Präsentation und Datenaustausch

- HTML: Überblick
- Interaktion mit HTML
- Überprüfung von Formularen
- Stylesheets: Grundlagen und Anwendung
- Transformation von XML: XSLT

HTML: Überblick

- Hypertext Markup Language (Tim Berners-Lee, CERN 1990)
- Auszeichnungssprache des Web
- erfordert Browser zur Präsentation
- Schwerpunkte
  - strukturelle Bestandteile eines Dokumentes (Absätze, Überschriften, Listen usw.)
  - Hypertext-Sprache: Verweise auf andere Dokumente, die vom Benutzer verfolgt werden können
- Weitere Elemente
  - Interaktive Elemente (Formulare)
  - Multimedia-Inhalte

HTML: Dokumentenstruktur

- HTML-Dokument:
  - Elemente, die in anderen Elementen verschachtelt sind
  - Element beginnt mit Start-Tag `<xxx>` und endet mit End-Tag `</xxx>`
  - Tag := Markierungsstruktur mit optionalen Attributen
  - höchste Ebene: <HTML>-Element bestehend aus
    - <HEAD> (Informationen über das Dokument)
    - <BODY> (Dokumenteninhalt)
  - BODY-Element: Behälterelemente, Strukturelemente, Sonderelemente

HTML: Behälterelemente

- können wie BODY andere Elemente beinhalten
- Erstellung von Abschnitten, Formularen, ...
- Beispiele
  - DIV: allgemeiner Abschnitt
  - FORM: Formular
  - BLOCKQUOTE: Zitat
  - ADDRESS: Name oder Adresse

HTML: Strukturelemente

- enthalten Text und Textelemente
- repräsentieren Abschnitte und Überschriften
- Beispiele:
  - P: Absatz
  - PRE: vorformatierter Abschnitt
  - H1 ... H6: Überschriften verschiedener Ebenen

HTML: Textelemente

- Text innerhalb von Strukturelementen
- charakterisieren von Wörtern
- Beispiele
  - EM: hervorheben (kursiv)
  - STRONG: stark hervorheben
  - B: fett
  - I: kursiv
  - SUB: tieferstellen
  - SUP : hochstellen

HTML: Sonderelemente

- repräsentieren Listen, Tabellen, Formularelemente
- Beispiele:
  - UL: unsortierte Liste
  - OL: sortierte Liste
  - DL: Definitionsliste
- UL und OL:
  - als Subelemente LI (list item)
- DL:
  - als Subelemente jeweils ein DT (Begriff) und ein oder mehrere DD (Definitionen)

HTML: Listen

- Beispiel Unsortierte Liste

```
<UL>
   <LI>Erstens</LI>
   <LI>Zweitens</LI>
</UL>
```

- Beispiel Definitionsliste

```
<DL>
   <DT>Begriff 1</DT>
   <DD>Definition zu Begriff 1</DD>
   <DT>Begriff 2</DT>
   <DD>Definition zu Begriff 2</DD>
</DL>
```


HTML: Tabellen

- Darstellung von Tabellen mit TABLE
- Inhalt bestehend aus
  - CAPTION: Über- oder Unterschrift
  - THEAD, TFOOT : Spaltenüberschriften
  - ein oder mehrere TBODY: Tabellenkörper
- THEAD, TFOOT, TBODY
  - jeweils aus TR (table row)-Elementen
- TR aus Tabellenzellen
  - TH: Tabellenüberschrift
  - TD: Tabellendaten

HTML: Tabellen

- Beispiel:

```
<table border=1>
<tbody>
    <tr><th rowspan=2></th>
    <th colspan=4>1999</th></tr>
    <tr><th>Q1</th><th>Q2</th>
        <th>Q3</th><th>Q4</th></tr>
    <tr><th>P1</th><td>45</td><td>67</td>
        <td>46</td><td>55</td></tr>
    <tr><th>P2</th><td>38</td><td>51</td>
        <td>41</td><td>61</td></tr>
</tbody>
</table>
```

	1999			
	Q1	**Q2**	**Q3**	**Q4**
P1	45	67	46	55
P2	38	51	41	61

Interaktion mit HTML

- Formular:
  - Gruppe von Eingabemöglichkeiten
  - Zurücksenden von Informationen an Web-Server
- erfordert
  - interaktive Elemente (Textfelder, Schalter, ...)
  - serverseitigen Mechanismus zur Verarbeitung
  - Kommunikation zwischen Browser und Server (HTTP)
  - Schnittstelle zwischen Server und Verarbeitungsmechanismus (CGI)

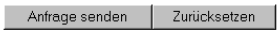
FORM-Element

- Element für Formulare
- definiert Methode zur Kommunikation mit Server
  - GET: Parameterübergabe durch Anhängen an URL (Standard)
  - POST: Parameterübergabe über Standardeingabe
- sowie Aktion
  - URL des Verarbeitungsprogramms (CGI-Programm, Servlet, PHP-Skript, ...)

FORM-Element

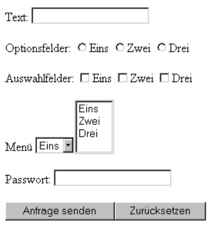
- Beispiel

```
<FORM ACTION="http://zeus/cgi-
bin/process">
   ... Eingabefelder ...
   <INPUT TYPE=submit>
   <INPUT TYPE=reset>
</FORM>
```



Formulare in HTML: Elemente

- Textfelder (einzeilig, mehrzeilig)
- Optionsfelder
- Auswahlfelder
- Menüs
- Schalter
- Paßwort-Felder



Formularelemente: Textfelder

- Einfaches Textfeld

```
<INPUT NAME="name" SIZE=len MAXLENGTH=max
   VALUE=default>
```

  - NAME: Feldname für serverseitige Verarbeitung
  - SIZE: Anzahl der anzuzeigenden Zeichen
  - MAXLENGTH: max. Anzahl von Zeichen
  - VALUE: Anfangswert
- Mehrzeilige Textfelder

```
<TEXTAREA ROW=rows COLS=cols>
   Anfangswert
</TEXTAREA>
```

Formularelemente: Optionsfelder

- Optionsfeld: Auswahl genau eines Feldes

```
<INPUT TYPE=radio NAME=name VALUE="Wert1"
   CHECKED>Eins
<INPUT TYPE=radio NAME=name
   VALUE="Wert2">Zwei
```

- NAME-Attribut muß für alle Felder einer Gruppe gleich sein
- ein Element muß CHECKED sein

Formularelemente: Auswahlfelder

- Auswahlfeld: Auswahl mehrerer Felder möglich

```
<INPUT TYPE=checkbox NAME=name
   VALUE="Wert1" CHECKED>Eins
<INPUT TYPE=checkbox NAME=name
   VALUE="Wert2">Zwei
```

- Initialisierung mit CHECKED
- NAME *kann* für alle Elemente gleich sein

Formularelemente: Schalter

- Submit-Schalter
  - Absenden der Eingabe
    `<INPUT TYPE=submit>`
- Reset-Schalter
  - Rücksetzen aller Eingabefelder auf Initialisierunswerte
    `<INPUT TYPE=reset>`
- Bezeichner werden vom Browser vergeben: Änderung über VALUE-Attribut
  `<INPUT TYPE=submit VALUE="Suchen">`

Formularelemente: Paßwortfelder

- Textfeld mit versteckter Eingabe:
  - * oder Leerzeichen für jedes Zeichen
  - Verwendung: Paßwort-Abfrage

```
<INPUT TYPE=password NAME=passwd>
```


Formularelemente: Versteckte Felder

- Speichern von Zustandsinformationen über verschiedene Seiten hinweg
  - Sitzungsdaten
  - Warenkorb

```
<INPUT TYPE=hidden NAME=name VALUE=data>
```


Gestaltung von Formularen

- Prinzipien
  - Inhaltlich zusammenhängende Felder zusammenfassen (durch <HR> trennen, Hintergrundfarbe)
  - gleichmäßiges Layout, Ausrichtung (Anordnung in Tabellen)
  - Auswahl geeigneter Elemente
  - Überprüfung korrekter Eingaben ???

Formularüberprüfung

- HTML/HTTP:
  - erfaßte Werte können erst bei Verarbeitung auf Server überprüft werden
- Ausweg:
  - für bestimmte Fälle
    - Datenformat (z.B. Datum, EMail-Adresse)
    - benötigte Werte
  - client-seitige Überprüfung *vor* Übertragung der Daten
  - Einbettung von Skripten in HTML-Dokumente
  - Beispiel: JavaScript

JavaScript im Web-Browser

- Einbettung von JavaScript-Funktionen in HTML-Dokumente
- Ausführung im Web-Browser
- Skript-Definition als spezielles SCRIPT-Element

```
<html>
<body>
<script language="JavaScript">
   document.write ("Hallo User !");
</script>
</body>
</html>
```


JavaScript im Web-Browser

- Zuordnung von JavaScript-Funktionen zu HTML-Elementen über Ereignisse als *Event Handler*

```
<form>
   <input type="button"
    value="Dr&uuml;ck mich !"
    onClick="alert('Danke !')">
</form>
```

Überprüfung von Eingaben

- EventHandler für FORM:
  - Funktion für **`onSubmit`**-Attribut
  - liefert **`true`** wenn gültige Eingabe → Übertragen der Daten zum Server
  - sonst **`false`** und Fehlermeldung

```
<FORM onSubmit="return verify (this);">
   Vorname:
   <INPUT TYPE=text NAME="firstname"><BR>
   Nachname:
   <INPUT TYPE=text NAME="lastname"><BR>
   <INPUT TYPE=submit>
</FORM>
```


Überprüfung von Eingaben

```
// Testet, ob alle Eingabefelder belegt sind
function verify (f) {
  for (var i = 0; i < f.length; i++) {
    var e = f.elements[i];
    if (e.type == "text" &&
        (e.value == null ||
         isblank (e.value))) {
           alert ("Fehler");
           return false;
     }
  }
  return true;
}
```


Überprüfung von Eingaben

- FORM: JavaScript-Objekt
  - Zugriff auf FORM-Elemente über ***form*.elements[]**
  - Feld von INPUT-Objekten mit eigenen Attributen
    - **`type`**: Typ des Objektes (hier Text)
    - **`value`**: aktueller Wert
    - **`name`**: Elementname

Überprüfung von Eingaben

```
// Testet, ob String s ein Leerstring ist
function isblank (s) {
  for (var i = 0; i < s.length; i++) {
    var c = s.charAt (i);
    if ((c != ' ') && (c != '\n') &&
        (c != '\t'))
      return false;
  }
  return true;
}
```


Stylesheets: Motivation

- HTML beschreibt *Struktur* eines Dokumentes
- Beschränkungen für anspruchsvolle Präsentationen
  - proprietäre Erweiterungen (z.B. <MARQEE>)
  - Bilder statt Text
  - Text in Tabellen
  - Programme statt HTML
- Nachteile:
  - eingeschränkter Nutzerkreis
  - beeinträchtigte Zugänglichkeit (z.B. durch Suchmaschine)
  - komplizierte Strukturen
  - Nutzung in einigen Jahren ?

Stylesheets: Motivation

- Stylesheet := Formatvorlage
  - Sammlung stilistischer Anweisungen
  - beschreibt wie HTML-Elemente dargestellt werden sollen (Größe, Farbe, Abstände, Aufteilung, ...)
- Trennung von Struktur und Präsentation
  - verbesserte Weiterverarbeitung
  - verschiedene Präsentationen möglich
    - in Abhängigkeit vom Ausgabegerät, vom Nutzer, ...

Stylesheets für WebDB-Anwendungen

- Vorteile:
  - einheitliches Erscheinungsbild für alle Seiten einer Anwendung
  - vereinfachte Anpassung der gesamten HTML-Benutzerschnittstelle
  - vereinfachte Seitengenerierung
  - Trennung von Struktur und Layout !!

Cascading Style Sheets (CSS)

- Sprache zur Definition von Formatvorlagen
- W3C-Standard zu Formatvorlagen für HTML-Dokumente
- CSS1 (implementiert in MS Internet Explorer, Netscape Navigator, Opera) + CSS2
- Weitere Ansätze
  - DSSSL (Document Style Semantics and Specification Language): ISO-Norm für SGML-Stylesheet-Sprache
  - XSL (Extensible Style Language): Stylesheet-Sprache für XML (in Entwicklung)

CSS: Grundbegriffe

- **Formatvorlage**: Menge von Anweisungen
- **Anweisung**: Aussage über stilistischen Aspekt eines oder mehrerer Elemente
- **Aufbau**:

Selektor Deklaration

```
H1 { color: blue }
```

Eigenschaft Wert

CSS: Grundbegriffe

- **Selektor**:
  - Verbindung zwischen HTML-Dokument und Fromat
  - spezifiziert betroffene Elemente
  - z.B. H1 - alle <H1>-Elemente
- **Deklaration**: beschreibt Effekt
  - Eigenschaft: Qualität, Merkmal, z.B. Farbe
  - Wert: Spezifikation der Eigenschaft, z.B. blue

Gruppieren

- Gruppieren von Selektoren

```
H1 { font-weight: bold }
H2 { font-weight: bold }
H3 { font-weight: bold }
```

  zu

```
H1, H2, H3 { font-weight: bold }
```

- Gruppieren von Eigenschaften

```
H1 { color: red }
H1 { font-weight: bold }
```

  zu

```
H1 { color: red; font-weight: bold }
```


Selektoren

- Typselektoren
  - Name des Elementtyps als Selektor, z.B. H1
  - gültig für alle Instanzen dieses Typs
- Attributselektoren
  - Zuordnung von Formatierungen über HTML-Attribute CLASS und ID
- Kontextuelle Selektoren
  - Berücksichtigung des Kontextes (der übergeordneten Elemente im Dokumentenbaum)
- externe Informationen
  - Pseudoklassen und -elemente

Attributselektoren

- Anwendung von Formatierungen auf Gruppen bzw. einzelne Instanzen
- HTML-Attribute:
  - CLASS
    - Klassifikation von Elementen
  - ID
    - Kennzeichnung eines einzelnen Elementes
  - STYLE
    - Ersatz für Selektormechanismus

Attributselektoren

- CLASS-Definition in HTML:
  <*element* **CLASS**=*bezeichner*>
- Beispiel: **`<P CLASS=main>`**
- Definition des Klassenselektors

Klassenselektor Deklaration

```
.main { color: blue }
```

Klassenname
Kennzeichnungssymbol

Attributselektoren

- ID-Definition in HTML:
  <*element* **ID**=*ident*>
- *ident* muß eindeutig im Dokument sein !
- Beispiel: **`<P ID=absatz1>`**
- Definition des ID-Selektors

ID-Selektor Deklaration

```
#absatz1 { color: blue }
```

ID-Wert
Kennzeichnungssymbol

Attributselektoren

- Formatinformationen als Attribut eines HTML-Elementes
- nur gültig für das zugeordnete Element
- Beispiel:

```
<BODY
  STYLE="color: black; background: white">
  <P STYLE="font-weight: bold">Ein Text.
```


Kontextuelle Selektoren

- Einschränkung der Anwendung von Selektoren durch Vorfahrenelemente
- Spezifikation des Kontextes, bestehend aus zwei oder mehr einfachen Selektoren (Typ-, Klassen-, ID-Selektoren)
- Beispiel:

```
H1 EM { color: blue }
```

- HTML:

```
<H1>Diese Überschrift ist
<EM>wichtig</EM>.</H1>
```


Pseudoklassen und -elemente

- Zuordnung von Formatierungen zu nicht direkt sichtbaren Elementen
- Pseudoklassen für Anchor (<A>)
  - LINK: Verweis
  - VISITED: bereits genutzer Verweis
  - ACTIVE: aktuell ausgewählter Verweis
- Beispiel

```
A:link { color: blue }
A:visited { color: black }
```

Pseudoklassen und -elemente

- Pseudoelemente
  - first-letter
  - first-line
- Beispiel

```
P:first-line {
     text-transform: uppercase }
P:first-letter { font-size: 200% }
```


Anbindung von Stylesheets

- Element <STYLE>
  - für das gesamte Dokument
  - Beispiel

```
<HTML>
 <STYLE TYPE="text/css">
      H1 { color: blue; text-weight: bold }
      P { color: black }
 </STYLE>
 <BODY>
      ...
 </BODY>
</HTML>
```

- Attribut STYLE
  - für ein einzelnes Element

Anbindung von Stylesheets

- Element <LINK>
  - Nutzung externer Vorlagen (z.B. organisationsweite Stylesheets)
  - Angabe über URL
  - Beispiel

```
<HTML>
 <LINK REL=STYLESHEET
HREF="/styles/main.css">  <BODY>
      ...
 </BODY>
</HTML>
```

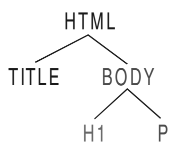
Anbindung von Stylesheets

- @import:
  - Importieren einer externen Vorlage und Verbinden mit der aktuellen
  - z.B. dokumentspezifische, zusätzliche Vorlage neben den organisationsweiten Stylesheets)
  - Beispiel:

```
<STYLE TYPE="text/css">
 @import "headings.css";
 H1 { color: red }
</STYLE>
```


Vererbung und Kaskadierung

- **Vererbung**:
  - Elemente erben Formateigenschaften von Elternelementen in der Baumstruktur
  - Beispiel:
    `BODY { color: red }`
  - Außerkraftsetzen durch speziellere Anweisungen
    `H1 { color: blue }`
  - Eigenschaften ohne Vererbung (z.B. `background`)



Vererbung und Kaskadierung

- **Kaskadierung**:
  - Regelung der Kombination mehrerer Formatvorlagen (Konfliktauflösung)
  - Überlappung der Formatvorlagen von Designer, Browser, Benutzer
  - Regeln:
    - Vorlagen des Designers heben Benutzeranweisungen auf
    - Gewichtung von Anweisungen durch `!important`
      `H1 { color: red !important }`

Eigenschaften am Beispiel

- Listen
  - **list-style-type: disc ●, circle ❍, square ■, decimal** (1, 2, 3, ...), **lower-roman** (i, ii, iii, ...) **lower-alpha** (a, b, c, ...), **..., none**
  - **list-style-image:** URL
  - **list-style-position: inside, outside**
  - Beispiele

```
UL { list-style-image:
url(http://host/image.gif) }
OL { list-style-type: decimal }
OL OL { list-style-type: upper-alpha }
OL OL OL { list-style-type: upper-roman }
```


Eigenschaften am Beispiel

- Schriften
  - **font-family:** Liste von Fonts
  - **font-style: normal, italic, oblique**
  - **font-variant: normal, small-caps**
  - **font-weight: normal, bold, bolder, lighter, ...**
  - **font-size:** absolut, relativ, prozentual
  - Beispiele

```
BODY { font-family: Garamond, Times }
EM.extra { font-size: 120% }
H2.font { font-size: smaller }
```


Eigenschaften am Beispiel

- Texte
  - **text-decoration: none, underline, blink, ...**
  - **text-transform: capitalize, uppercase, lowercase**
  - **text-align: left, right, center, justify**
  - Beispiele

```
H1 { text-transform: capitalize;
     text-align: center }
```

Eigenschaften am Beispiel

- Rahmeneigenschaften
  - **`border-color:`** Farbe
  - **`border-style: none, dotted, solid, inset, outset, ...`**
  - **`border-width: thin, medium, thick`**, Wert
- Plazierung und Ränder
  - **`float, clear`**
  - **`margin-left, margin-right`**
  - **`width, heigth`**

Zusammenfassung

- CSS: erster Ansatz einer Layoutsprache für Web-Dokumente
  - Trennung von Struktur und Layout
  - Vereinfachung einer
- gegenwärtige Implementierung mit Einschränkungen
  - Unterschiede bei einzelnen Browsern bzw. fehlerhafte Umsetzung
  - optionale Features (z.B. Pseudoelemente)

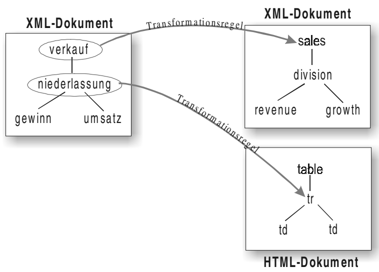
Transformation von XML

- Wandlung der Struktur von XML-Dokumenten
  - zur Präsentation (XML → HTML)
    - XML-Daten im Web veröffentlichen
  - zum Datenaustausch (XML → XML)
    - Rechnungsdatensätze einer Firma A in Format einer Firma B konvertieren
  - zur Strukturänderung (XML → XML)
    - Datenaufbereitung, z.B. Inhaltsverzeichnis erstellen

Transformation von XML: XSLT

- XSLT := Extensible Stylesheet Language Transformation
- Transformationsteil der Stylesheet-Sprache XSL
  - XPath: Verweise auf Teile von XML-Dokumenten
  - XSL: Formatierung (Farben, Fonts, Layout, ...)
- Transformation
  - Menge von Regeln, die Umwandlung eines Dokumentenquellbaums in Ergebnisbaum

XSLT: Prinzip



XSLT: Grundbegriffe

- Stylesheet:
  - Dokument mit Transformationsanweisungen (template rules)
- Muster (pattern):
  - beschreibt Knoten im Dokumentenbaum, auf die Transformation anzuwenden ist
- Schablone (template):
  - wird als Ergebnis der Transformation instantiiert und in den Ergebnisbaum übernommen
  - umfaßt konstanten Text und weitere XSLT-Instruktionen

Stylesheet: Struktur

- Stylesheet ist selbst XML-Dokument

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/XSL/Transform/1.0">
  <!-- Stylesheets -->
</xsl:stylesheet>
```

- für Erzeugung von HTML

```
 <xsl:stylesheet xmlns:xsl=
  "http://www.w3.org/XSL/Transform/1.0"
  xmlns=http://www.w3.org/TR/REC-html40"
  result-ns="">
```


Definition von Regeln

- Notation

```
<xsl:template match=pattern>
   <!-- content -->
</xsl:template>
```

- *pattern*: Muster zu Beschreibung des bearbeitenden Elementes
- *content*: Schablone, die bei Anwendung der Regel zu instantiieren ist
  - Text + Elemente des Ergebnisbaums
    Bsp.: `<td>Ein Text</td>`
  - Weitere XSLT-Instruktionen
    Bsp.: `<xsl:apply-templates/>`

Muster in Regeln

- Bedingung für Auswahl eines Knoten (Element) des Dokumentenbaumes und Anwendung der Regel
- entsprechend Mustern in XQL
- Beispiele
  - `/` Wurzelelement
  - `*` jedes Element
  - `para` jedes Element `para`
  - `para/item` jedes Element `item` unter einem Elternelement `para`
  - `@class` jedes Attribut `class`

Anwendung von Regeln

- rekursive Weiterverarbeitung der Kindelemente eines ausgewählten Knotens
- Notation

```
<xsl:apply-templates select=expression>
    <!-- content -->
</xsl:apply-templates>
```

- optionales **select**-Attribut zur Einschränkung auf bestimmte Elemente
- *expression*: Muster

Erzeugung des Ergebnisbaumes

- Text

```
<xsl:template match="node1">
    <p>Ein konstanter Text.</p>
</xsl:template>
<xsl:template match="node">
    <xsl:text>    </xsl:text>
</xsl:template>
```

- Elemente

```
<xsl:element name="elem-name"> ... </xsl:element>
```

- Attribute

```
<xsl:attribute name="attr-name"> ..
</xsl:attribute>
```


Berechnung

- Generierung von Text für den Ergebnisbaum durch
  - Extraktion von Text aus dem Quellbaum
  - Bestimmung des Wertes einer Variablen
- Notation

```
<xsl:value-of select=expression />
```

- Beispiel

```
<xsl:template match="person">
    <p>
        <xsl:value-of select="@first-name"/>
        <xsl:text> </xsl:text>
        <xsl:value-of select="@last-name"/>
    </p>
</xsl:template>
```

Wiederholte Anwendung

- Anwendung einer Regel auf eine Menge von (gleichartigen) Elementen
- Notation

```
<xsl:for-each select=expression>
   <!-- content -->
</xsl:for-each>
```

- *expression*: bestimmt Element, auf das die Schablone *content* anzuwenden ist

Bedingte Verarbeitung

- IF-Bedingung

```
<xsl:if test=expression>
   <!-- content -->
</xsl:if>
```

- Auswahl

```
<xsl:choose>
   <xsl:when test=expression>
       <!-- content -->
   </xsl:when>
   ...
</xsl:choose>
```


Sortieren

- Sortieren der Kindelemente eines `xsl:apply-templates`- oder `xsl:for-each`-Elementes bzgl. eines Schlüssels
- Notation

```
<xsl:sort select=expression data-type=type
order=ordering />
```

  - *expression*: bestimmt Sortierschlüssel
  - *type*: Typ des Schlüssels (`text, number`)
  - *ordering*: Ordnung (`ascending, descending`)

Beispiel: XML-Dokument

```
<verkauf>
   <abteilung id="Nord">
        <umsatz>10</umsatz>
        <gewinn>9</gewinn>
   </abteilung>
   <abteilung id="West">
        <umsatz>4</umsatz>
        <gewinn>-1</gewinn>
   </abteilung>
</verkauf>
```


Beispiel: XSLT-Stylesheet (Auszug)

```
<xsl:for-each select="verkauf/abteilung">
 <xsl:sort select="umsatz" data-type="number"
          order="descending" />
  <tr>
   <td><em><xsl:value-of select="@id" /></em></td>
   <td><xsl:value-of select="umsatz"/></td>
   <td><xsl:if test="gewinn &lt; 0">
         <xsl:attribute name="style">
           <xsl:text>color:red</xsl:text>
         </xsl:attribute>
       </xsl:if>
       <xsl:value-of select="umsatz"/></td>
  </tr>
</xsl:for-each>
```


Beispiel: HTML-Ergebnis (Auszug)

```
<table border="1">
  <tr>
    <td><em>Nord</em></td>
    <td>10</td><td>9</td>
  </tr>
  <tr>
 <td><em>West</em></td>
    <td>4</td>
    <td style="color:red">-1</td>
  </tr>
</table>
```

Zusammenfassung

- XSLT: Mechanismus zur Transformation von XML-Dokumenten
  - als Teil der Stylesheet-Sprache XSL
  - zur Umwandlung von Dokumenten (z.B. nach XML oder HTML)
- Deklarative Transformationsvorschrift
  - keine Programmierung notwendig
- Status
  - kurz vor Verabschiedung durch W3C
  - erste Implementierung verfügbar (XT)