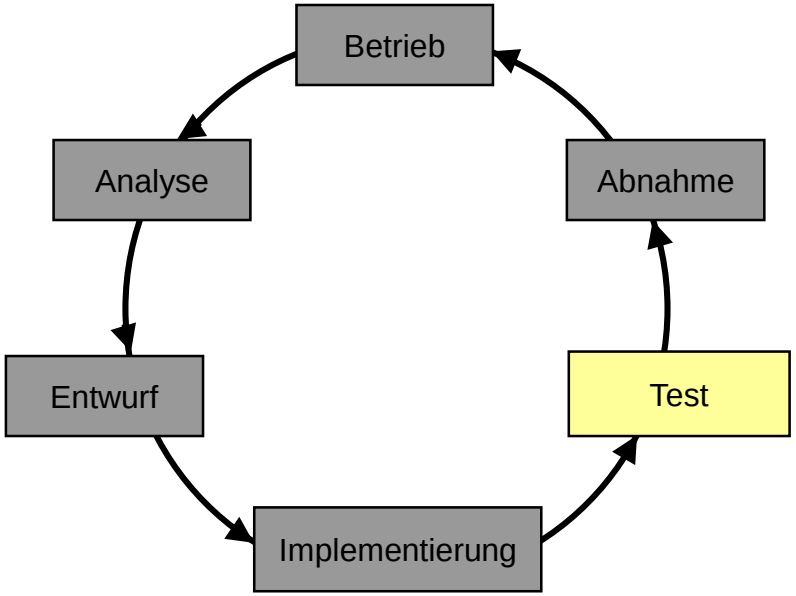
Initialisierung der bestellten Pizza mit Status & ID der Speisekarte Die ID wird erst beim Speichern gesetzt, liefert vorher UNDEF

**Parameter:**
*$pizzaID* PizzaID der Pizza in der Speisekarte
*$status* Bestell_Status der Pizza

**Rückgabe:**
Boolean true, wenn alles geklappt hat

# Hochschule Darmstadt
# Fachbereich Informatik

## 6.2.4 Arbeitstechniken im Test



# Testen in Webprojekten

- Üblicherweise findet Webentwicklung in vielen Iterationen statt
  - ⇨ Nach jeder Iteration muss getestet werden
  - ⇨ Automatisierte Tests sind die einzige Möglichkeit für einen systematischen Test
  - ⇨ Es gibt Tools
    - zur automatisierten Durchführung von Unit-Tests
    - zur Erzeugung von Dokumentationen der Testfälle
    - zur Berechnung der Fehlerüberdeckung
    - für Performance-Analysen und Profiling
    - für automatisierte GUI-Tests

# Unit-Tests mit PHP – Das Prinzip

> Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test instead.
>
> --Martin Fowler

- Für PHP gibt es Unit-Tests
  http://www.phpunit.de
  - ⇨ ähnlich zu CPP-Unit oder JUnit
  - ⇨ Festlegen der Tests
    - zu jeder Klasse wird eine Testklasse angelegt
      (die Testklasse implementiert Schnittstellen vom Testframework)
    - die Methoden der zu testenden Klasse werden in den Tests aufgerufen
    - es wird das erwartete "Soll-Ergebnis" festgelegt
    - bei Bedarf können vor und nach jedem Test Befehle ausgeführt werden
  - ⇨ Durchführung der Tests
    - ein einfacher Aufruf startet alle geschriebenen Tests
    - Tritt ein Unterschied zwischen Ist-Ergebnis und Soll-Ergebnis auf, erfolgt eine entsprechende Meldung
    - `phpunit tests_for_myfile.php` führt die Unit-Tests in der Datei tests_for_myfile.php aus

# PHPUnit: Eine Beispielklasse und eine Testklasse

```
class calc{
      public $l; // enthaelt den linken Operanden
      public $r; // enthaelt den rechten Operanden
      public function __construct($l,$r) // Konstruktor    {
        $this->l = $l;
        $this->r = $r;
      }
      ...
      public function sub() {
        return $this->l + $this->r;
}
```


```
class calcTest extends PHPUnit_Framework_TestCase {
      public function testSub() {
          $l = 5; $r = 3;
          $erg = $l - $r; // Ergebnis auf alternativem Weg berechnen
          $obj = new calc ($l, $r);
          $this->assertEquals($erg, $obj->sub());
      }
     ...
}
```


```
There was 1 failure:

1) testSub(calcTest)
Failed asserting that <integer:8> matches expected value <integer:2>.
C:\Links\ewa\Vorlesung\Links\PHP\90_calctest.php:16
```

# PHPUnit: Hilfsfunktionen

- Leere Testklasse automatisch anlegen:
  - ⇨ `phpunit --skeleton MeineKlasse`
    liest MeineKlasse.php ein und erzeugt MeineKlasseTest.php –
    für jede Methode wird ein (leerer) Testfall erzeugt
  - ⇨ neue PHPUnit-Version: `phpunit-skelgen --test MeineKlasse`
- Zugriff auf private (!) Attribute
  - ⇨
    ```
    $mytestclass->readAttribute
          ($object, 'NameOfPrivateAttribute');
    ```
    liefert den Wert des Attributs *NameOfPrivateAttribute*
  - ⇨ dadurch können Methoden getestet werden, ohne sich auf die Funktionsfähigkeit von anderen Methoden zu verlassen
- Zugriff auf private Methoden
  - ⇨ möglich durch Einführung einer neuen Klasse, die alle `private`-Methoden erbt und `public` macht. Für diese Klasse werden dann die Tests erstellt.

# PHPUnit: Dokumentation der Tests

- Die Namen der Testmethoden werden automatisch zu Text umformatiert
  - ⇨ Aus `testInitializePassesParametersToObject()` wird "Initialize passes parameters to object"
  - ⇨ HTML-Ausgabe
    `phpunit --testdox-html out.html tests_for_myfile.php`
  - ⇨ Bildschirm-Ausgabe
    `phpunit --testdox tests_for_myfile.php`

```
AngebotenePizza
 [x] Initialize passes parameters to object
 [x] Safe stores object in database
 [x] Delete removes entry from database
 [ ] Delete from database with invalid identifier
 [x] Get pizza id returns identifier
 [x] Get preis return preis
 [x] Get name returns name
 [x] Get bild returns name of bild
```

# PHPUnit: Anweisungsüberdeckung


- Während eines Testdurchlaufs werden die ausgeführten Programmteile markiert
  - ⇨ `phpunit --coverage-html ./coverage tests_for_myfile.php`
  - ⇨ führt die Unit-Tests in `tests_for_myfile.php` aus und erzeugt im Verzeichnis coverage einen HTML-Report der Anweisungsüberdeckung

D:\users\Ralf\Documents\Hahn\14SS\EWA\Vorlesung\Links\PHP / 90_calc.php

| | Code Coverage | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Classes and Traits | | | Functions and Methods | | | | Lines | | |
| Total | | 50.00% | 1 / 2 | | 75.00% | 3 / 4 | CRAP | | 91.67% | 11 / 12 |
| DivException | | 100.00% | 1 / 1 | | | | 0 | | | |
| calc | | 0.00% | 0 / 1 | | 75.00% | 3 / 4 | 6.02 | | 91.67% | 11 / 12 |
| __construct($l,$r) // Konstruktor | | | | | 100.00% | 1 / 1 | 1 | | 100.00% | 3 / 3 |
| sub() | | | | | 100.00% | 1 / 1 | 1 | | 100.00% | 1 / 1 |
| mul() | | | | | 0.00% | 0 / 1 | 2 | | 0.00% | 0 / 1 |
| div($throwException = false) | | | | | 100.00% | 1 / 1 | 3 | | 100.00% | 7 / 7 |

# PHPUnit: Anweisungsüberdeckung (II)


- Die Anweisungsüberdeckung zeigt an, welche Anweisung wie oft ausgeführt wurden
  - ⇨ nicht ausgeführte Teile werden rot markiert
  - ⇨ jede Anweisung sollte mindestens (!) einmal ausgeführt werden
  - ⇨ fehlende Tests sollten ergänzt werden
  - ⇨ 100% Anweisungsüberdeckung sind keine Garantie für Fehlerfreiheit

```
4  class calc
5  {
6      public $l; // enthaelt den linken Operanden
7      public $r; // enthaelt den rechten Operanden
8      public function __construct($l,$r) // Konstruktor
9      {
10         $this->l = $l;
11         $this->r = $r;
12     }
13
14     // gibt das
15     public func
16     { // Hier r
17         return
18     }
19
20     // Gibt das Ergebnis der Multiplikation zurueck
21     public function mul()
22     {
23         return $this->l * $this->r;
24     }
```

5 tests cover line 10

- calcTest::testSub
- calcTest::testDiv
- calcTest::testDivByZero
- calcTest::testDivByZeroThrowsException
- calcTest::testClassDivException

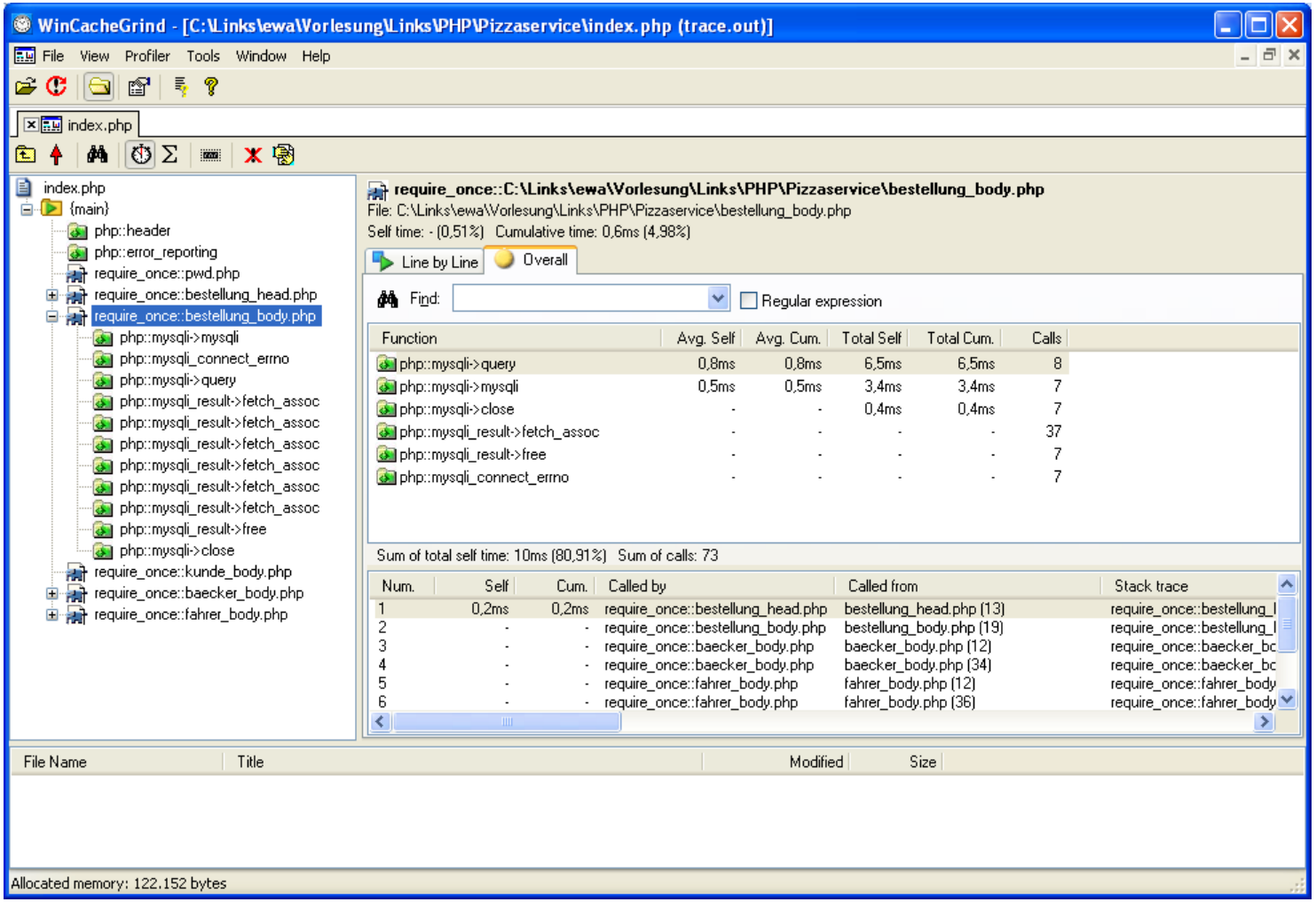
Der Konstruktor wurde 5 mal ausgeführt!

Die methode mul() wurde nicht ausgeführt! Es gibt keinen Test dafür.

# Profiling mit PHP

- Während eines PHP-Aufrufs kann die Ausführungsdauer und -häufigkeit der Programmteile protokolliert werden
  - ⇨ wenn PHP entsprechend konfiguriert ist (xdebug.profiler) wird eine "cachegrind"-Datei erzeugt
  - ⇨ `phpunit tests_for_myfile.php`
- Spezielle Viewer (z.B. wincachegrind) bereiten die Daten auf
  - ⇨ Profiling erlaubt gezielte Performance-Optimierung und
  - ⇨ erleichtert die Suche nach Speicherfehlern

# Profiling mit PHP



# GUI Tests mit Selenium

http://seleniumhq.org/projects/ide

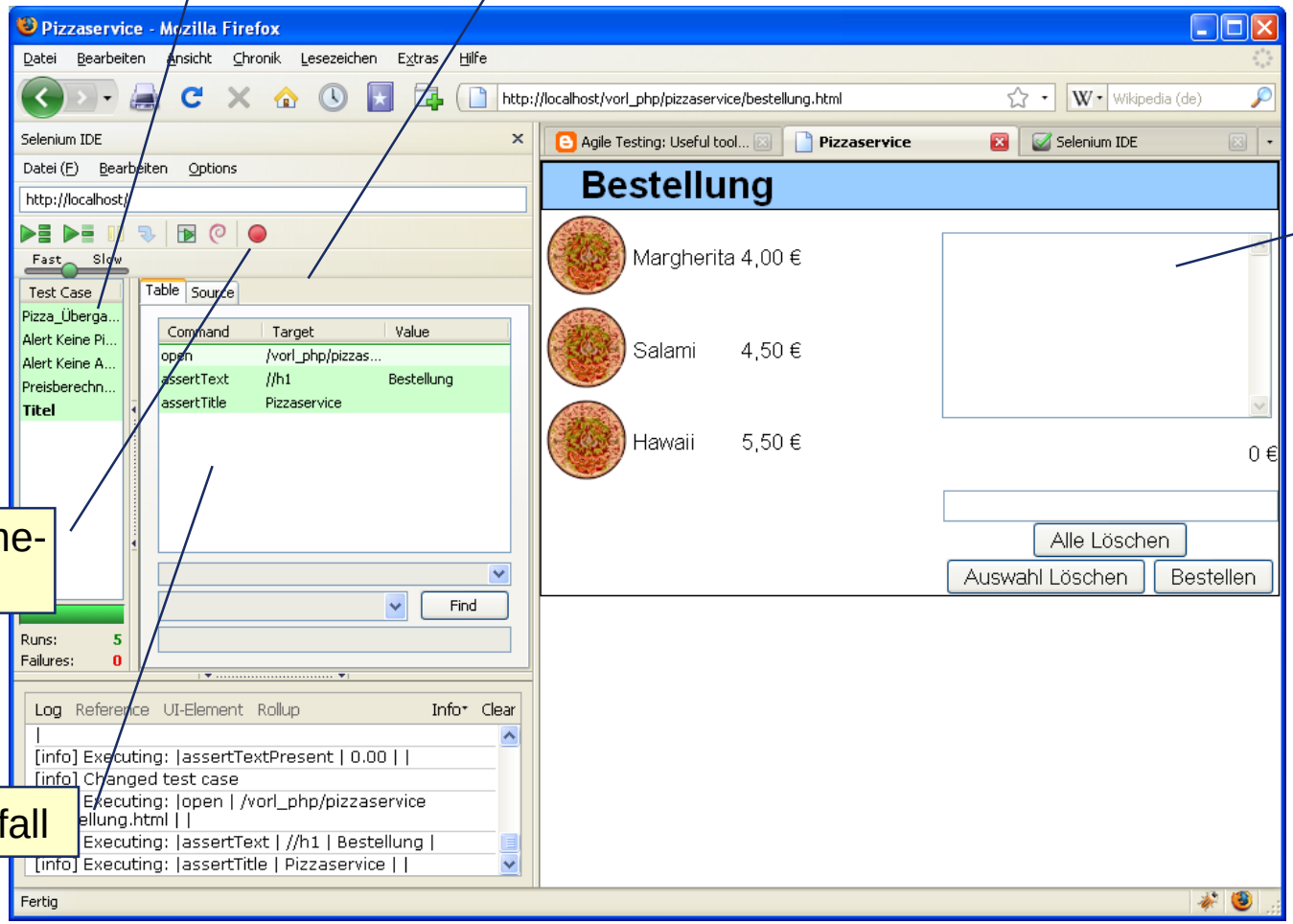
- Bisher haben wir "nur" Klassen mit UnitTests getestet
  - ⇨ aber wie testet man die gesamte Webanwendung incl. GUI?
- Automatisch ausführbare GUI-Tests wären praktisch
  - ⇨ Selenium IDE (Firefox Addon)
    - kann Mausklicks und Eingaben im Browser (intelligent) aufzeichnen und später wieder abspielen
    - dabei können Zusicherungen ("asserts") definiert und beim Abspielen überprüft werden: Prüfung von Texten, Inhalten, Formaten, Popups uvm.
    - Die Zuordnung zu Seitenelementen kann über die HTML-Id's erfolgen und ist dann unabhängig von der Positionierung
    - rechte Maustaste bietet in der Webseite sinnvolle Überprüfungen an
    - Vorsicht: Manchmal schlagen Tests fehl, wenn die Abfragen schneller ablaufen als die Kommunikation zwischen Webserver und Client. Die Geschwindigkeit muss dann korrigiert werden.

Selenium bietet eine geniale und einfache Möglichkeit automatisierbare GUI-Tests zu erstellen!

# Selenium IDE

Testfälle (Testsuite)

Selenium als Sidebar (Ansichtsoption)



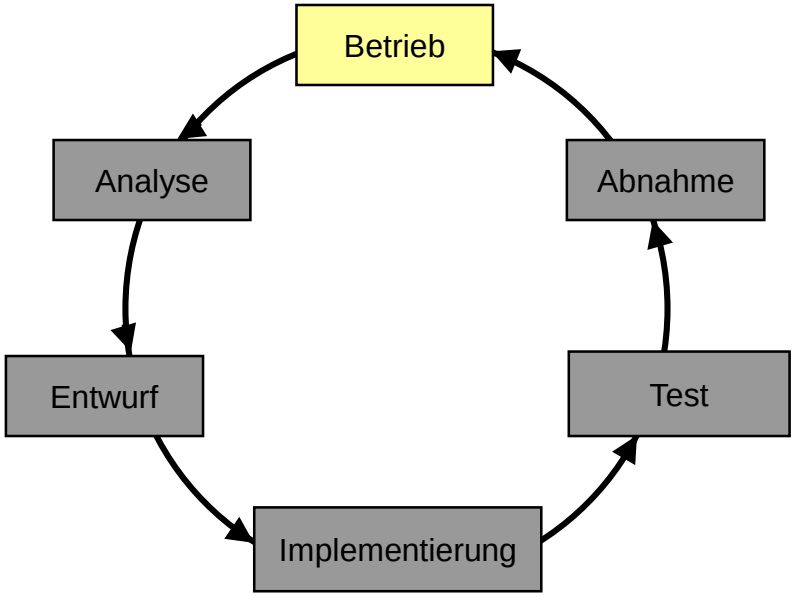

Webseite im Test

Aufnahme-knopf

Testfall

# Hochschule Darmstadt
# Fachbereich Informatik

## 6.2.5 Arbeitstechniken im Betrieb



# Fehlermeldungen beim Surfen

- Die Suche nach "Warning: Access denied for user" liefert über 1 Million Treffer bei Google
  - ⇨ die gefundenen Webseiten enthalten oft fehlerhaften PHP-Code
  - ⇨ die Fehlermeldungen werden nicht unterdrückt
  - ⇨ manche Meldungen enthalten User oder Passwort

**Warning**: mysql_connect() [function.mysql-connect]: Access denied for user 'thue4b1'@'lw-s09.sserv.de' (using password: YES) in **/www/thue/thue4b1/html/mysql.php** on line **3**

**Warning**: mysql_select_db() [function.mysql-select-db]: Access denied for user 'thue4b1'@'localhost' (using password: NO) in **/www/thue/thue4b1/html/mysql.php** on line **4**

**Warning**: mysql_select_db() [function.mysql-select-db]: A link to the server could not be established in **/www/thue/thue4b1/html/mysql.php** on line **4**

verbraucherzentrale Thüringen

**Warning**: mysql_connect() [function.mysql-connect]: Access denied for user 'thue4b1'@'lw-s09.sserv.de' (using password: YES) in **/www/thue/thue4b1/html/DBStuff.inc** on line **4**

**Warning**: mysql_select_db(): supplied argument is not a valid MySQL-Link resource in **/www/thue/thue4b1/html/DBStuff.inc** on line **5**

**Verbraucherzentrale Thüringen e.V.**

**Warning**: mysql_connect() [function.mysql-connect]: Access denied for user 'thue4b1'@'lw-s09.sserv.de' (using password: YES) in **/www/thue/thue4b1/html/mysql.php** on line **3**

**Warning**: mysql_select_db() [function.mysql-select-db]: Access denied for user 'thue4b1'@'localhost' (using password: NO) in **/www/thue/thue4b1/html/mysql.php** on line **4**

**Warning**: mysql_select_db() [function.mysql-select-db]: A link to the server could not be established in **/www/thue/thue4b1/html/mysql.php** on line **4**

**Warning**: mysql_query() [function.mysql-query]: Access denied for user 'thue4b1'@'localhost' (using password: NO) in **/www/thue/thue4b1**

**Warning**: mysql_query() [function.mysql-query]: Access denied for user 'thue4b1'@'localhost' (using password: NO) in **/www/thue/thue4b1/html/footer.php** on line **45**

**Warning**: mysql_query() [function.mysql-query]: A link to the server could not be established in **/www/thue/thue4b1/html/footer.php** on line **45**

# Entwicklungs- vs. Produktivumgebung

- Entwickler und Anwender haben eine unterschiedliche Einstellung zu Fehlermeldungen
  - ⇨ Für einen Entwickler sind Fehlermeldungen wichtige Informationen
  - ⇨ Für einen Anwender sind Fehlermeldungen bestenfalls ärgerliche oder unverständliche Texte
  - ⇨ In einer Entwicklungsumgebung sollten Fehlermeldungen sichtbar sein – in einer Produktivumgebung sollten sie "unsichtbar" protokolliert werden
- Alle Web-Systeme bieten die Möglichkeit Fehler wahlweise in eine Log-Datei zu schreiben oder auf den Bildschirm zu bringen
  - ⇨ z.B. im PHP-Code :

```
ini_set("display_errors", 0);          // 0=aus
ini_set("error_reporting", E_ALL);     // alle Meldungen
ini_set("log_errors", 1);              // Logging an
ini_set("error_log", "./mylog.txt");   // Logfile
```

  - ⇨ Immer gewissenhaft konfigurieren

# Web Page Analyzer

- Es gibt diverse Tools und Services im Internet um Webseiten zu analysieren
  - ⇨ z.B. http://websiteoptimization.com oder auch das Firefox Addon YSlow
  - ⇨ Liefert Analyse der abgerufenen Dateien (HTML, CSS, Grafiken etc.)
    - mit Anzahl und Größe der Dateien
    - mit geschätzten Downloadzeiten für verschiedene Geschwindigkeiten
    - Gibt Hinweise auf Performance Probleme und Optimierungsmöglichkeiten

Ergebnis von websiteoptimization.com für www.fbi.h-da.de 03/ 2017 (Auszug)

## External Objects

| External Object | QTY |
|---|---|
| Total HTML: | 1 |
| Total HTML Images: | 4 |
| Total CSS Images: | 39 |
| Total Images: | 43 |
| Total Scripts: | 15 |
| Total CSS imports: | 0 |
| Total Frames: | 0 |
| Total Iframes: | 0 |

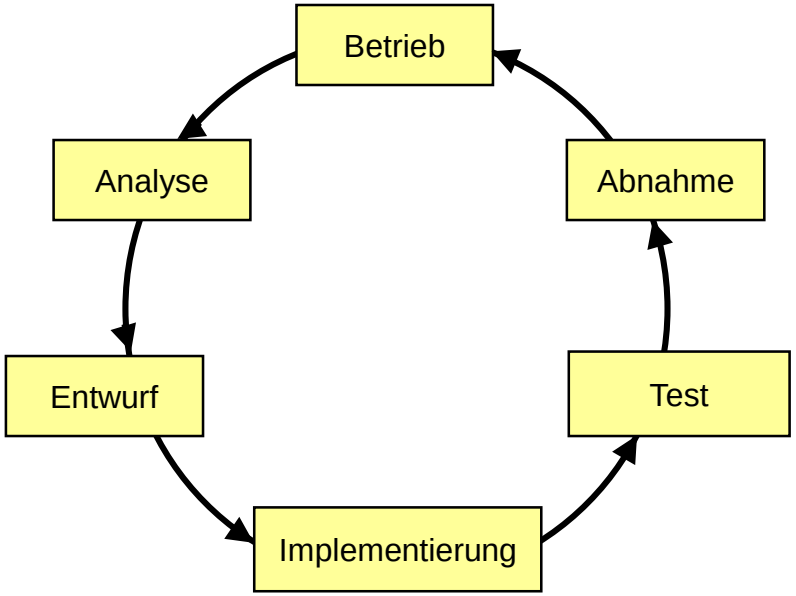
## Download Times*

| Connection Rate | Download Time |
|---|---|
| 14.4K | 1135.65 seconds |
| 28.8K | 573.72 seconds |
| 33.6K | 493.45 seconds |
| 56K | 300.79 seconds |
| ISDN 128K | 100.30 seconds |
| T1 1.44Mbps | 19.48 seconds |

TOTAL_IMAGES - Warning! The total number of images on this page is 43 , consider reducing this to a more reasonable number. Recommend combining, replacing, and optimizing your graphics. Replace graphic rollover menus with CSS rollover menus to speed display and minimize HTTP requests. Consider using CSS sprites to help consolidate decorative images. Use CSS techniques such as colored backgrounds, borders, or spacing instead of graphic techniques to reduce HTTP requests. Replace graphic text headers with CSS text headers to further reduce HTTP requests. Finally, consider optimizing parallel downloads by using different hostnames to reduce object overhead.

# Hochschule Darmstadt
# Fachbereich Informatik

## 6.2.6 Zusammenfassung



# Arbeitstechniken in der Webentwicklung

- Es wurden Vorgehensmodelle und Arbeitstechniken für die Webentwicklung vorgestellt
  - ⇨ Die Vorgehensmodelle erleichtern eine iterative und inkrementelle Entwicklung von Webanwendungen
  - ⇨ Auch wenn die meisten Techniken mit PHP demonstriert wurden – es gibt diese oder ähnliche Techniken auch für Ruby, Java etc.
  - ⇨ Die beschriebene Ansätze werden oft für große Systeme eingesetzt – sie sind aber durchaus auch in kleinen Projekten sinnvoll und einsetzbar (z.B. Unittests, Selenium Tests)

Auch für webbasierte Anwendungen können (und sollten) Sie alles anwenden, was Sie über Softwaretechnik, Software Ergonomie und GUIs gelernt haben!

# Hochschule Darmstadt
# Fachbereich Informatik

## 6.3 Content Management Systeme

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Webauftritte mit vielen Bearbeitern

- Viele Webauftritte werden von vielen Anwendern bearbeitet
  - ⇨ z.B. der Auftritt einer Hochschule http://www.fbi.h-da.de
  - ⇨ viele Anwender kennen weder HTML noch CSS & Co. (und wollen es auch nicht kennen lernen)
  - ⇨ dennoch soll
    - der Inhalt von Laien bearbeitet werden können
    - der Auftritt einheitlich aussehen
    - keine Gefahr bestehen, versehentlich das System zu beschädigen
- Ein (Web) Content Management System ("CMS") bietet diese (und weitere) Funktionalität
  - ⇨ Designer gestalten graphische Elemente und entwerfen Styles
  - ⇨ Programmierer implementieren Stylesheets und Sonderfunktionen
  - ⇨ Autoren schreiben Inhalte in vorgegebene Templates
  - ⇨ Bekannte Vertreter sind z.B. Joomla oder Typo3

# TYPO3

- Open Source Content Management System
  - ⇨ kostenlos
  - ⇨ zunehmende Verbreitung
  - ⇨ Einführung an der h_da im Juni 2005
- verfügbar für Unix, Linux, MacOS und Windows
  - ⇨ benötigt Apache oder IIS, PHP, MySQL
    (andere Datenbanken als Erweiterung)
- Bearbeitungsmodi:
  - ⇨ Frontend: rein Inhaltliche Änderung von bestehenden Seiten
  - ⇨ Backend: Änderungen an der Vernetzungsstruktur der Seiten
  - ⇨ Frontend und Backend über Standard-Browser zugänglich

# TYPO3: Frontend

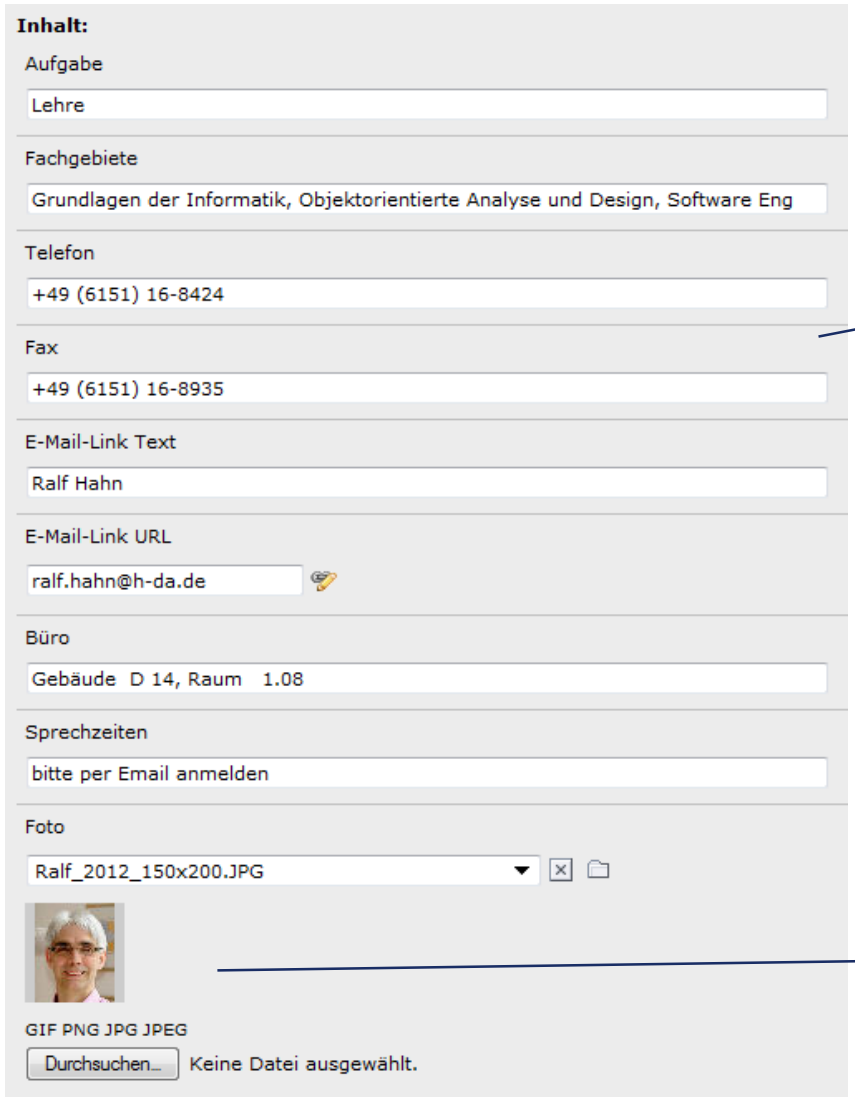
## Informationen zur Person

| | |
|---|---|
| **Aufgabe** | Lehre |
| **Fachgebiete** | Grundlagen der Informatik, Objektorientierte Analyse und Design, Software Engineering, Software Produktlinien |
| **Telefon** | +49 (6151) 16-8424 |
| **Fax** | +49 (6151) 16-8935 |
| **E-Mail** | Ralf Hahn |
| **Büro** | Gebäude D 14, Raum 1.08 |
| **Sprechzeiten** | bitte per Email anmelden |

Prof. Dr. Ralf Hahn

ruft Editor für dieses Element auf

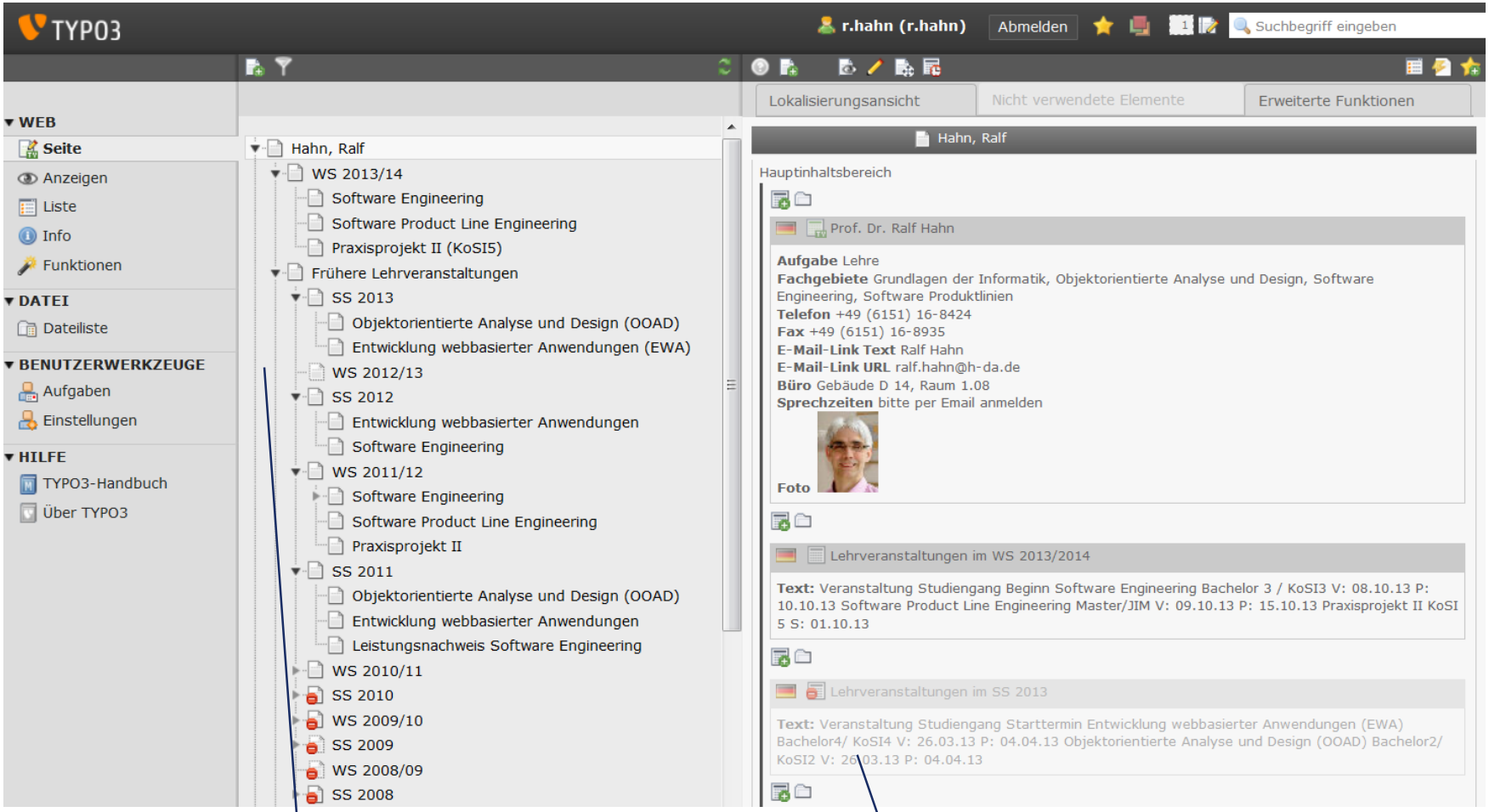
# TYPO3: Frontend II: Editor




Eingabemaske gibt Standard-elemente vor

Bild zuerst als Upload zur Verfügung stellen

# TYPO3: Backend




Vernetzungsstruktur der Seiten

Inhalte für verschiedene Abschnitte

# TYPO3: Ergebnis


automatisch erzeugt

integrierte Suchmaschine

generiert aus der Liste der Unterseiten

Layout und Design zentral gesteuert

# Hochschule Darmstadt
# Fachbereich Informatik

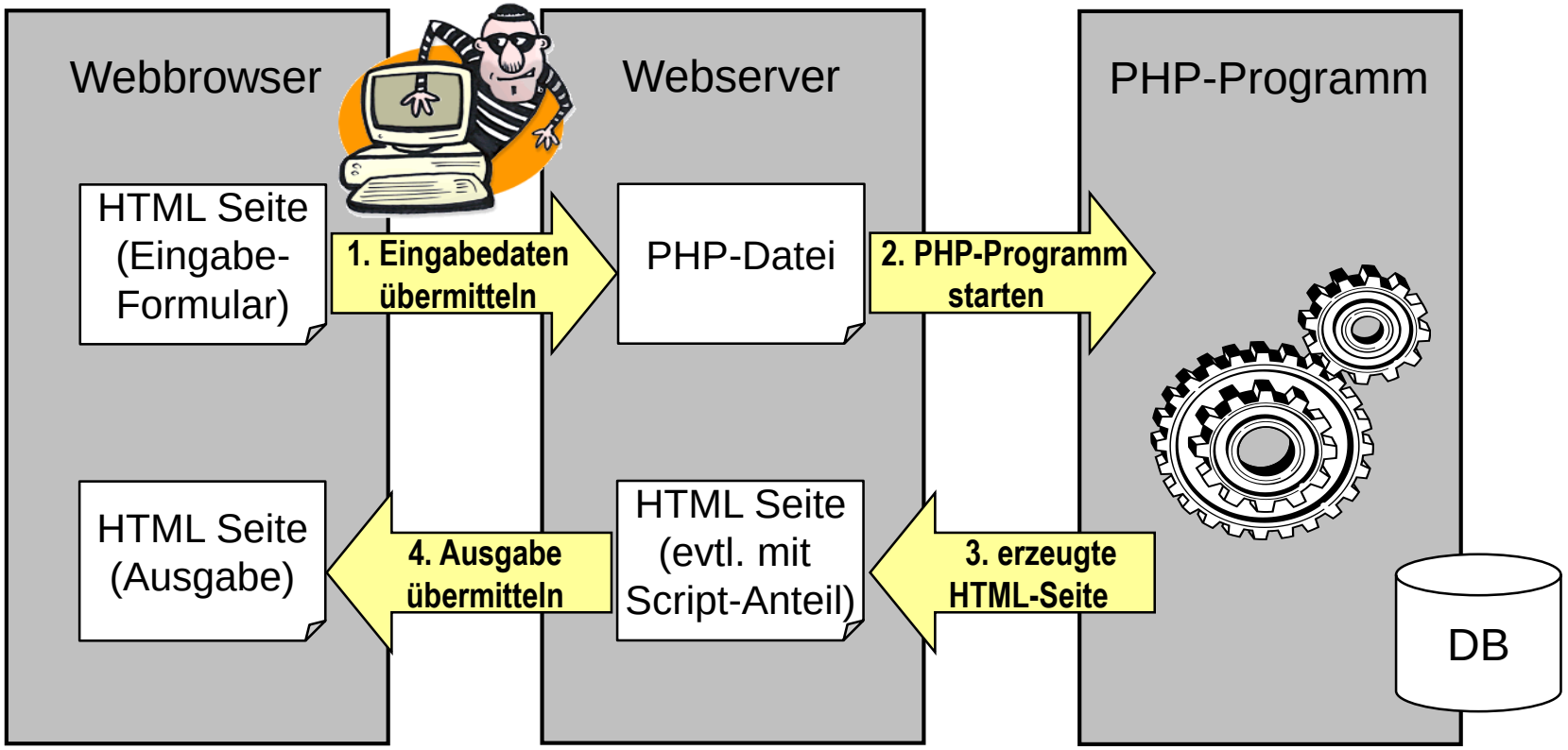
## 7. Zusammenfassung

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

# Behandelte Themen

1. Einleitung
   - 1.1 Softwaretechnik für webbasierte Anw.
   - 1.2 Ergonomie für webbasierte Anw.
2. Webclient
   - 2.1 HTML
   - 2.2 CSS
   - 2.3 ECMA Script
   - 2.4 AJAX
3. Webserver
   - 3.1 Webserver Software
   - 3.2 CGI
   - 3.3 PHP
4. Zwischen Webclient und Webserver
   - 4.1 HTTP
   - 4.2 Sessionverwaltung
5. Sicherheit
6. Professionelle Webentwicklung
   - 6.1 Vorgehensmodelle in der Webentw.
   - 6.2 Arbeitstechniken in der Webentw.
   - 6.3 Content Management Systeme
7. Zusammenfassung

# Einsatz der Technologien im Zusammenhang




- HTML
- CSS
- ECMA-Script
- DOM
- AJAX

- HTTP
- Sessions
- Server-Konfiguration

- CGI
- PHP
- MySQL

# Zielsetzung erreicht?

- aus der Modulbeschreibung:
  - ⇨ Die Studierenden sollen
    - ✔ Aktuelle Auszeichnungssprachen kennen und anwenden
    - ✔ Skriptsprachen für client- und serverseitige Webprogrammierung anwenden
    - ✔ ein Dokument Objekt Modell verstehen
    - ✔ die Architektur webbasierter Client/Server-Anwendungen mit Datenbankanbindung verstehen
    - ✔ Methoden und Techniken zur Entwicklung webbasierter Anwendung
    - ✔ Sicherheitsaspekte im Kontext von Webanwendungen verstehen
  - ⇨ Konkret: Nach der Veranstaltung...
    - kennen Sie den Sinn, Zweck und die Grenzen der verschiedenen Techniken
    - verstehen Sie das Zusammenspiel der verschiedenen Techniken
    - kennen Sie die wesentlichen Standards
    - sind Sie in der Lage, komplexe und standardkonforme Webseiten zu erstellen
    - haben Sie die Grundlagen, um sich in diverse andere Web-Techniken einzuarbeiten