

S1 Einführung in Web-bezogene Sprachen

Wofür benötigt man Web-bezogene Sprachen?

Gestaltung von Web-Dokumenten:

Beschreibung der Struktur, der Hypertext-Verweise, der Präsentation durch Annotationen (mark-up)

Sprachen: HTML, XML

Programmierung von Web-Diensten:

Erzeugung individueller Web-Seiten, Zugriff auf Dateien und Datenbanken, interaktive Elemente, wie Formulare, auf Web-Seiten

Script-Sprachen: spezielle Programmiersprachen für solche Zwecke
PHP, JavaScript, Perl, VBScript, ASP, SQL

Mit passenden Hilfsmitteln werden auch allgemeine Programmiersprachen eingesetzt, wie Java, C++

beide Zwecke hängen zusammen - Sprachen werden integriert:

Web-Seiten (in HTML) enthalten Programme (in PHP),
Programme (in PHP) erzeugen Web-Seiten (in HTML)

In dieser Vorlesung wird in die Benutzung von **HTML** und **PHP** eingeführt.
JavaScript und XML werden kurz gezeigt.

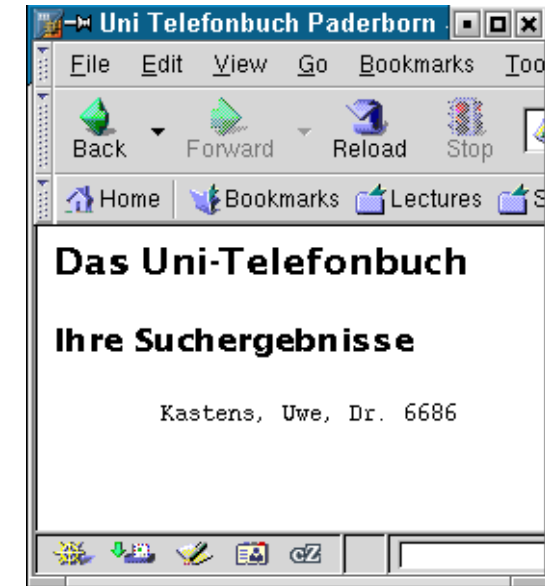
Beispiel für eine dynamische, interaktive Web-Seite Telefonverzeichnis



Anklicken der URL
fordert ein Formular an

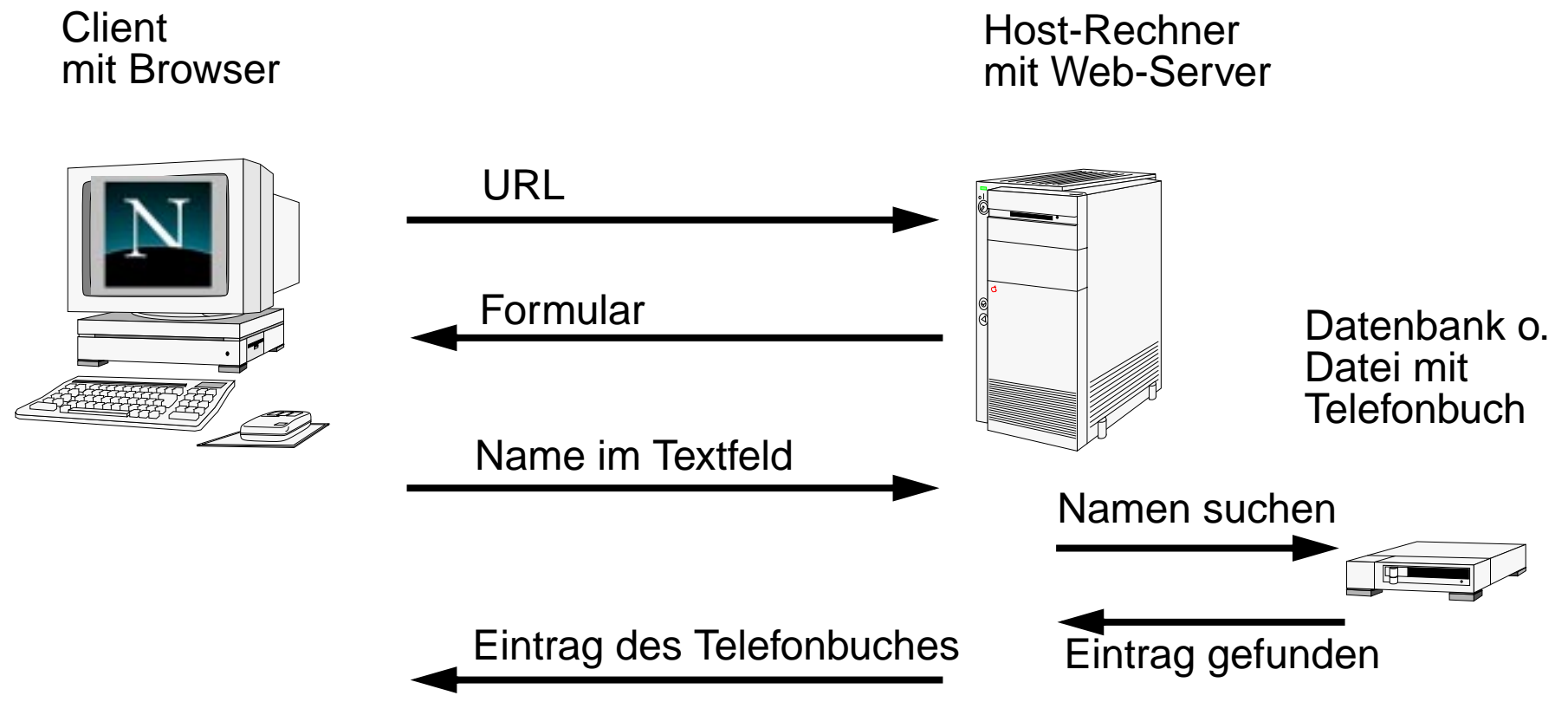


Eintragen einer Anfrage
fordert eine Suche im
Telefonbuch an

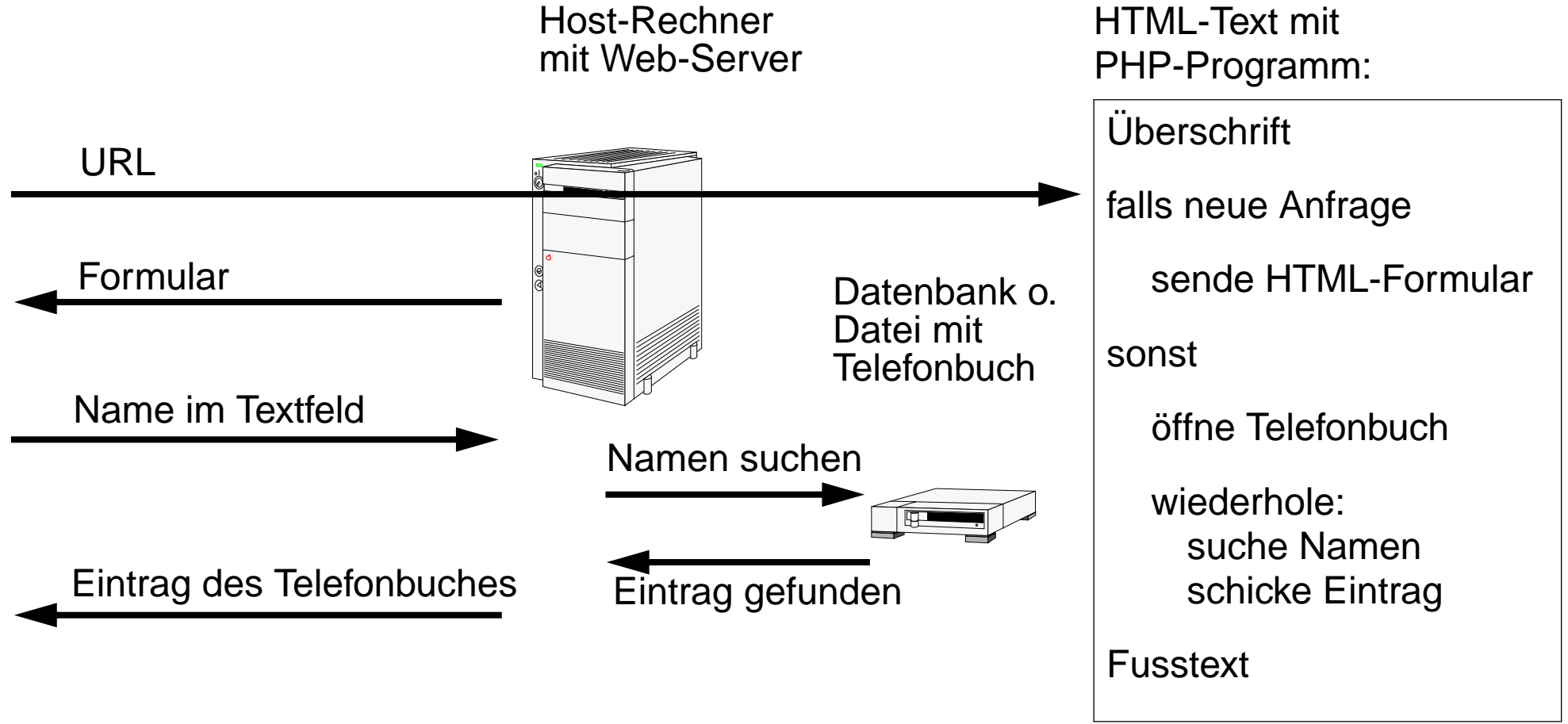


Ergebnis der Suche
wird angezeigt

Interaktion zwischen Client und Server



Interaktion mit PHP-Programm im HTML-Text



Struktur des PHP-Programms im HTML-Text

```

<html>
<head>
<title>Uni Telefonbuch Paderborn </title>
</head>
<body>
<h3>Das Uni-Telefonbuch</h3>
<?php // Typischer Aufbau einer PHP-Seite:
if (!isset($_REQUEST['wunsch'])) {
    echo <<<FORMULARANZEIGE
        <h4>Wen suchen Sie?</h4>
        <form action="http://ag-kastens/.../unitel.php"
            method="POST">
        <p><input type="text" name="wunsch">
        <p><input type="submit" value="Abschick">
        </form>
        FORMULARANZEIGE;
}
else {
    $wunsch = $_REQUEST['wunsch'];
    echo "<h4>Ihre Suchergebnisse</h4><p>\n<pre>\n";
    $fp = fopen("telefonbuch.txt" ,"r");
    while (!feof($fp)) {
        $line = fgets($fp, 64);
        if (preg_match("/$wunsch/i", $line)) {
            echo "\t$line";
        }
    }
    fclose($fp);
}
?>
</pre>
</body></html>

```

HTML-Text mit
PHP-Programm:

Überschrift

falls neue Anfrage

sende HTML-Formular

sonst

öffne Telefonbuch

wiederhole:

suche Namen

schicke Eintrag

Fusstext

Ein erster Eindruck von HTML

```
<html>  
<head>  
<title>Uni Telefonbuch Paderborn </title>  
</head>  
<body>  
<h3>Das Uni-Telefonbuch</h3>
```

Das Dokument und seine **Textelemente werden annotiert** mit sogenannten **Tags**, z. B.

```
<title> ... </title>  
<h3> ... </h3>
```

Die Tags haben Namen und **kennzeichnen Anfang und Ende** einer Struktur.

Strukturen können **geschachtelt** sein.

So wird die **Struktur** des Dokumentes und seine **Darstellung** beschrieben.

```
</pre>  
</body></html>
```

Ein erster Eindruck von PHP

```
if (!isset($_REQUEST['wunsch'])) {
    echo <<<FORMULARANZEIGE
        <h4>Wen suchen Sie?</h4>
        <form action="http://ag-kastens/.../unitel.php"
            method="POST">
        <p><input type=text name=wunsch>
        <p><input type=submit value=Abschick>
        </form>
    FORMULARANZEIGE;
}
else {
    $wunsch = $_REQUEST['wunsch'];
    echo "<h4>Ihre Suchergebnisse</h4><p>\n<pre>\n";
    $fp = fopen("telefonbuch.txt" ,"r");
    while (!feof($fp)) {
        $line = fgets($fp, 64);
        if (preg_match("/$wunsch/i", $line)) {
            echo "\t$line";
        }
    }
    fclose($fp);
}
```

PHP ist eine Programmiersprache mit

- **Ablaufstrukturen** wie bedingten Anweisungen und Schleifen:

```
if (...) { .. } else { .. }
```

```
while (...) { ... }
```

- **Variablen, Zuweisungen und Funktionsaufrufen:**

```
$line = fgets ($fp, 64);
```

Das Programm wird vom Web-Server ausgeführt. Die Ausgabe wird auf dem Browser des Client angezeigt.

Integration von PHP-Programm und HTML-Text

```
<html>
<head>
<title>Uni Telefonbuch Paderborn </title>
</head>
<body>
<h3>Das Uni-Telefonbuch</h3>
<?php // Typischer Aufbau einer PHP-Seite:
if (!isset($_REQUEST['wunsch'])) {
    echo <<<FORMULARANZEIGE
        <h4>Wen suchen Sie?</h4>
        <form action="http://ag-kastens/../../unitel.php"
            method="POST">
            <p><input type=text name=wunsch>
            <p><input type=submit value=Abschick>
        </form>
        FORMULARANZEIGE;
}
else {
    $wunsch = $_REQUEST['wunsch'];
    echo "<h4>Ihre Suchergebnisse</h4><p>\n<pre>\n";
    $fp = fopen("telefonbuch.txt" ,"r");
    while (!feof($fp)) {
        $line = fgets($fp, 64);
        if (preg_match("/$wunsch/i", $line)) {
            echo "\t$line";
        }
    }
    fclose($fp);
}
?>
</pre>
</body></html>
```

PHP-Programm in HTML-Dokument eingebettet, geklammert durch

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

HTML-Text wird vom PHP-Programm ausgegeben, durch echo-Anweisung:

```
echo "<h4>...</h4>";
```

E1. Einführung zu Eigenschaften von Sprachen

Sprachen in der Informatik werden

- für bestimmte **Zwecke** geschaffen
hier: **Auszeichnungssprachen** (HTML) und
Skriptsprachen (PHP, Javascript)
weitere Aufgabengebiete für Sprachen in dieser Einführung;
- je nach Zweck und **Niveau** mit einfachen oder komplexen,
wenigen oder zahlreichen **Sprachkonstrukten** ausgestattet;
- durch **Regeln formal oder informell definiert**; sie legen fest:
Notation der Symbole (Lexeme),
Struktur der Sätze (Syntax),
Bedeutung der Konstrukte (Semantik);
- durch Software-Werkzeuge **übersetzt** oder **interpretiert**

Alle diese Aspekte beeinflussen die **Eigenschaften der Sprachen**.

Die **Verbreitung der Sprachen** wird auch beeinflusst durch

- Erlernbarkeit und Handhabbarkeit,
- Verfügbarkeit von Werkzeugen,
- Marktmechanismen

4 Ebenen der Spracheigenschaften

Ein **Satz einer textuellen* Sprache** ist eine **Folge von Zeichen** eines zu Grunde liegenden **Alphabetes**

Beispiel: ein PHP-Programm ist ein Satz der Sprache PHP;
hier ein Ausschnitt daraus:

```
$line = fgets ($fp, 64);
```

Die **Struktur eines Satzes** wird in 2 Ebenen definiert:

1. **Notation von Grundsymbolen (Lexemen, token)**
2. **Syntaktische Struktur**

Die **Bedeutung eines Satzes** wird in 2 weiteren Ebenen an Hand der Struktur für jedes Sprachkonstrukt definiert:

3. **statische Semantik**

Eigenschaften, die vor der Ausführung bestimmbar sind.

4. **dynamische Semantik**

Eigenschaften, die erst während der Ausführung bestimmbar sind.

Auf jeder der 4 Ebenen gibt es auch Regeln, die korrekte Sätze erfüllen müssen.

*) Es gibt auch **visuelle Sprachen**. Ihre Sätze werden aus graphischen Symbolen zusammengesetzt.

Ebene 1: Notation von Grundsymbolen

Ein **Grundsymbol** wird aus einer **Folge von Zeichen des Alphabetes** gebildet. Die Regeln zur Notation von Grundsymbolen werden z. B. durch **reguläre Ausdrücke formal definiert** (siehe E2).

```
$line = fgets ($fp, 64);
```

Typische Grundsymbole in Programmiersprachen

4 Symbolklassen:

Beispiele aus PHP

Bezeichner (identifizier)

`$line` `fgets`

Namen für Variable, Funktionen, ...

Literale (literals)

`64` `"telefonbuch.txt"`

Zahlwerte, Zeichenreihen

Wortsymbole (keywords)

`while` `if`

kennzeichnen Sprachkonstrukte

Spezialzeichen

`<=` `=` `;` `{` `}`

Operatoren, Separatoren

Zwischenräume, Tabulatoren, Zeilenwechsel und **Kommentare** zwischen den Grundsymbolen dienen der Lesbarkeit und sind sonst bedeutungslos

Kommentar

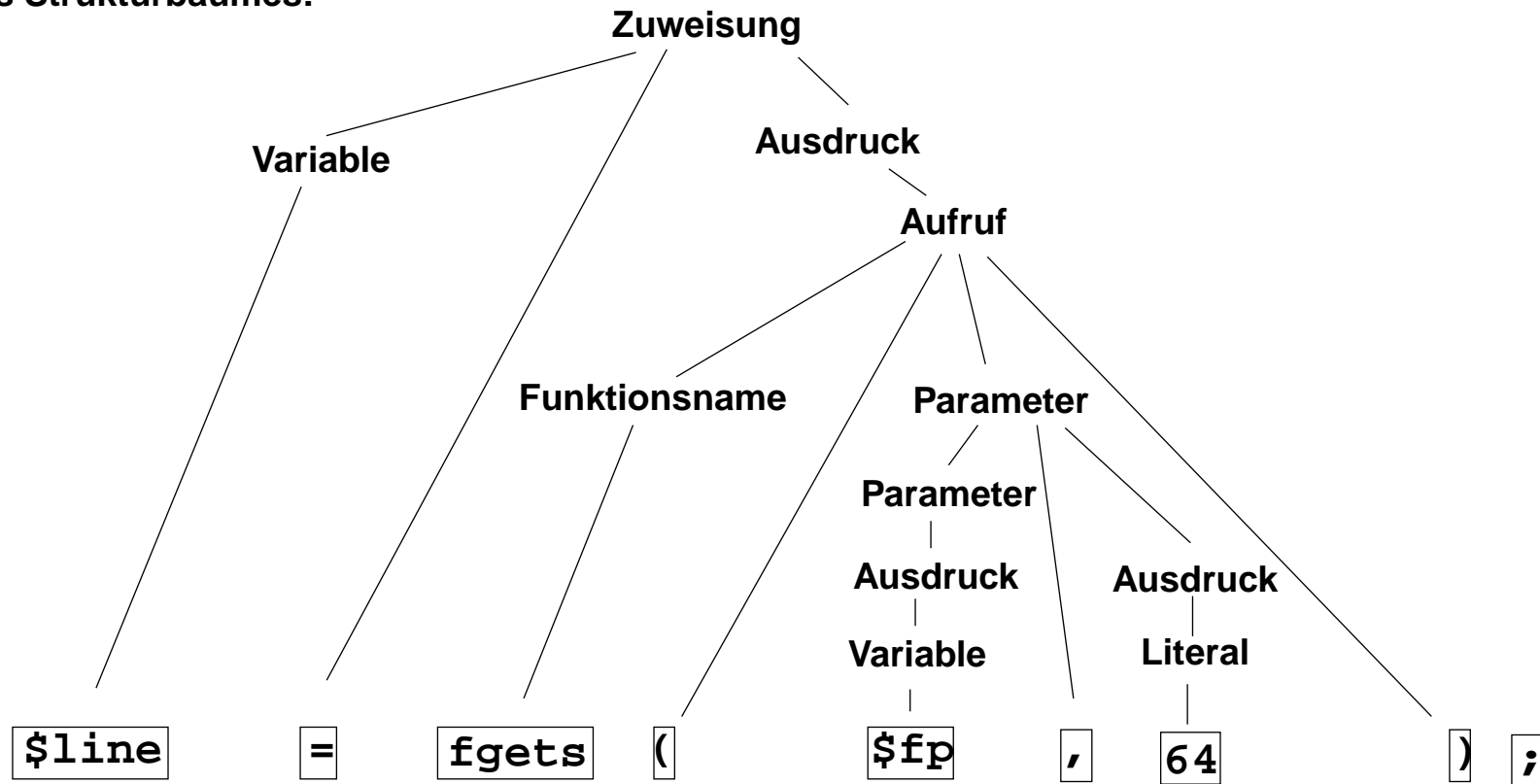
`/* Kommentar in C-Notation */`

Ebene 2: Syntaktische Struktur

Ein Satz einer Sprache wird **in seine Sprachkonstrukte** gegliedert. Sie sind meist ineinander **geschachtelt**. Diese **syntaktische Struktur** wird durch einen **Strukturbaum** dargestellt. Die **Grundsymbole sind Blätter** in diesem Baum.

Die Syntax einer Sprache wird durch eine **kontextfreie Grammatik präzise definiert**. Die Grundsymbole sind die **Terminalsymbole** der Grammatik (siehe E2).

Teil des Strukturbaumes:



Ebene 3: Statische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Bedeutung (Semantik) beschreiben, soweit sie **an der Programmstruktur festgestellt** werden können (**statisch**), ohne das Programm auszuführen.

Typische Eigenschaften der statischen Semantik (für **übersetzte Sprachen**):

- **Bindung von Namen:**

Regeln, die einer **Anwendung** eines Namens seine **Definition** zuordnen.

z. B. „Zu dem Funktionsnamen in einem Aufruf muss es eine Funktionsdefinition mit gleichem Namen geben.“

- **Typregeln:**

Sprachkonstrukte wie **Ausdrücke** und **Variable** liefern bei ihrer Auswertung einen **Wert eines bestimmten Typs**. Er muss im Kontext zulässig sein und kann die Bedeutung von Operationen näher bestimmen.

z. B. „Die Operanden des + Operators müssen Zahlwerte sein.“

5 + "Text" ist in vielen Sprachen ein Typfehler

In der Sprache **PHP** gehören

die **Bindungsregeln** zur **statischen Semantik**,

die **Typregeln** aber zur **dynamischen Semantik**, da sie erst bei der Ausführung des Programms angewandt werden können.

Ebene 4: Dynamische Semantik

Eigenschaften von Sprachkonstrukten, die ihre Wirkung beschreiben und **erst bei der Ausführung (dynamisch) bestimmt oder geprüft** werden können.

Typische Regeln der **dynamischen Semantik** beschreiben

- welche **Voraussetzungen für eine korrekte Ausführung** eines Sprachkonstruktes erfüllt sein müssen, z. B.
„Ein numerischer Index einer Array-Indizierung, wie in `$var[$i]`, darf nicht kleiner als 0 sein.“
- welchen **Effekt die Ausführung** eines Sprachkonstruktes verursacht, z. B.
„Eine Zuweisung der Form *Variable = Ausdruck* wird wie folgt ausgewertet:
Die Speicherstelle der Variablen auf der linken Seite wird bestimmt.
Der Ausdruck auf der rechten Seite wird ausgewertet.
Das Ergebnis ersetzt dann den Wert an der Stelle der Variablen.“

In der Sprache PHP gehören auch die Typregeln zur dynamischen Semantik.

Übersetzung von Sprachen

Ein **Übersetzer** transformiert jeden korrekten Satz (Programm) der **Quellsprache** in einen **gleichbedeutenden Satz der Zielsprache**.

- Die meisten **Programmiersprachen zur Software-Entwicklung** werden übersetzt, z. B. C, C++, Java, Ada, Modula
- Zielsprache ist dabei meist eine **Maschinensprache** eines realen Prozessors oder einer abstrakten Maschine.
- Übersetzte Sprachen haben eine **stark ausgeprägte statische Semantik**.
- Der Übersetzer prüft die Regeln der statischen Semantik, wie Bindungs- und Typregeln; er findet viele Arten von **Fehlern vor der Ausführung**.

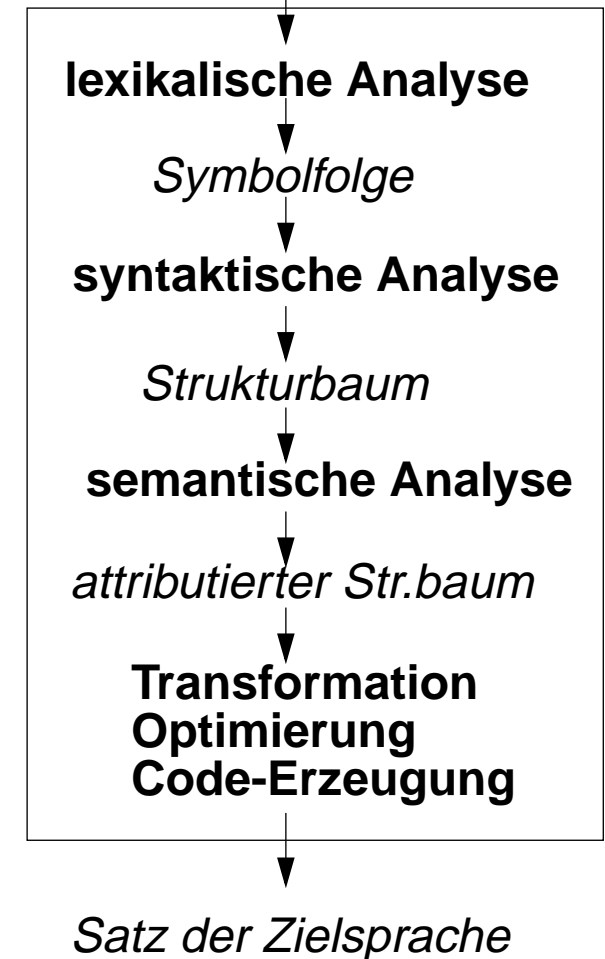
Es gibt auch **Übersetzer für andere Sprachen**:

Textformatierung: LaTeX -> PostScript

Spezifikationssprachen: UML -> Java

Übersetzer und seine Phasen

Satz der Quellsprache



Interpretation von Sprachen

Ein **Interpreter** liest einen Satz (Programm) einer Sprache und führt ihn aus.

Sprachen, die so **strikt interpretiert** werden:

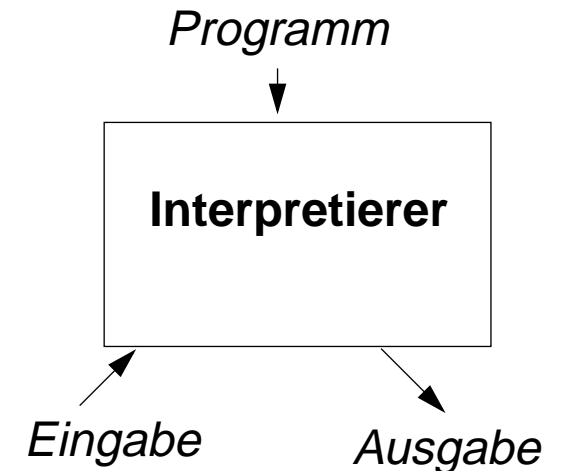
- haben **einfache Struktur** und **keine statische Semantik**,
- Bindungs- und Typregeln werden erst bei der Ausführung geprüft,
- nicht ausgeführte Programmteile bleiben ungeprüft,

z. B. Lisp, Prolog

Manche Interpreter erzeugen vor der Ausführung eine **interne Repräsentation des Satzes**;
dann können auch Struktur und Regeln der statischen Semantik vor der Ausführung geprüft werden,
z. B. **Skriptsprachen** PHP, JavaScript, Perl

Interpreter können **auf jedem Rechner verfügbar** gemacht werden und in andere Software (Browser) **integriert** werden.

Interpretation kann 10-100 mal **zeitaufwändiger** sein als die Ausführung von übersetztem Maschinencode.



Zwecke von Sprachen: allgemeine Software-Entwicklung

Anforderungen:

- Algorithmen klar und effizient formulieren
- komplexe Datenstrukturen klar und effizient formulieren
- prüfbare Regeln (statische Semantik) einhalten, dadurch Fehler reduzieren
- modulare Gliederung großer Programme mit expliziten Schnittstellen
- Schnittstellenprüfung beim Zusammensetzen von Bibliothekskomponenten

Konsequenzen:

- umfangreiche, komplexe Sprachen: viele Konstrukte, viele Regeln
- relativ hoher Schreibaufwand, durch explizite, redundante Angaben

Nutzen

hoch bei großen Software-Systemen
recht umständlich für kleine, einfache Aufgaben

Sprachstile: imperativ, objektorientiert, (funktional)

Sprachen: Modula-2, Ada, C++, Eiffel, Java, (SML)

Zwecke von Sprachen: Skriptsprachen

Scripting:

Zusammensetzen von Kommandos zu einem wiederverwendbaren „Drehbuch“

Anforderungen:

- kleine, einfache Aufgaben lösen ohne komplexe Algorithmen und Datenstrukturen
- existierende Funktionen nutzen und verknüpfen
- Verzicht auf Sicherheit durch prüfbare Regeln zugunsten kürzerer Programme
- Textverarbeitung und Ein- und Ausgabe sind wichtig
- gute Verfügbarkeit und Handhabbarkeit

Konsequenzen:

- einfache Sprachen: wenige Konstrukte, wenige Regeln, kurze Programme
- dynamische Typprüfung
- interpretiert, d. h. ohne Übersetzung ausführbar

Herkunft: Kommandosprachen von Betriebssystemen, JCL, Unix Shell

Sprachstile: imperativ, objektorientiert

Sprachen: PHP, Perl, JavaScript, Python

Zwecke von Sprachen: Auszeichnungssprachen

Auszeichnungssprache: Markup language

Anforderungen:

- Struktur von Dokumenten beschreiben: Überschriften, Absätze, Listen, Tabellen
- hierarchische Gliederung, auch Hypertext-Verweise
- Darstellung der Strukturen beschreiben - aber abtrennbar
- von Menschen schreib- und lesbar
- keine Programmierung - aber integrierbar

Konsequenzen:

- Strukturelemente werden mit lesbaren Markierungen gekennzeichnet (markup)
- geklammerte, geschachtelte Strukturen
- Menge und Bedeutung der Markierungen festlegbar (SGML, XML)
- Darstellung getrennt beschreibbar (HTML + CSS)

Sprachen: SGML, HTML, XML

Sprachen für weitere Zwecke

Zugriff auf Datenbanken

Selektionen aus Relationen, Verknüpfungen
übersetzt in Datenbankzugriffe

SQL

Spezifikation von Hardware-Bausteinen

VHDL

Spezifikation von Software-Komponenten

UML

allgemeine Spezifikation

SETL, Z

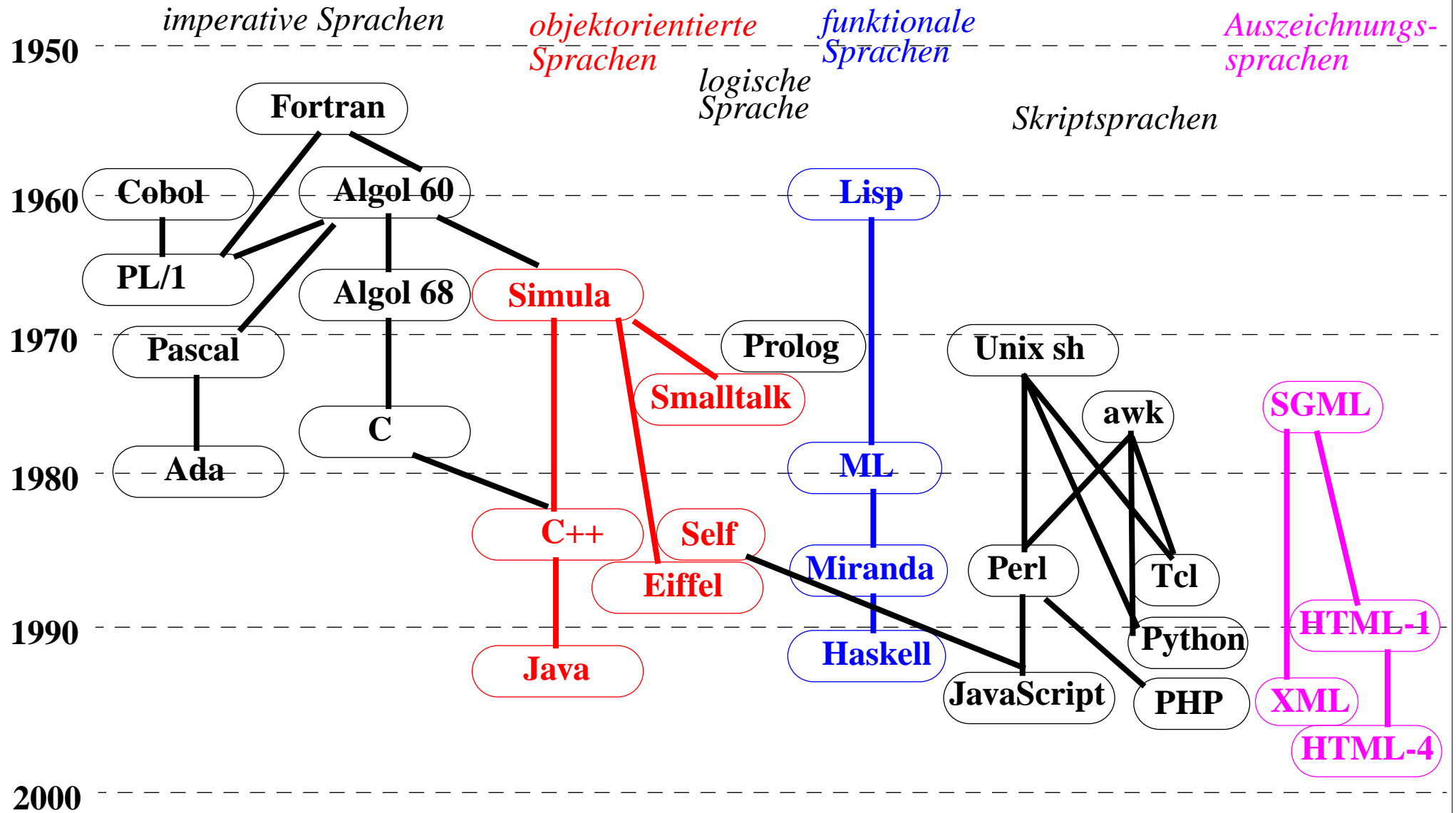
Spezifikation von Grammatiken

EBNF

Textsatz

TeX, LaTeX, PostScript

Entstehungszeit und Verwandtschaft wichtiger Sprachen



nach [D. A. Watt: Programmiersprachen, Hanser, 1996; Seite 5] und [Computer Language History <http://www.levenez.com/lang>]

S2 HTML

HTML steht für **Hypertext Markup Language**: eine **Auszeichnungssprache**, in der man **Dokumente** durch **Hyperlinks (Verweise)** miteinander **verknüpfen** kann.

Auszeichnungen (Markups) sind typografische Anweisungen für

- die Strukturierung von Texten in Abschnitte, Absätze, Listen, Tabellen, usw.
- die Gestaltung von Zeichen: Schrifttyp, Schriftgrad, Schriftstil, Schriftfarbe
- die Bildmontage: Platzierung und Größe
- das Layout des Dokumentes: Tabellen und Frames
- die Verknüpfung von Dokumenten: Anker, Links, sensitive Bildbereiche
- die Dialoggestaltung: Formulare, Schaltelemente

HTML-Markups oder **Tags** haben die Form `<table> ... </table>`
Sie treten meist als **Klammern von Anfangs- und Ende-Tag** auf.

Tags können zusätzlich **Attribute** haben,

```
<table border="3" bgcolor="#2332FF">
```

HTML wurde ursprünglich aus der **Meta-Sprache SGML** abgeleitet.

HTML wird nun auch aus der **Meta-Sprache XML** abgeleitet: XHTML.

S2.1 Auszeichnung von Strukturen und Texten

```
<!DOCTYPE HTML PUBLIC
  "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Hallo Welt</title>
  </head>
  <body>
    <h1>Hello World!</h1>
  </body>
</html>
```



Grundstruktur jeder HTML-Datei:

<!DOCTYPE ...>:
Kommentar mit optionaler Angabe der HTML Definition

<html>:
äußere Klammer

<head>:
Angaben über das Dokument

<body>:
Inhalt des Dokumentes

Schreibweise von Tags und Attributen

Tags haben die Form

`<TagName>`

Anfangs-Tag

`</TagName>`

Ende-Tag

Zwischen `<` und `TagName` dürfen keine Zwischenräume stehen.

Tags, die **Strukturen** kennzeichnen, sollen **immer klammernd** geschrieben werden, auch wenn das Ende-Tag optional ist.

Tags können **Attribute** haben.

Sie ordnen dem gekennzeichneten Bereich bestimmte Eigenschaften zu.

Jedes Attribut hat die Form

`Name="Wert"`

z. B. `border="4"`

Das Anfangs-Tag kann zwischen dem Tag-Namen und dem `>` eine Folge von Attributen enthalten

`<table border="4" frame="box">`

Verwenden Sie **nur notwendige Attribute**. Viele Attribute sind veraltet oder werden von verschiedenen Browsern unterschiedlich behandelt.

Detaillierte Angaben zur Darstellung werden besser mit anderen Mitteln (z. B. Style Sheets, siehe Folie 2.32) gemacht als mit Attributen.

Überschriften und Absätze

```

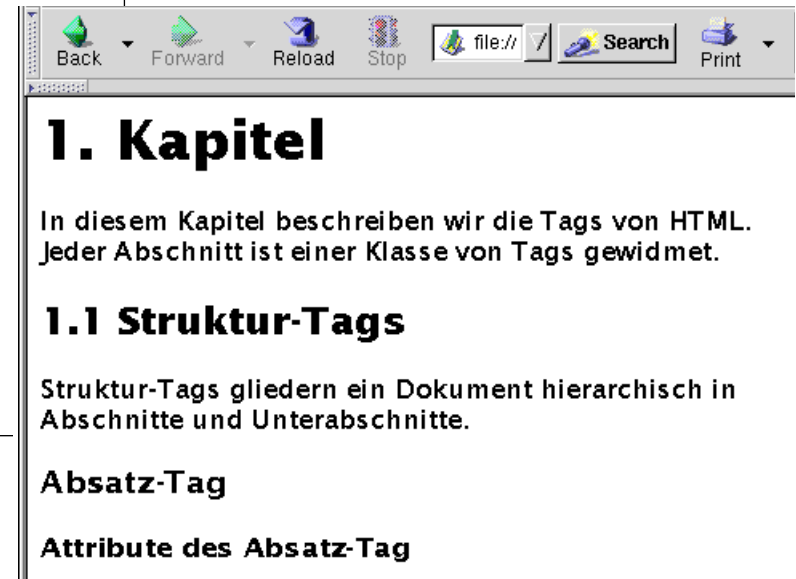
<html><head>
  <title>&Uuml;berschriften</title>
</head>
<body>
  <h1>1. Kapitel</h1>
  <p> In diesem Kapitel beschreiben wir
    die Tags von HTML.
    <br>Jeder Abschnitt ist einer Klasse
    von Tags gewidmet.
  </p>
  <h2>1.1 Struktur-Tags</h2>
  <p>Struktur-Tags gliedern ein
    Dokument hierarchisch in
    Abschnitte und Unterabschnitte.
  </p>
  <h3>Absatz-Tag</h3>
  <h4>Attribute des Absatz-Tag
  </h4>

```

<h1>, <h2>, ..., <h6>:
Überschriften fallenden Ranges

<p>:
Jeder Absatz wird explizit ausgezeichnet.

**
:**
erzwingt Zeilenwechsel.



Aufzählungen

```

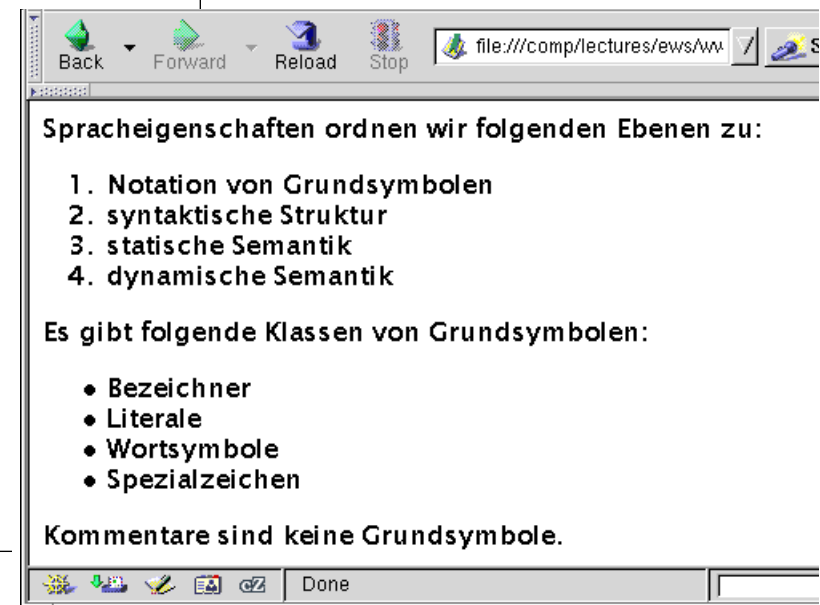
<html><head>
  <title>Aufzählungen</title>
</head><body>
  <p> Spracheigenschaften ordnen wir
    folgenden Ebenen zu:
    <ol>
      <li>Notation von Grundsymbolen</li>
      <li>syntaktische Struktur</li>
      <li>statische Semantik</li>
      <li>dynamische Semantik</li>
    </ol>
    Grundsymbolklassen sind:
    <ul>
      <li>Bezeichner</li>
      <li>Literale</li>
      <li>Wortsymbole</li>
      <li>Spezialzeichen</li>
    </ul>
    Kommentare sind keine
    Grundsymbole.
  </p>
</body></html>

```

****:
nummerierte Aufzählung
von Textelementen

****:
Aufzählung von
Textelementen mit
Markierung („Bullet“)

****:
Kennzeichnung der
Textelemente



Definitionslisten

```
<html><head>
  <title>Definitionslisten</title>
</head><body>
  <p> Wir definieren folgende Begriffe:
    <dl>
      <dt>Client/Server-Prinzip</dt>
      <dd>Einige Rechner (Server) bieten
        eine Dienstleistung an, andere
        Rechner (Clients) nutzen den
        Dienst.</dd>
      <dt>Gateway</dt>
      <dd>Rechner, der ein Netz mit
        anderen Netzen verbindet.</dd>
      <dt>W3C</dt>
      <dd>World Wide Web Consortium</dd>
    </dl>
  </p>
</body></html>
```

<dl>:

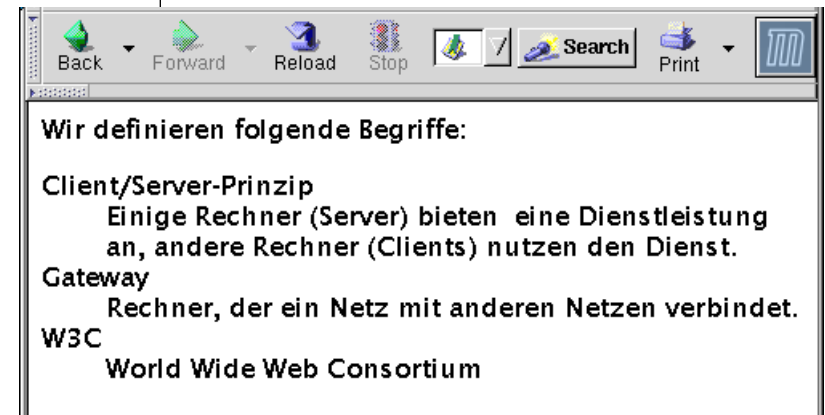
eine Liste mit Paaren von Begriffen und Erklärungen dazu

<dt>:

markiert den Begriff

<dd>:

markiert die Erklärung

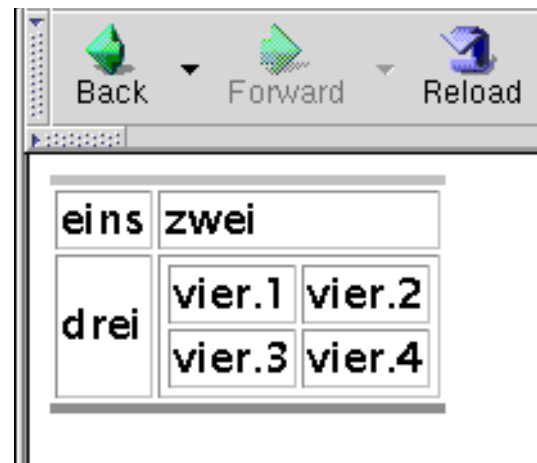


Tabellen

```

<html><head>
  <title>Tabellen</title>
</head><body>
  <table border="4" frame="hsides">
    <tr><td>eins</td> <td>zwei</td>
    </tr>
    <tr><td>drei</td>
      <td><table border="2" frame="void">
        <tr><td>vier.1</td>
          <td>vier.2</td>
        </tr>
        <tr><td>vier.3</td>
          <td>vier.4</td>
        </tr>
      </table>
    </td>
  </tr>
</table>
</body></html>

```



<table>:
eine Tabelle

Attribut **border**:
Breites des Randes um
die Zellen

Attribut **frame**:
äußerer Rahmen;

Werte:
void, box,
above, below,
hsides, vsides,
lhs, rhs

<tr>:
Tabellenzeile

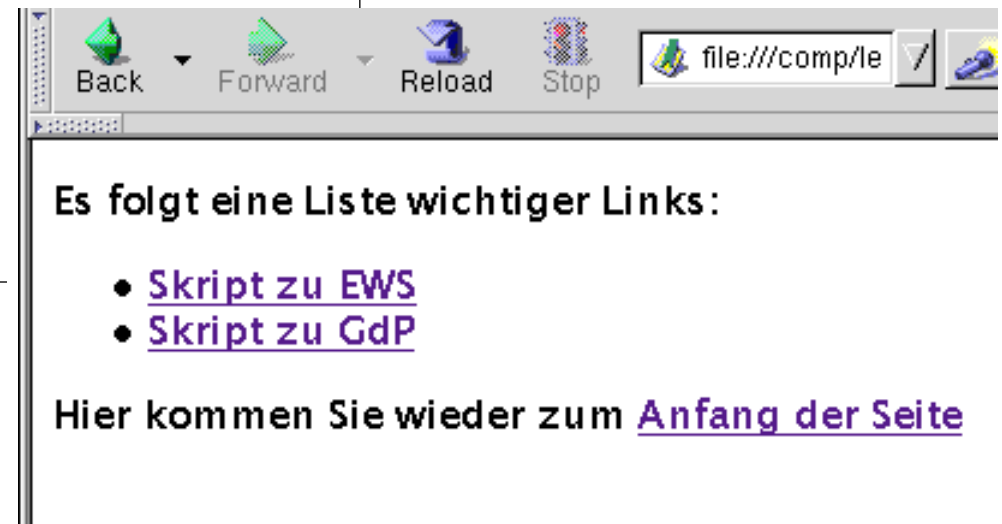
<td>:
Tabellenzelle,
kann selbst eine
Tabelle enthalten

Anker

```
<html><head>
  <title>Anker</title>
</head><body>
  <a name="SeitenAnfang">
  <p> Es folgt eine Liste wichtiger Links:
  <ul>
    <li> <a href="http://ag-kastens.upb.de/lehre/material/ews">
      Skript zu EWS</a>
    </li>
    <li> <a href="http://ag-kastens.upb.de/lehre/material/gdp">
      Skript zu GdP</a>
    </li>
  </ul>
  Hier kommen Sie wieder zum
  <a href="#SeitenAnfang">
  Anfang der Seite</a>
  </p>
</body></html>
```

:
platziert den Anker XYZ;
wird adressiert mit #XYZ

**Text
:**
setzt einen Link zu Adr
mit dem angegebenen
Text



Repräsentation spezieller Zeichen

Zeichen, die in der Notation von **HTML eine spezielle Bedeutung** haben, müssen **umschrieben** werden, wenn sie in normalem Text nicht als HTML-Zeichen gemeint sind

<	<	<p>Das Tag &lt;p&gt; kennzeichnet einen Absatz</p>
>	>	
&	&	für jedes Zeichen gibt es auch einen numerischen Code: &#38;
"	"	
	 	non-breaking space, Leerzeichen, das nicht durch Zeilenumbruch ersetzt werden darf

Allgemeine Form: **&Zeichename;** oder **&#Zahl;** wobei die Zahl das Zeichen codiert

HTML-Beschreibungen enthalten Listen mit solchen Zeichendefinitionen (*entities*), z. B.

ä	ä	
Ä	Ä	
ß	ß	<p>Maßkrüge</p>
ñ	ñ	<p>El niño</p>
©	©	<p>©2006 bei Dr. M. Thies</p>

Zweck-bezogene Hervorhebungen von Text

Textstücke können gekennzeichnet werden, damit sie durch **besonderen Schriftsatz hervorgehoben** werden. Dafür sollte man den **Zweck der Hervorhebung** kennzeichnen und **nicht die Darstellung** festlegen, z. B.

```
<p>Vor Verlassen des Raumes das <em>Licht ausschalten</em>,
aber <strong>nicht den Notausschalter</strong> benutzen!
```

Hier ist der Zweck, **Betonung** und **starke Betonung**, angegeben. Die Darstellung im Schriftsatz kann man unabhängig davon festlegen.

Das ist in folgendem Beispiel nicht mehr möglich und daher **nicht empfohlen**:

```
<p>Vor Verlassen des Raumes das <b>Licht ausschalten</b>,
aber <font color="red">nicht den Notausschalter</font> benutzen!
```

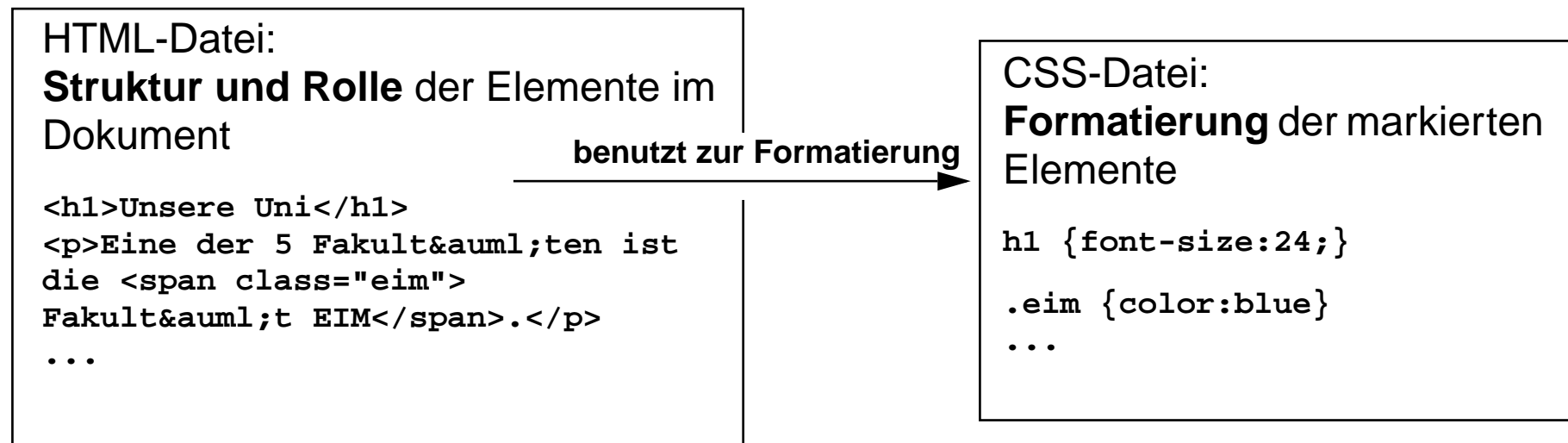
Weitere Beispiele für **Zweck-bezogene Hervorhebungen**:

<code><cite></code>	Zitat	Zweck wird durch das class-Attribut individuell bestimmt
<code><samp></code>	Beispiel	
<code><dfn></code>	Definition	
<code><code></code>	Quell-Code	<code></code>
<code><kbd></code>	Tastatureingabe	für elementaren Fließtext
<code><var></code>	Variablenname	<code><div class="meinAnteil"></code>
		für Textblöcke mit Unterstrukturen

S2.2 Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) ist eine Sprache zur **Definition von Formateigenschaften für HTML-Auszeichnungen**.

CSS wird verwendet, um **Formatierungsangaben** von den Auszeichnungen der Struktur und der Zweck-bezogenen Hervorhebungen zu **trennen**:



Ergebnisse:

- **konsistente Formatierung im ganzen Dokument**
- konsistente Formatierung **in vielen Dokumenten** (Corporate Identity)
- **einfache Änderung** der Formatierung in der CSS-Datei - HTML-Dateien bleiben unverändert

Notationen von Formatangaben in CSS

Im <head>-Abschnitt der HTML-Datei wird ein **Link** auf die CSS-Datei eingetragen mit dem **relativen Dateinamen** oder der vollen URL:

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

CSS-Angaben können auch in die HTML-Datei eingebettet werden - sind dann aber **nicht wiederverwendbar**: `<style type="text/css"> CSS-Angaben </style>`

Formatangaben in CSS für bestimmte Arten von HTML-Tags:

```
h1, h2 {text-align:center;color:red;}
```

```
em      {font-weight:bold;}
```

```
strong  {font-weight:bold;color:red;}
```

allgemeine Form: **HTML-Tags { Folge von Formatangaben }**

Formatangabe: **Name:Wert;** keine Leerzeichen; nur mehrteilige Werte in "

Beispiele:

```
font-family:Times;  
font-style:italic;  
text-decoration:underline;  
list-style-type:disc;  
list-style-type:lower-alpha;
```

Freie Klassifikation von HTML-Tags

Bestimmte **Zwecke der Hervorhebung** kann man durch **frei erfundene Namen klassifizieren**, z. B.

Die Idee, `der Entwurf` und die ..

```
<div class="meinAnteil">
```

```
<h3>Entwurf</h3>
```

```
<p>Das Software-System besteht aus ...</p>
```

```
...
```

```
</div>
```

In einem CSS Style Sheet kann man solchen Klassen Formatierangaben zuordnen:

```
.meinAnteil {font-weight:bold;color:blue;}
```

oder in einer anderen CSS-Datei bescheidener auf die Hervorhebung verzichten:

```
.meinAnteil {}
```

Auch den Struktur-Tags mit festgelegter Bedeutung können über das Klassenattribut Formateigenschaften zugeordnet werden:

```
<p class="meinAnteil">Der Entwurf des Systems ist ...</p>
```

Allerdings wäre eine Klammerung mit `` konsequenter.

Beispiel: 2 verschiedene Formatierungen

```

<html><head>
  <title>CSS Beispiel</title>
  <link rel="stylesheet" type="text/css"
    href="cssbsp.css">
</head><body>
  <h1>Das Fantasie-System</h1>
  <p> In diesem Kapitel beschreiben wir
  die Aufgabe, den
  <span class="ukas">Entwurf</span>
  und die Implementierung des
  Fantasie-Systems.</p>

  <h2>Aufgabe</h2>
  <p>Zu den <dfn>Aufgaben</dfn>
  geh&ouml;ren blah und blah</p>

  <div class="ukas">
  <h2>Entwurf</h2>
  <p>Der <dfn>Entwurf</dfn> des Systems ist
  gut gelungen blah blah</p>

  <h4>Detail-Entwurf</h4>
  <p>Der Detail-Entwurf ist auch ganz
  toll.</p>
  </div>

  <h2>Implementierung</h2>
</body></html>

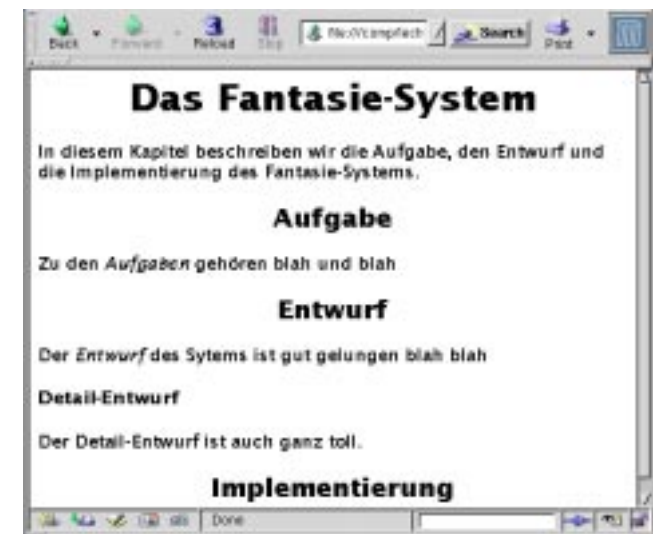
```



```

h1, h2, h3 {text-align:left;}
dfn      {font-style:italic;font-weight:bold;}
.ukas    {font-weight:bold;color:blue;}

```



```

h1, h2, h3 {text-align:center;}
dfn {font-style:italic;}
.ukas {}

```

S2.3 Formulare mit Eingabeelementen

Graphische Elemente für **interaktive Eingaben**:

z. B. Textfelder, Auswahllisten, Schaltelemente, usw.

- strukturiert und gestaltet mit allgemeinen Dokument-Tags
- zu Formularen zusammengefasst

Eingabedaten werden

- zum **Web-Server gesandt** und dort verarbeitet,
siehe Beispiel Telefonbuch;
siehe Abschnitt *W4 Interaktive, dynamische Web-Seiten*
- **im Browser** geprüft und/oder verarbeitet
siehe Abschnitt *S4 JavaScript*

The screenshot shows a web browser interface with a navigation bar at the top containing 'Back' and 'Forward' buttons, and 'Home' and 'Bookmarks' links. Below the navigation bar is a form with several sections:

- Zuname:** A text input field containing the text 'Kastens'.
- Gästebuch:** A text area containing the text 'Das sieht noch recht mager aus!'.
- Versand:** A checkbox labeled 'eilig' which is checked.
- Zahlung:** Two radio buttons, 'Bar' (selected) and 'Karte'.
- Wochentag:** A dropdown menu with 'Freitag', 'Samstag' (highlighted), and 'Sonntag' as options.
- At the bottom of the form are two buttons: 'löschen' and 'abschicken'.

Ein einfaches Formular

```
<form action="http://www.upb.de" method="get">
  <p>Ihr Zuname:
    <input type="text" name="Zuname" size="10">
  </p>
  <input type="submit" value="abschicken">
</form>
```

<form>

fasst Eingabeelemente (<input>) und weitere Elemente zu einem Formular **zusammen**. Alle Eingabedaten eines Formulars werden gemeinsam verarbeitet.

action-Attribut

Ziel, an das die Eingabe geschickt wird;
Web-Adresse;
Web-Seite enthält ggf. ein Programm, das die Eingabe verarbeitet

method-Attribut

charakterisiert die **Art der Übertragung** zum Ziel;
hier **get**: als Parameter der URL



Schaltflächen (Knöpfe)

```
<input type="button" value="zurück"
      onclick="javascript:history.back()">
<input type="reset" value="löschen">
<input type="submit" value="abschicken">
```

<input type="button">
charakterisiert eine **allgemeine Schaltfläche**

value-Attribut

Beschriftung der Schaltfläche

onclick-Attribut

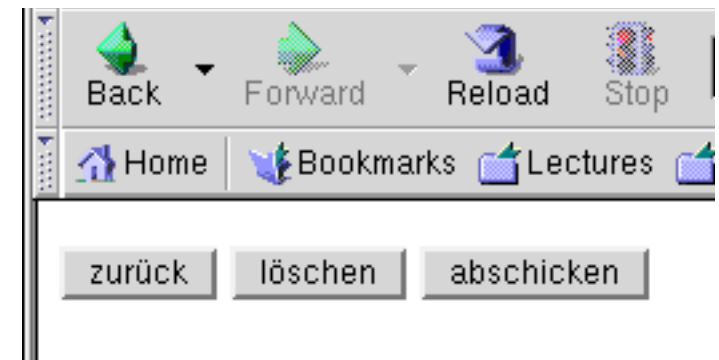
Aktion, die beim Betätigen ausgeführt wird;
hier Aufruf einer Funktion in JavaScript

<input type="reset">

spezielle Schaltfläche: alle **Eingabeelemente** in den
Initialzustand zurücksetzen

<input type="submit">

spezielle Schaltfläche: **Eingabe abschicken**;
eine davon in jedem Formular!



Einzeiliges Textfeld

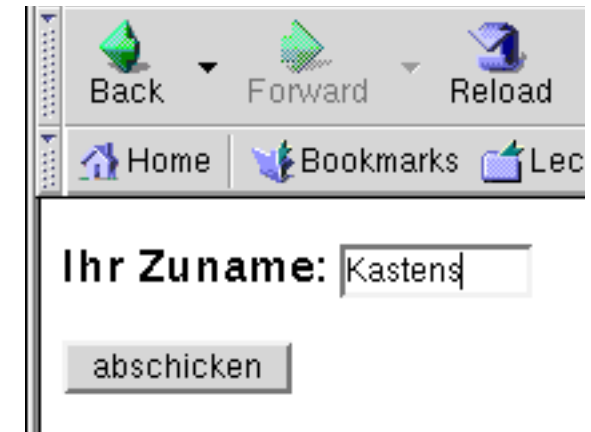
```
<input type="text" name="Zuname" size="10" maxlength="60">
```

`<input type="text">`
charakterisiert **einzeiliges Textfeld**

name-Attribut
identifiziert das Eingabeelement
im verarbeitenden Programm

size-Attribut
Größe des angezeigten Textfeldes in Zeichen

maxlength-Attribut
Anzahl der Zeichen, die **maximal eingegeben**
werden können
Scrolling, wenn `maxlength > size`



Mehrzeiliges Textfeld

```
<p>Eintrag ins Gästebuch:<br>  
  <textarea name="Eintrag" rows="5" cols="20">Hier schreiben!  
  </textarea>  
</p>
```

<textarea>

charakterisiert **mehrzeiliges Textfeld**;
zwischen den Tags kann
Initialisierung stehen

name-Attribut

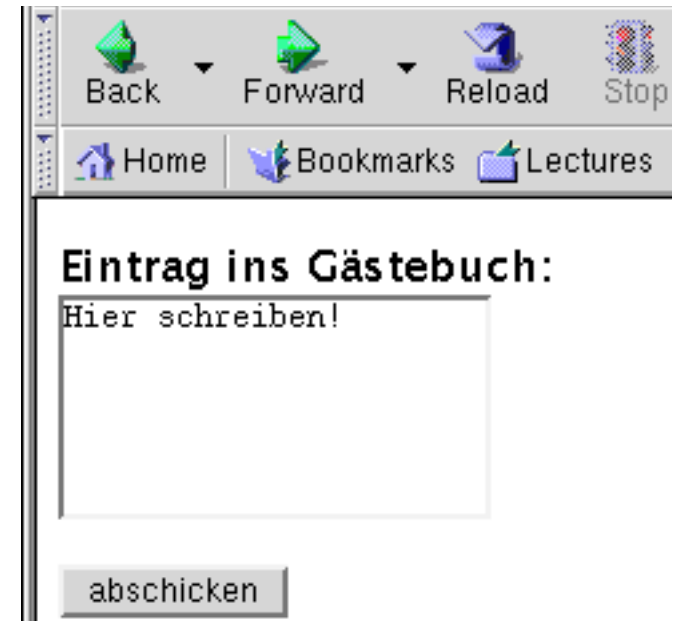
identifiziert das Eingabeelement
im verarbeitenden Programm

rows-Attribut

Anzahl der angezeigten **Zeilen**

cols-Attribut

Anzahl der angezeigten **Spalten**



Auswahlliste

```
<p>Wochentag:<br>
  <select name="tag" size="3">
    <option value="fr">Freitag
    <option value="sa">Samstag
    <option value="so">Sonntag
    <option value="mo">Montag
  </select>
</p>
```

<select>

Klammert eine Liste von Einträgen,
aus der einer ausgewählt werden kann

name-Attribut

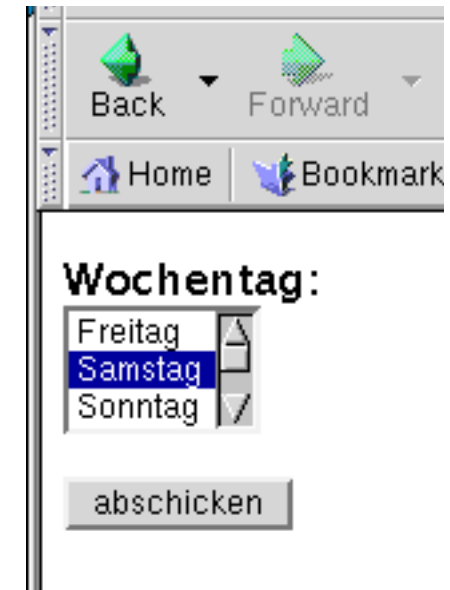
identifiziert das Eingabeelement
im verarbeitenden Programm

size-Attribut

Anzahl der angezeigten **Einträge**

value-Attribut

für das gewählte Element wird das
Paar (**name, value**) übertragen



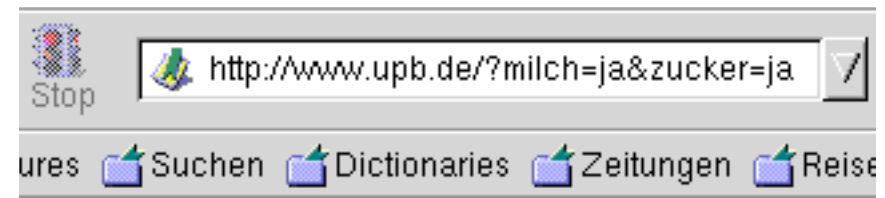
Checkbox

```
<p>Nehmen Sie zum Kaffee<br>  
  <input type="checkbox" name="milch" value="ja">Milch<br>  
  <input type="checkbox" name="zucker" value="ja">Zucker<br>  
</p>
```

`<input type="checkbox">`
ein Element, das man **auswählen** kann

name-Attribut
identifiziert das Eingabeelement
im verarbeitenden Programm

value-Attribut
wenn das Element gewählt ist, wird das
Paar (**name, value**) übertragen



Auswahlknöpfe

```
<p>Zahlungsart :<br>  
  <input type="radio" name="pay" value="cash">Bar<br>  
  <input type="radio" name="pay" value="card">Kreditkarte<br>  
  <input type="radio" name="pay" value="order">&Uuml;berweisung  
</p>
```

<input type="radio">

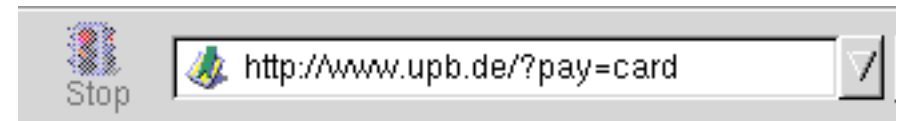
eine Gruppe von Knöpfen, von denen man genau einen **auswählen** kann (radio buttons)

name-Attribut

identifiziert das Eingabeelement im verarbeitenden Programm; alle Elemente der Gruppe haben den gleichen Namen

value-Attribut

wenn der Knopf gewählt ist, wird das Paar (**name, value**) übertragen



Datei zum Web-Server senden

```
<form action="http://www.upb.de" method="get"
  enctype="multipart/form-data">
  <p>Dateiname:
    <input type="file" name="Datei">
  </p>
```

<input type="file">

Datei versenden;

Name direkt angeben oder im
Dateiselektor auswählen

name-Attribut

identifiziert das Eingabeelement

enctype="multipart/form-data"

weiteres Attribut im **form**-Tag nötig

