Studienarbeit

Entwicklung eines adaptiven computergestützten HTML-Kurses

von
cand. inform. Dennis Maciuszek

Aufgabenstellung und Betreuung:
Prof. Dr. Ursula Goltz
Prof. Dr. Cornelia E. Dowling

Technische Universität Braunschweig
Institut für Software, Abteilung Programmierung
Institut für Psychologie,
Abteilung Mathematische Psychologie und Sozialpsychologie

10. Januar 2001

Erklärung

Ich versichere, die vorliegende Arbeit selbständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

Braunschweig, _____ _____

(Dennis Maciuszek)

Zusammenfassung

Es wird eine Anwendung für das adaptive computergestützte Lehrsystem AdAsTra entworfen. Lerngegenstand sind die Grundlagen der Hypertextbeschreibungssprache HTML. Das entstehende Lernprogramm ist sowohl für Anfänger als auch für Fortgeschrittene konzipiert, denn es stellt den Lernern Übungsaufgaben, die auf ihr Vorwissen abgestimmt sind.

Zu diesem Zweck wird ein Modell des Wissensgebietes hergeleitet, aufbauend auf welchem der Wissensstand eines beliebigen Lerners modelliert bzw. vom Programm diagnostiziert werden kann. Die mathematischen Grundlagen liefert die Theorie der Wissensräume. Die Wissenstruktur HTML wird mittels einer neue Variante der Expertenbefragung aus einem Hypermedia-Dokument extrahiert.

Ausgehend von der gewonnenen Wissensstruktur wird eine große Anzahl Übungsaufgaben mit ausführlichen Erklärungen und Feedback kreiert und implementiert. Dabei werden zunächst Überlegungen zum psychologisch geschickten Entwurf von computergestellten Aufgaben allgemein angestellt und zusammengetragen.

Inhaltsverzeichnis

Abbildungsverzeichnis **6**

Tabellenverzeichnis **7**

1 Einführung **8**

2 Wissensstruktur HTML **20**
2.1 Die Struktur von "SELFHTML" 20
2.2 Aufstellen der Implikationsrelation 28
2.3 Berechnung der Basis des Wissensraumes 36

3 Aufgabengestaltung **43**
3.1 Planung aus psychologischer Sicht 43
3.2 Entwurf der Aufgabentypen 60
3.3 Implementierung mit AdAsTra/Prola 2 67

Literaturverzeichnis **75**

A Darstellungen der Wissensstruktur **77**
A.1 SFBen 77
A.2 Basisdateien 92

B Aufgaben des HTML-Kurses **101**
B.1 Aufgabenliste 101
B.2 Kursdateien 166

Abbildungsverzeichnis

1.1 HTML-Seite in einem Browser 9
1.2 UND/ODER-Graph für einen HTML-Ausschnitt 15

2.1 Obere Gliederungsebene in SELFHTML 22
2.2 Mittlere Gliederungebene in SELFHTML 23
2.3 Untere Gliederungebene in SELFHTML 24
2.4 Die Baumstruktur von SELFHTML 25
2.5 Geschwisterordnung in SELFHTML 26
2.6 Querverweise in SELFHTML 28
2.7 UND/ODER-Graph aus der Baumstruktur 31
2.8 UND/ODER-Graphen mit Geschwisterordnung 32
2.9 UND/ODER-Graph einschließlich Querverweisen 35
2.10 UND/ODER-Graph in KQ_Win 38
2.11 Screenshot der KQ_Win-Befragung 42

3.1 Einschätzungsaufgabe mit Texteingaben kombiniert 47
3.2 Einige Gestaltgesetze 53
3.3 Screenshot einer Hilfestellung in AdAsTra 72
3.4 Screenshot einer Frage in AdAsTra 73
3.5 Screenshot eines Feedbacks in AdAsTra 73

Tabellenverzeichnis

2.1 Übersetzungsregeln Baumstruktur in SFBen 30
2.2 Übersetzungsregel Geschwisterordnung in SFBen 32
2.3 Übersetzungsregeln Querverweise in SFBen 33
2.4 Dimensionen der Basismatrizen nach der Expertenbefragung . 42
2.5 Dimension der Basismatrix nach der Integration 42

3.1 Bildschirmaussehen 52
3.2 Kapitelfarben 52
3.3 Itemschwierigkeiten 55
3.4 Aufgabentyp 1 – Auswahlaufgabe 61
3.5 Aufgabentyp 2 – Zuordnungsaufgabe 62
3.6 Aufgabentyp 3 – Umsetzungsaufgabe Schieberegler 63
3.7 Aufgabentyp 4 – Umsetzungsaufgabe Image Map 64
3.8 Aufgabentyp 5 – Implementierungsaufgabe vorgegeben 65
3.9 Aufgabentyp 6 – Implementierungsaufgabe kreativ 66

Kapitel 1

Einführung

Ziel Computer-Lernprogramme eröffnen Schülern und Schülerinnen vor allem im Selbststudium ganz neue Möglichkeiten, die herkömmliche Lehr- und Übungsbücher nicht bieten: Veranschaulichung von Lerninhalten mittels multimedialer Techniken (z. B. animierte Grafik, Ton), *Interaktion* und damit direkte Rückmeldungen *(Feedback)* zu Antworten auf Übungsaufgaben, eine weniger lineare Abfolge im Lernweg und vieles mehr. Dabei hängt es von den getroffenen didaktischen und software-ergonomischen Design-Entscheidungen bzw. deren Umsetzungen ab, ob ein Programm tatsächlich die erhofften Vorteile im Lernkomfort erzielt.

Computerunterstützter Unterricht (Computer-Based Training) kann auf sehr unterschiedliche Arten stattfinden. In dieser Arbeit wird ein *adaptiver Übungskurs* zum Lerngebiet "Hypertext Markup Language" (HTML), der Sprache zur Programmierung von Internet-Seiten (World Wide Web), entwickelt. Er richtet sich genauso an bereits erfahrene Webdesigner, die ihre Kenntnisse überprüfen und vervollständigen möchten, wie an Anfänger, die den Einstieg in ein kreatives Hobby suchen oder beruflich mit der Gestaltung einer Homepage beauftragt wurden. Das entstehende Lernprogramm orientiert sich am Wissensniveau eines Lerners (Adaptivität) und geht damit in die Richtung *Intelligentes Tutorielles System*. Der Schwerpunkt des Kurses wird nicht so sehr auf der Erläuterung des Stoffes liegen, stattdessen wird dem Benutzer eine Reihe von interaktiven Übungsaufgaben präsentiert, die er/sie am Rechner zu bearbeiten hat.

Die Schwierigkeit der dargebotenen Aufgaben wird automatisch an die individuellen Kenntnisse des Schülers angepaßt. Der zugehörige Algorithmus und die Datenstrukturen, auf welchen er arbeitet, entstammen der mathematisch-psychologischen Theorie der *Wissensräume*, wie sie von Jean-Paul Doignon und Jean-Claude Falmagne eingeführt und u. a. von Cornelia E. Dowling an der TU Braunschweig erweitert wurde. Ich verwende das an

der Abteilung Mathematische Psychologie entwickelte Autorensystem "AdAsTra" sowie dessen eingebaute Programmiersprache, um den Übungskurs zu erstellen. Endergebnis sind alle benötigten Kursdateien, die in das Lehrsystem geladen und damit in HTML-Schulungen eingesetzt werden können.

Lerngebiet HTML Man kann mittlerweile relativ komplex aufgebaute Websites ohne jede Kenntnis der Sprache HTML gestalten.

Dafür gibt es "WYSIWYG"-Editoren, die ähnlich einem herkömmlichen Textverarbeitungsprogramm auf Tastendruck die gewünschte Funktionalität in ein Hypertext-Dokument einfügen. Der Designer bekommt sofort das Ergebnis zu sehen, ohne sich Gedanken um den zugrundeliegenden Quelltext machen zu müssen.

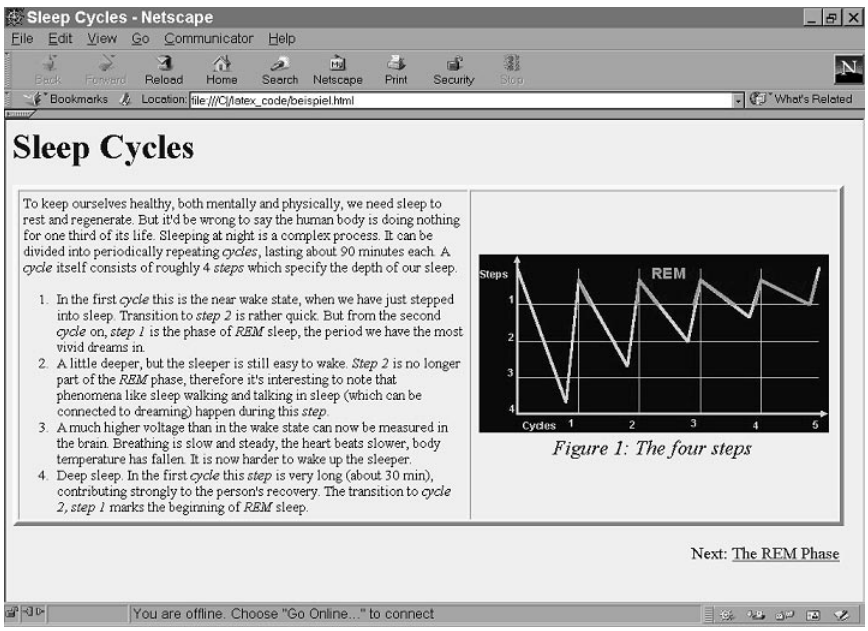


Abbildung 1.1: HTML-Seite in einem Browser

Für eine ordentliche Gestaltung der Seiten ist es allerdings hilfreich, sich zumindest mit dem Prinzip der wichtigsten HTML-Befehle vertraut zu machen. Noch mehr kommt es dem Entwurf zugute, wenn man die Quelldatei tatsächlich lesen und selbst Änderungen vornehmen kann, wobei ein selbstgeschriebener HTML-Code generell einsichtiger ist als ein automatisch generierter.

Will man irgendwann wirklich professionell aufgemachte WWW-Seiten erstellen und dabei HTML-Erweiterungen bzw. mit HTML kombinierbare

Sprachen (dynamisches HTML, JavaScript, Perl, VRML, CSS Style-Sheets usw.) einsetzen, dann ist ein Beherrschen der Grundlagen, eben des Schreibens von HTML-Dateien, unerläßlich.

Es lohnt sich also weiterhin, die Sprache HTML zu lernen.

HTML wird benutzt, um in einer einfachen Textdatei das Aussehen und Verhalten einer Hypertext-Seite zu beschreiben. Dieser Quellcode kann systemunabhängig von einem Browser interpretiert, d. h. übersetzt und dargestellt werden. Abb. 1.1 gibt ein Beispiel einer Hypertext-Seite.

In der HTML-Datei trifft man zuerst globale Festlegungen, wie z. B. den Titel des Dokuments. Darauf folgt der Seitentext, manipuliert durch diverse Kommandos (in spitzen Klammern `<...>` geschrieben, *Tags* genannt). Das Verändern der Schriftgröße für einen Teilbereich wäre ein Beispiel, das Anordnen in einer Tabelle oder das Unterlegen eines Textstücks mit einem *Hyperlink* (Verweis zu einer anderen HTML-Seite – lokal oder im Netz). Weiterhin können Grafiken und andere Dateiinhalte in den Text eingefügt werden.

Als Beispiel folgt jetzt die Quelldatei der Hypertext-Seite in Abb. 1.1 (mit Kommentarzeilen `<!--...-->`).

```
<html>

<head>
<title>Sleep Cycles</title>
</head>

<!-- globale Farbdefinitionen -->
<body text="#0000CC" bgcolor="#CCFFFF" link="#990000"
vlink="#330000" alink="#FF0000">
<!-- Ueberschrift -->
<h1>Sleep Cycles</h1>
<!-- Tabelle -->
<center>
<table border=5 cellpadding=5>
<!-- linke Spalte der Tabelle -->
<tr><td>
<!-- ab hier schwarze und kleinere Schrift -->
<font color="#000000" size=-1>
To keep ourselves healthy, both mentally and physically, we need sleep to
rest and regenerate. But it'd be wrong to say the human body is doing
nothing for one third of its life. Sleeping at night is a complex process.
It can be divided into periodically repeating <i>cycles</i>, lasting about
90 minutes each. A <i>cycle</i> itself consists of roughly 4 <i>steps</i>
which specify the depth of our sleep.
<!-- numerierte Liste mit 4 Elementen -->
<ol>
<li>
```

```
In the first <i>cycle</i> this is the near wake state, when we have just
stepped into sleep. Transition to <i>step 2</i> is rather quick. But from
the second <i>cycle</i> on, <i>step 1</i> is the phase of <i>REM</i>
sleep, the period we have the most vivid dreams in.
</li>
<li>
A little deeper, but the sleeper is still easy to wake. <i>Step 2</i> is
no longer part of the <i>REM</i> phase, therefore it's interesting to note
that phenomena like sleep walking and talking in sleep (which can be
connected to dreaming) happen during this <i>step</i>.
</li>
<li>
A much higher voltage than in the wake state can now be measured in the
brain. Breathing is slow and steady, the heart beats slower, body
temperature has fallen. It is now harder to wake up the sleeper.
</li>
<li>
Deep sleep. In the first <i>cycle</i> this <i>step</i> is very long (about
30 min), contributing strongly to the person's recovery. The transition to
<i>cycle 2, step 1</i> marks the beginning of <i>REM</i> sleep.
</li>
</ol>
<!-- Aufheben der Schriftmodifikationen -->
</font>
</td>
<!-- rechte Spalte der Tabelle -->
<td>
<center>
<!-- eingefuegte Bilddatei -->
<img src="figure1.jpg" alt="The four steps" hspace=5 vspace=5 height=212
width=416><br>
<!-- grosse Schrift -->
<font size=4><i>Figure 1: The four steps</i></font>
</center>
</td></tr>
<!-- Ende der Tabelle -->
</table>
</center>
<br>
<div align=right>Next:
<!-- ein Hyperlink -->
<a href="rem.html">The REM Phase</a>
</div>
</body>

</html>
```

HTML ist das Werkzeug, durch dessen Beherrschung man im Internet bunte, informative und interaktive Präsentationen von Wissen und Ideen

erstellen kann. Der zu entwickelnde computergestützte Kurs soll das Erlernen der wichtigsten Sprachelemente von HTML Version 4.0, welche in [Nef99] bzw. der Hypertext-Ausgabe [Mü98] beschrieben ist, unterstützen.

Wissensraumtheorie Allgemein geht man davon aus, daß ein Schüler, der sich weiterbilden möchte, „mittelschwere“ Lerninhalte bevorzugt.

Die Theorie der Wissensräume sagt mathematisch exakt, was für einen individuellen Lerner unter einer „mittelschweren“ Aufgabe zu verstehen ist: das Wissen, welches das Vorwissen der Person um den kleinstmöglichen bisher unbekannten Aspekt erweitert. Inhalte, die im Wissensumfang des Schülers bereits enthalten sind, wären zu leicht. Lernstoff, der weit über seine/ihre derzeitigen Fähigkeiten hinausreicht, wäre zu schwierig.

Die Wissensraumtheorie definiert die Begriffe des *Wissensinhalts (Item)*, des *Wissenszustands* für einen bestimmten Schüler und schließlich die Menge aller möglichen Wissenszustände, den *Wissensraum*; jeweils bezogen auf ein Wissensgebiet (die Domain[1]).

Sei beispielsweise das Lerngebiet ein kleiner Bereich der HTML-Programmierung, welcher das Manipulieren des Schriftstils (nur Größe und Farbe) für einen Textabschnitt umfaßt.

Man kann das Wissensgebiet in kleinere Lernziele aufspalten, wobei keine „gleichmäßige“ Verteilung erwünscht ist, sondern Aufgabenstellungen unterschiedlicher „Schwierigkeit“. Für jeden Wissensinhalt kann entschieden werden, ob ein bestimmter Schüler ihn beherrscht oder nicht.

1. **Tags**
   Wissen, wie in HTML Kommandos notiert werden
   `<`*Tag*`>...</`*Tag*`>`

2. **Schriftgröße relativ**
   Kenntnis des Wertebereiches für Schriftgrößen relativ zur Normalgröße
   `-2` bis `+4`

3. **Schriftgröße absolut**
   Kenntnis des Wertebereiches für absolute Schriftgrößen
   `1` bis `7`

4. **Schriften-Tag**
   Wissen, wie das Kommando zur Schriftmanipulation aussieht
   `<font ...>`

[1] nicht zu verwechseln mit dem gleichlautenden Begriff aus der Internet-Welt

5. **Farben**
   Kenntnis der hexadezimalen Darstellung von Farbwerten
   z. B. `#CCFFFF`

6. **Schriftgröße definieren**
   Befähigung zum Durchführen von Schriftgrößenmanipulationen
   z. B. `<font size=4>Figure 1: The four steps</font>`
   oder `<font size=+1>Figure 1: The four steps</font>`

7. **Schriftfarbe definieren**
   Befähigung zum Durchführen von Schriftfarbenmanipulationen
   z. B. `<font color=#000000>To keep ourselves...</font>`

Um das Wissen eines Schülers in dem Gebiet einschätzen zu können, betrachtet man die (endliche) Menge aller Items.

$$Q = \{Tags, relativ, absolut, Schrift, Farben, Schriftgroesse, Schriftfarbe\}$$

Der Wissenszustand[2] einer bestimmten Person ist nun gegeben durch alle die Inhalte, die sie/er beherrscht, z. B.

$$\{Tags, absolut, Schrift, Schriftgroesse\} \subseteq Q.$$

Die Zusammensetzung solcher Wissenszustände ist keine zufällige. Einige Items setzen die Kenntnis gewisser anderer Items voraus, bevor sie selbst verstanden werden können (aufeinander aufbauende Lerninhalte).

Zur Darstellung der Voraussetzungsbeziehungen innerhalb einer Wissensstruktur wird eine *Implikationsrelation* I („Voraussetzung für“) von $2^Q \setminus \{\emptyset\}$ nach Q eingeführt.

Das Paar

$$(\{Tags\}, Schrift) \in I$$

sagt aus: „Wenn ein Schüler schon nicht weiß, was Tags sind, dann kennt er/sie mit Sicherheit auch das Schriften-Tag nicht.“ Man nennt das eine *Standardformbehauptung* (SFB) mit *Prämisse* $\{Tags\}$ und *Konsequenz Schrift*.

Logisch würde man die Implikation als

$$\neg Tags \Rightarrow \neg Schrift$$

schreiben.

[2] entspricht dem „Lernermodell“ in der Terminologie der Intelligenten Tutoriellen Systeme

Ein Wissensinhalt kann *mehrere* Voraussetzungen haben, von denen schon die Unkenntnis *einer* bedeutet, daß der Inhalt garantiert nicht beherrscht wird.

Ein HTML-Programmierer wird z.B. nicht die Schriftfarbe verändern können, wenn er/sie das Schriften-Tag nicht kennt. Ebensowenig, wenn er/sie den Code zur Farbdefinition nicht anwenden kann. Das sind dann *zwei* Standardformbehauptungen,

$$\begin{aligned}&(\{Schrift\}, Schriftfarbe),\\ &(\{Farben\}, Schriftfarbe) \quad \in \quad I.\end{aligned}$$

Die zwei Logikausdrücke dazu können auch kürzer als eine ODER-Verknüpfung geschrieben werden:

$$\neg Schrift \vee \neg Farben \Rightarrow \neg Schriftfarbe$$

Genauso kann es Wissensinhalte mit mehreren Voraussetzungen geben, bei denen erst das Nicht-Beherrschen *aller* impliziert, daß ein Schüler diese Fähigkeit nicht besitzt.

Den Wert für die Schriftgröße relativ bzw. absolut angeben zu können, ist Voraussetzung für das Definieren der Größe. Hat man aber eine der beiden Formen nicht gelernt, kann man sich immer noch mit der anderen behelfen (vgl. Item Nr. 6).

Dies ist ein primitiver Fall von zwei unterschiedlichen Lernwegen, welche zum selben Ziel, dem übergeordneten Wissen, führen – formuliert in *einer* SFB mit zweielementiger Prämisse:

$$(\{relativ, absolut\}, Schriftgroesse) \in I$$

In der Schreibweise mit logischen Operatoren entspricht das einer UND-Verknüpfung.

$$\neg relativ \wedge \neg absolut \Rightarrow \neg Schriftgroesse$$

Die zwei extremen Auffassungen von Voraussetzungen sind beliebig untereinander kombinierbar.

Analog zur Schriftfarbe kann man nämlich die Schriftgrößendefinition als Ganzes ebenfalls nicht lernen, wenn man das Schriften-Tag noch nicht kennt.

Schriftgroesse bekommt eine weitere Prämisse,

$$(\{Schrift\}, Schriftgroesse) \in I,$$

und das (negierte) Item geht per ODER-Verknüpfung in den obigen Logik-Term ein:

$$(\neg relativ \wedge \neg absolut) \vee \neg Schrift \Rightarrow \neg Schriftgroesse$$

Die entstehenden komplexen Ausdrücke der gesamten Relation I stellt man übersichtlich in Form eines *UND/ODER-Graphen* dar.

In Abb. 1.2 ist er so gezeichnet, daß die Prämissen unten stehen und die Konsequenzen oben. Ein Pfeil ist als logische Implikation zu lesen. UND-Knoten kennzeichnen alternative Lernwege. Mehrere in einen Itemknoten eingehende Kanten entsprechen mehreren SFBen bzw. einer ODER-Verknüpfung (s. oben).

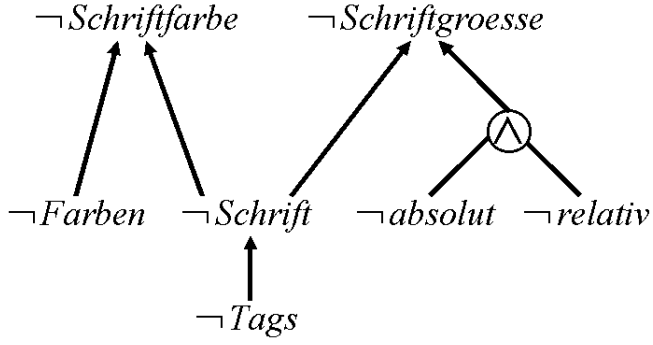


Abbildung 1.2: UND/ODER-Graph für einen HTML-Ausschnitt

Das Aufstellen der Implikationsrelation ist der entscheidende Modellierungsschritt, der getan werden muß, um dann das Wissen eines Schülers nicht nur beschreiben, sondern auch in das Gesamtsystem einordnen zu können. Die methodische Herleitung für das Lerngebiet HTML (mittels einer Variante der „Expertenbefragung“) findet in Kapitel 2 statt.

Wirklich alle Voraussetzungs-Implikationen $I^\diamond$ erhält man mittels *Abschluß* der Relation I bzgl. (verallgemeinerter) Reflexivität, z. B. dann auch

$$(\{Farben\}, Farben) \in I^\diamond,$$

und (verallgemeinerter) Transitivität, u. a.

$$(\{Tags\}, Schriftfarbe) \in I^\diamond$$

wegen

$$\begin{array}{c}(\{Tags\}, Schrift),\\ (\{Schrift\}, Schriftfarbe) \quad \in \quad I^\diamond.\end{array}$$

Diese *Menge der positiven Folgerungen* aus den gültigen SFBen $I^\diamond$ (defi-

niert in [Dow93]) ordnet[3] die Items gemäß ihrer Voraussetzungsbeziehungen und damit nach ihrer „Schwierigkeit".

Ein Schüler habe eine Anzahl von Wissensinhalten zu lernen, die er/sie nicht beherrscht. Dazu kommen dann immer sämtliche implizierten („schwierigeren") Inhalte; das sind diejenigen, die im UND/ODER-Graphen „darüber stehen".

Sein/ihr aktueller Wissenszustand kann keine Items beinhalten, für welche die notwendigen Vorraussetzungen („leichter zu lernende" Inhalte) selbst nicht enthalten sind. Die meisten der konstruierbaren Wissenszustandsmengen 2^Q treten somit bei keinem Schüler wirklich auf.

Interessant ist daher die Menge aller durch $I^\diamond$ ermöglichten Wissenszustände, der Wissensraum $\mathcal{K}$. Man berechnet ihn, indem man aus 2^Q die den Implikationen widersprechenden Zustände entfernt. Immer enthalten bleiben sowohl die leere Menge (ein Schüler, der noch nichts gelernt hat), als auch Q selbst (ein Schüler, der schon alles weiß).

Selbst für die kleine Beispiel-Domain ist der Wissensraum schon ziemlich unhandlich. Mit zunehmender Itemzahl wächst $|\mathcal{K}|$ sehr schnell. Um in einem Übungsprogramm Speicher zu sparen, berechnet man daher eine andere, dem Wissensraum äquivalente Datenstruktur.

Die *Basis* $\mathcal{B} \subseteq \mathcal{K}$ eines Wissensraumes (definiert in [Dow93]) ist die kleinste Menge, aus der man $\mathcal{K}$ noch herleiten kann: $\mathcal{B}$ enthält keine Wissenszustände mehr, die aus Vereinigungen der übrigen entstehen können. Bezüglich mengentheoretischer Vereinigung ist $\mathcal{K}$ abgeschlossen, deshalb bekommt man ihn immer aus

$$\mathcal{B} \cup (\text{sämtliche Vereinigungen aus Basiselementen})$$

zurück.

Die Basis erhält damit alle Informationen des UND/ODER-Graphen bzw. der Implikationsrelation I. Mehr noch: $\mathcal{B}$ kann ganz einfach daraus abgeleitet werden, indem man für jedes Item den Wissenszustand aus dem (nicht negierten) Item selbst und seinen (transitiven) Voraussetzungen bildet – bei alternativen Voraussetzungen wären das mehrere Zustände pro Item – und diese *für einen Wissensinhalt minimalen Wissenszustände* in einer Mengenfamilie zusammenfaßt.

Die Basis für den UND/ODER-Graphen in Abb. 1.2 hat folgende Gestalt

[3] strenggenommen keine Halb- oder Quasiordnung, weil diese ebenso wie Reflexivität und Transitivität nur für Relationen in einer Menge X, nicht von Menge X nach Menge Y, definiert sind („UND-Verknüpfungen stören")

(unterstrichen jeweils das Item, für welches der Zustand minimal ist).

$$\begin{aligned}\mathcal{B} = \{ \quad & \{\underline{Tags}\}, \{\underline{relativ}\}, \{\underline{absolut}\}, \{Tags, \underline{Schrift}\}, \{\underline{Farben}\}, \\ & \{Tags, relativ, Schrift, \underline{Schriftgroesse}\}, \\ & \{Tags, absolut, Schrift, \underline{Schriftgroesse}\}, \\ & \{Tags, Schrift, Farben, \underline{Schriftfarbe}\} \\ \}\end{aligned}$$

$\mathcal{K}$ würde zusätzlich alle Vereinigungen wie z. B. $\{Tags, relativ, Schrift\}$, insbesondere auch Q und $\emptyset$, enthalten.

Weiß man nun, welchem Element aus $\mathcal{K}$ das derzeitige Wissen eines Schülers entspricht, dann kann man ihm/ihr eine Aufgabe stellen, deren Bewältigung diesen Zustand um genau einen neuen Wissensinhalt erweitern würde, so daß wieder ein Element aus $\mathcal{K}$ entstünde (also um einen auf dem Vorwissen aufbauenden Inhalt).

Autorensystem AdAsTra Das eingesetzte Lehrsystem wurde und wird von Mitarbeitern und Diplomanden des Instituts für Psychologie, Abt. Mathematische Psychologie entwickelt. Zuletzt fand eine Portierung der Software nach *Java* statt, wobei einige wesentliche Features ergänzt wurden – näheres dazu in der Studienarbeit von Ralf Schreiber [Sch99].[4]

Die grundlegende Funktionalität von AdAsTra besteht in

- der *Wissensdiagnose* („Test“): Ermitteln des Wissenszustandes eines Schülers in einer Domain mittels zu beantwortender Übungsaufgaben (protokolliert),
- der eigentlichen Übung: Lernen neuer Wissensinhalte einer Domain, basierend auf dem vorher diagnostizierten Wissenszustand des Schülers (protokolliert),
- einer Dateiverwaltung für mehrere Schüler und verschiedene Domains,
- einer Entwicklungsumgebung (WYSIWYG-Editoren + Programmiersprache *Prola 2*) zum Erstellen der Aufgaben einer Domain mit
  - Frage (für Test und Übung),
  - Lösung (für Test und Übung),
  - Feedback (nur Übung)
  - Hilfestellung (die Erklärung/der instruktionale Teil; nur Übung).

[4] seitdem mehrfach erweitert, mit einer Internet-Variante im Entstehen

Für die Wissensstruktur HTML werden im Kapitel 3 unter Ausnutzung der multimedialen Gestaltungselemente und Interaktionsformen, welche AdAsTra zur Verfügung stellt, die Übungsaufgaben entworfen. Dabei müssen didaktische, lern- und motivationspsychologische Überlegungen und Anforderungen in Einklang mit der Realisierbarkeit eines $|Q| > 100$ Items umfangreichen Kurses gebracht werden.

Die in AdAsTra implementierten Algorithmen zur Auswahl der als nächstes zu stellenden Aufgabe (genauer: des als nächstes abzufragenden Items) im Test und in der Übung arbeiten auf der Basis des Wissensraumes. Diese muß dem Programm beim Anlegen eines neuen Kurses (Definieren der Items) in Form einer Matrix (b_{ij}) übergeben werden.

In den Zeilen $i = 1, \ldots, |\mathcal{B}|$ der Matrix stehen die minimalen Wissenszustände, die Spalten $j = 1, \ldots, |Q|$ repräsentieren die Items.

$$b_{ij} := \begin{cases} 0 & \text{Item } j \text{ nicht im Wissenszustand } i \text{ enthalten} \\ 1 & \text{Item } j, \text{ für das der Wissenszustand } i \text{ minimal ist} \\ 2 & \text{Item } j \text{ als Folgerung im Wissenszustand } i \text{ enthalten} \end{cases}$$

Es folgt die Basisdatei (vorangestellt die Angaben über die Dimension: Spaltenzahl; Zeilenzahl) zu dem durchgängigen Beispiel dieses Kapitels:

```
7
8
1000000
0100000
0010000
2001000
0000100
2202010
2022010
2002201
```

Die vorletzte Zeile etwa entspricht dem Basiselement

$$\{Tags, absolut, Schrift, \underline{Schriftgroesse}\}.$$

Matrix und AdAsTra/Prola-2-Skripte bilden den adaptiven HTML-Kurs.

Motivation Kaum eine technische Entwicklung hat unsere Gesellschaft in den letzten Jahren so stark verändert wie das Internet. Was einst einigen wenigen vorbehalten geblieben war, nämlich eigene Ideen und Wissen einer breiten Masse zugänglich zu machen, ist nun allgemein möglich geworden.

Sowohl privat als auch beruflich gehört der Informationsaustauch via World Wide Web für immer mehr Menschen zum Alltag.

Ein computergestützter Übungskurs für die Sprache HTML kann dazu beitragen, Interessenten einen schnellen und angenehmen Zugang zur Gestaltung von WWW-Seiten zu verschaffen; sie mit dem Werkzeug vertraut zu machen, das ihnen die Teilnahme am globalen Austausch von Ideen und Informationen ermöglicht.

Desweiteren wird das Entwicklungsergebnis als größere Anwendung der Wissensraumtheorie Indizien dafür liefern, welche Möglichkeiten und Vorteile die adaptiven Lernprogramme gegenüber herkömmlichen bieten. Die theoretischen Forschungsergebnisse der mathematischen Psychologie werden in einer Fallstudie aus der Praxis veranschaulicht, die Ideen in einem Praxisexperiment greifbar und eventuelle von der Theorie nicht berücksichtigte Aspekte sichtbar.

Mir persönlich bietet die Entwicklung eines HTML-Kurses auf Grundlage der Wissensraumtheorie die Gelegenheit, die Struktur eines interessanten Anwendungsgebiets, mit dem ich mich in der Freizeit beschäftigt habe, mit Hilfe von Modellen zu erfassen, um es so besser zu verstehen. Dann will ich das Wissen für andere Lernende verständlich aufbereiten und dabei die Denkweise der Schüler selbst beleuchten.

Das ist für mich Informatik: Die formale Darstellung und die automatische Verarbeitung von Informationen aus der Welt um uns herum. In dieser Sichtweise liegt mein Interesse begründet, interdisziplinäre Aufgabenstellungen, wie hier aus Psychologie, Mathematik, Pädagogik und Informatik, zu bearbeiten.

Ich möchte diese Verbindungen in meiner Arbeit herausstellen, indem ich neben dem Resultat auch die Überlegungen auf dem Weg dorthin beschreibe.

Kapitel 2

Wissensstruktur HTML

2.1 Die Struktur von "SELFHTML"

Methode Gesucht ist eine Implikationsrelation, die das umfangreiche Lerngebiet HTML in für einen Übungskurs hinreichender Weise repräsentiert.

Zur Modellierung eines Themengebietes und der Lernprozesse von Schülern darin werden verschiedene Ansätze verfolgt. Eine Auflistung findet sich in [Alb99]. Grundsätzlich wird dort zwischen drei Herangehensweisen unterschieden.

- *Datenanalyse*

  Versuchspersonen werden bereits fertige Aufgaben gestellt und aus den Antwortmustern die Implikationsrelation hergeleitet.

  Dies ist am ehesten praktikabel, wenn man viel Datenmaterial zu wenigen Aufgaben hat.

  Da es im Rahmen der Studienarbeit jedoch nicht möglich ist, umfangreiche Tests durchzuführen und aus der Vermutung heraus, daß eine große Anzahl Items benötigt wird, empfiehlt sich dieses Vorgehen nicht.

- *Systematische Aufgabenkonstruktion*

  Das zu erfassende Wissensgebiet wird unter denkpsychologischen Gesichtspunkten analysiert und in kleinste Teile zerlegt. Aus diesen *Komponenten* baut man systematisch Wissensinhalte (Aufgaben/Probleme) zusammen. Die erhaltenen mathematischen Strukturen (z. B. „Sequenzen" von Komponenten) können mittels einer angemessenen Implikationsrelation, die explizit definiert wird (z. B. „Sequenzen-Inklusion"), geordnet werden.

Das Verfahren führt zu einem äußerst präzisen Modell (wohldefinierte Items und Implikationen). Es besteht allerdings bei aller Detailgenauigkeit die Gefahr, daß Denkprozesse, die außerhalb des Modells liegen, komplett außer acht gelassen werden.

Diese Lösung käme sicher in Frage. Als Komponenten der Probleme (Teilziele des Problemlösevorgangs) könnten hier die syntaktischen Sprachelemente von HTML dienen. Aus diesen wären beliebig komplexe Aufgaben explizit konstruierbar.

- *Expertenbefragung*

  Experten in dem Wissensgebiet werden Standardformbehauptungen präsentiert: „Wenn eine Person die Wissensinhalte $\{p_1, ..., p_k\}$ (Prämisse) noch nicht gelernt hat, ist es dann sicher, daß sie/er auch den Wissensinhalt q (Konsequenz) noch nicht gelernt hat?" Die für zutreffend befundenen Behauptungen gehen in die Implikationsrelation ein: $(\{p_1, ..., p_k\}, q) \in I$.

  Die Expertenbefragung dringt zwar nicht so tief in Aufgaben- und Denkstrukturen vor, liefert aber eher einen umfassenden Gesamtblick „von oben" auf die Itembeziehungen.

  Für einen praxisnahen Kurs könnte die Anwendung dieser Methode die besten Resultate erbringen. Man würde dann nicht so sehr komplex zusammengesetzte Aufgaben präsentieren, sondern mehr in die Breite gehen, wobei möglichst viele Einzelthemen abgedeckt werden würden.

Zur Gewinnung von Wissensstrukturen wird an der TU Braunschweig die Expertenbefragung eingesetzt. Sie wird durch das institutseigene Tool "KQ_Win" unterstützt.

Als Experten müßte man einen HTML-Ausbilder gewinnen, mit dessen Hilfe man die Items entwerfen und welcher dann im Dialog mit KQ_Win die Implikationsrelation aufstellen würde.

Geht es vielleicht einfacher? Was wäre, wenn bereits eine von Experten aufgestellte Wissensstruktur vorläge, die man relativ unproblematisch in die gewünschte Implikationsrelation transformieren könnte?

Es gibt ja eine Reihe von Lehrbüchern über HTML! Kann von dem konzentrierten Expertenwissen in Buchform auf eine implizit enthaltene Itemmenge und Relation geschlossen werden? Ohne größeren Aufwand sicher nur aus übermäßig gut gegliederten Dokumenten, schließlich sind Lernziele unterschiedlicher Schwierigkeit und vielfältige Voraussetzungsbeziehungen dazwischen gesucht.

Ein strukturiertes Lehrwerk für die HTML – und zwar in Hypermedia-Form, wo die Lernabschnitte mittels Hyperlinks gemäß ihrer Voraussetzungen verknüpft sind – existiert: "SELFHTML" [Mü98]. Meines Wissens haben viele deutschsprachige HTML-Programmierer mit diesem Standardwerk die Sprache erlernt, so daß der Autor Stefan Münz als einer der besten HTML-Experten im oben genannten Sinne gelten dürfte.

Ich werde also die im weiteren zu verwendende Wissensstruktur aus der Dokumentenstruktur von SELFHTML extrahieren[1], was einer Variation der Expertenbefragung in Richtung (manueller) Dokumentenanalyse oder Informations- bzw. Wissensextraktion entspricht.

Lernziele Man sucht zunächst nach einer Einteilung des Wissensgebietes in kleine Lernabschnitte variabler Schwierigkeit.

SELFHTML ist enorm groß, so daß eine Beschränkung auf die grundlegenden Aspekte von HTML nötig ist, um den Übungskurs im Rahmen einer Studienarbeit realisierbar zu halten. Gleichzeitig muß aber die praktische Relevanz des Kurses halbwegs gegeben sein.

Die erste Einschränkung liegt darin, nur Lerninhalte des Kapitels **HTML** zu verwenden. Dieses ist per Hyperlinks in Unterkapitel gegliedert: Abb. 2.1.

HTML	$\longrightarrow$	**Allgemeine Regeln für HTML**
	$\longrightarrow$	**Allgemeine Angaben zur Datei**
	$\longrightarrow$	**Dateiweite Farben und Hintergrund**
	$\longrightarrow$	**Absatztypen und Textgestaltung**
	$\longrightarrow$	**Tabellen**
	$\longrightarrow$	**Verweise – Hyperlinks**
	$\longrightarrow$	**Grafik in HTML**
	$\longrightarrow$	**[Formulare]**
	$\longrightarrow$	**[Frames – mehrere Bildschirmfenster]**
	$\longrightarrow$	**[Multimedia-Ojekte in HTML]**
	$\longrightarrow$	**[Layer – positionierbare Elemente]**
	$\longrightarrow$	**[Weiterführende HTML-Elemente]**
	$\longrightarrow$	**[Universalattribute]**

Abbildung 2.1: Obere Gliederungsebene in SELFHTML

[1] Der entstehende Übungskurs soll die analysierten SELFHTML-Kapitel nicht „ersetzen"; der erklärende Teil wird wesentlich kleiner sein. Er ist andererseits auch nicht „redundant", denn er wird über ein viel höheres Maß an Interaktion verfügen. Ich werde teilweise Itembezeichnungen übernehmen; Texte, Beispiele und Aufgaben jedoch vollständig selbst entwerfen (Kap. 3).

Die eingeklammerten Bereiche sehe ich als spezielleres Wissen an. Sie sollen nicht weiterverfolgt werden.

Diese Entscheidung fällt bei den Kapiteln **Formulare** und **Frames** besonders schwer, sind doch gerade dies für schon fortgeschrittene Lerner die interessanten Gestaltungselemente für professionellere Präsentation und Web-Interaktion.

Beide Themen sind aber relativ umfangreich, und der Umgang mit ihnen erfordert einerseits wohlüberlegte Designentscheidungen bei Verwendung der von vielen Anwendern ungeliebten Frames, andererseits vertiefte Programmierkenntnisse zumindest für die "CGI"-Verarbeitung von Formularen.

Klickt man im SELFHTML-Dokument nun auf einen der Links (beispielsweise den unterstrichenen), so gelangt man auf eine tiefere Ebene: Abb. 2.2.

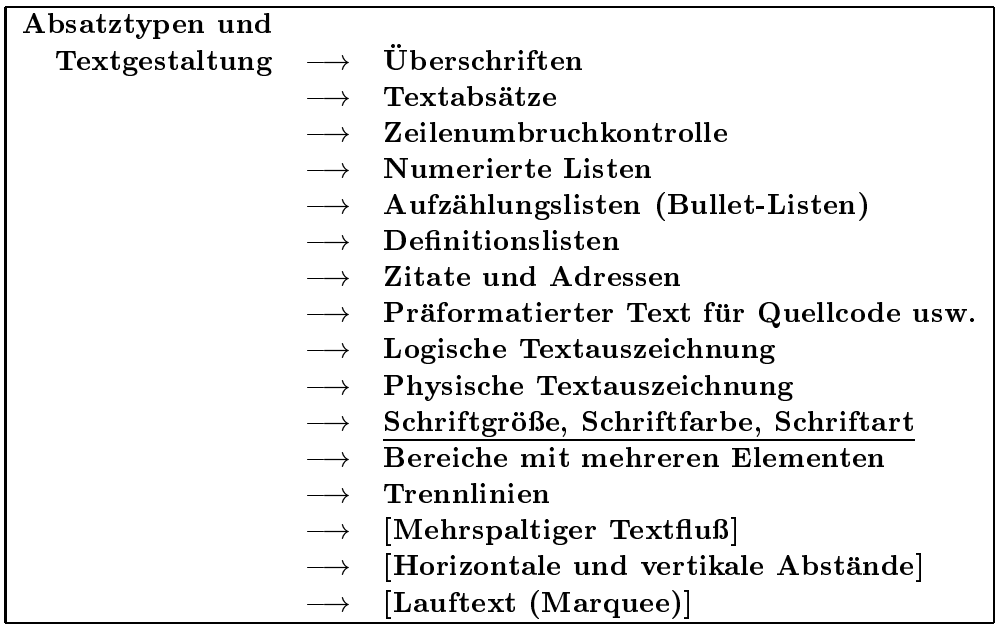


Abbildung 2.2: Mittlere Gliederungebene in SELFHTML

In allen sieben Lernbereichen (**Allgemeine Regeln für HTML**, **Allgemeine Angaben zur Datei** usw.) werden auf der mittleren Ebene wieder nicht so wichtige Fäden fallengelassen, um die Itemzahl handhabbar zu halten.

Die letzten drei Verzweigungen von **Absatztypen und Textgestaltung** etwa führen zu für das Lernprogramm allzu speziellem Wissen, denn die Kommandos werden nur von jeweils einem Browser (mal "Netscape Navigator", mal "Microsoft Internet Explorer") interpretiert.

Durch einen weiteren Klick (in diesem Bereich z.B. auf **Schriftgröße, Schriftfarbe, Schriftart**) verzweigt man auf die dritte und tiefste Ebene: Abb. 2.3.

Schriftgröße, Schriftfarbe, Schriftart	$\longrightarrow$	**[Allgemeine Hinweise]**
	$\longrightarrow$	**Schriftgröße definieren**
	$\longrightarrow$	**Schriftfarbe definieren**
	$\longrightarrow$	**Schriftart definieren**
	$\longrightarrow$	**Normalschriftgröße definieren**
	$\longrightarrow$	**[Normalschriftfarbe definieren]**
	$\longrightarrow$	**[Normalschriftart definieren]**

Abbildung 2.3: Untere Gliederungebene in SELFHTML

Auch hier werden Themen (erneut unter dem Aspekt der Wahrung von Systemunabhängigkeit) ausgeklammert.

Normalschriftfarbe definieren und **Normalschriftart definieren** werden von dem weit verbreiteten Netscape-Browser nicht verstanden. Man kann die gleichen Effekte auch über **Schriftfarbe definieren** und **Schriftart definieren** erreichen.

Hinter den untersten Links verbergen sich die eigentlichen Instruktionseinheiten von SELFHTML. Hier finden sich ausführliche Darstellungen der Lerninhalte.

Zum Thema **Schriftfarbe definieren** gibt es textuelle Beschreibungen und Erläuterungen, ein Codebeispiel, die Darstellung im Browser sowie ergänzende Hinweise zur Benutzung des Kommandos.

Von diesen kleinen Lernzielen umfaßt das **HTML**-Kapitel in seiner Gesamtheit weit über 200. Daraus kann der Hauptteil der Wissensinhalte im Sinne der Wissensraumtheorie ausgewählt werden.[2]

Außerdem werden die zugehörigen Überschriften der mittleren und oberen Ebene für Items höherer und höchster Schwierigkeit übernommen.

Das läßt schon vermuten, daß nicht nur die Lerninhalte, sondern auch die Voraussetzungsbeziehungen dazwischen aus dem SELFHTML-Dokument extrahiert werden können. Und tatsächlich kann man drei verschiedene Arten der Ordnung von Lerninhalten ausfindig machen.

[2]Es zeigt sich schon, daß der in Kapitel 1 vermutete Wissensstruktur-Ausschnitt zu detailliert gewählt war: Es wird kein **Schriften-Tag** mehr separat als Voraussetzung für **Schriftfarbe definieren** auftreten. Dafür wird das vermutete Item **Tags** (Abschnitt **Tags (Steuerbefehle in HTML)**) differenzierter betrachtet werden. **Schriftgröße relativ** und **Schriftgröße absolut** gehen beide in ein Größen-Item ein; dagegen wird es zur hexadezimalen Farbdefiniton ein alternatives Item geben.

Baumstruktur Die Lernabschnitte in SELFHTML sind, wie in Abb. 2.1 – 2.3 gesehen, hierarchisch verlinkt.

Man kann die verschiedenen Überschriften auf den drei Gliederungsebenen als einen Baum darstellen, wobei die Kanten den jeweiligen Links zwischen den Ebenen entsprechen und in den Knoten die Titel der Lernabschnitte stehen: Abb. 2.4 (zuvor verfolgter Pfad unterstrichen).

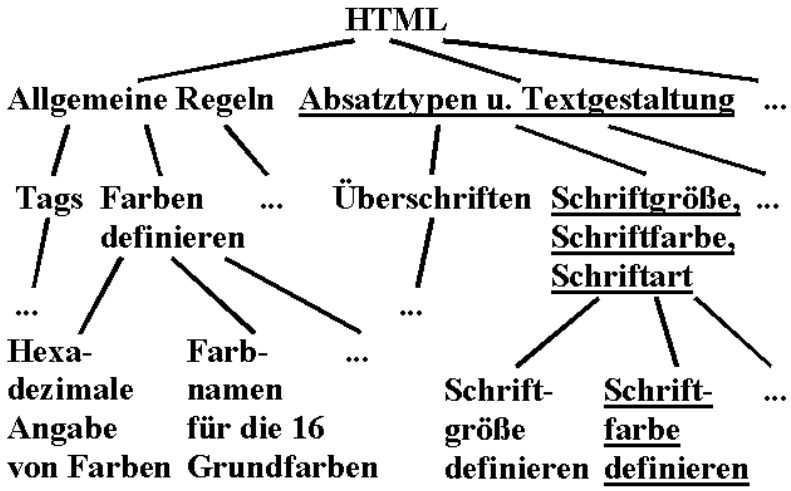


Abbildung 2.4: Die Baumstruktur von SELFHTML

Die Blätter, entnommen aus der unteren SELFHTML-Gliederungsebene (instruktionale Texte), machen den Großteil der Begriffe aus. Deren Eltern- und Großelternknoten (aus den beiden höheren Ebenen) entsprechen solchen Lernabschnitten, die das ihnen untergeordnete Wissen zusammenfassen.

Wenn jemand beispielsweise die kompletten Abschnitte unter **Schriftgröße, Schriftfarbe, Schriftart** gelernt hat, dann beherrscht er/sie dieses schwierigere Wissen (mittlere Ebene) und gleichzeitig die Einheiten der Kinderknoten (untere Ebene).

Oder besser umgekehrt formuliert: Ein Eltern-Wissensabschnitt (mittlere Ebene) kann bei Unkenntnis der einzelnen Kinderknoten (untere Ebene) *mit Sicherheit nicht beherrscht werden* (er verkörpert ja daraus kombiniertes Wissen).

Diese Formulierung entspricht genau den in der Wissensraumtheorie verwendeten Voraussetzungsbeziehungen (SFBen). Ob nun jeder Kinderknoten bekannt sein muß (Normalfall, da verschiedene Lerninhalte zusammengefaßt werden) oder austauschbare HTML-Elemente (also UND-Verknüpfungen) darunter sind, muß in jedem Einzelfall festgestellt werden.

Inhalte von stark unterschiedlicher Schwierigkeit sind ein Gewinn für die spätere Wissensdiagnose. So können komplexere Aufgaben gestellt werden, die bei richtiger Beantwortung viele andere Aufgabenstellungen einsparen, nämlich den kompletten Unterbaum des dem Item entsprechenden Knoten.

Geschwisterordnung Neben der „vertikalen“ Baumhierarchie von Lerninhalten läßt sich zwischen einigen Blättern noch eine „horizontale“ Hierarchie ausmachen.

Die Inhalte sind dort nach Schwierigkeit geordnet, meist indem das erste Blatt Voraussetzung für seine Geschwister ist. Dort wird das grundlegende Tag erklärt – z. B. **Numerierte Listen definieren** – welches in den Geschwisterknoten – **Listen alphabetisch numerieren**, **Numerierung beeinflussen**... – durch zusätzliche Tags oder *Attribute* erweitert wird, die man ohne seine Kenntnis nicht lernen kann.

Eine solche Reihenfolge von Wissenseinheiten kann auch mehrstufig auftreten: **Tabellen definieren** ist Voraussetzung für **Zeilen und Spalten definieren** ist Voraussetzung für **Spalten vordefinieren**.

Leider ist diese Sorte von Beziehungen (mit der Ausnahme **E-Mail-Verweise**) nicht explizit im Dokumenttext gekennzeichnet. Man entdeckt die wenigen Fälle aber beim Durchgehen des Dokuments, wenn das erste verlinkte Blatt (unterste Ebene) ein grundlegendes Tag zum Inhalt hat (meist ein „XY definieren“).

Trägt man die horizontalen Verknüpfungen als Kanten in den Baum ein, entstehen Kreise, und er wird damit zum Graph – s. Abb. 2.5.

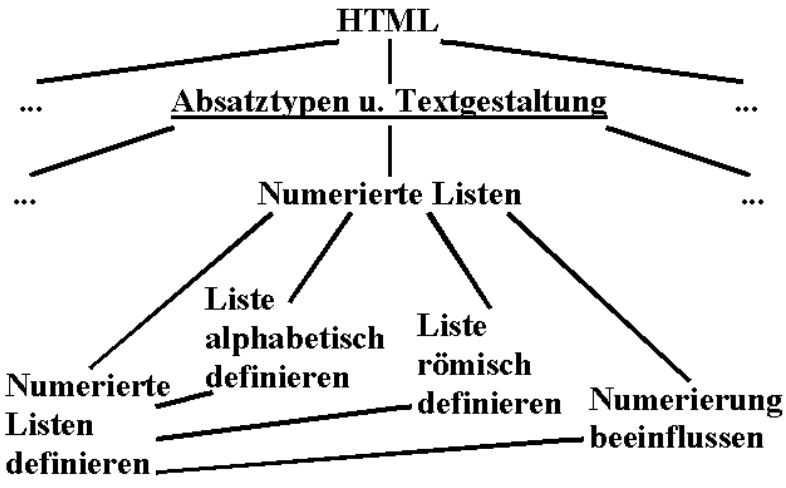


Abbildung 2.5: Geschwisterordnung in SELFHTML

Querverweise In den instruktionalen Erläuterungen sowie den Einleitungen zu den Linklisten der höheren Ebenen finden sich weitere Links, die über Eltern-Kinder-Verbindungen hinausgehen und Knoten aus disjunkten Teilgraphen miteinander in Beziehung setzen.

Typische Formulierungen in SELFHTML (zitiert aus [Mü98]):

1. in **Schriftfarbe definieren**: „Zur Bedeutung der Farben und für Hinweise zur Farbauswahl lesen Sie **Farben definieren in HTML**."

   Eine konkrete neue Veraussetzung (man kann **Schriftfarbe definieren** ohne Kenntnis von **Farben definieren in HTML** nicht lernen); sie ist in Abb. 2.6 eingezeichnet (gestrichelte Kante).

2. in **Absatztypen und Textgestaltung**: „Sinnvolle Vorkenntnisse zum Verständnis: **Allgemeine Regeln für HTML**."

   Dies ist eine etwas vage Voraussetzung (Abb. 2.6). Einige Kinderknoten von **Allgemeine Regeln für HTML**, die auf transitivem Wege dann auch Voraussetzungen wären, müssen nicht notwendigerweise beherrscht werden (z. B. **Kommentare und Credits** oder **Umlaute, Sonderzeichen und HTML-eigene Zeichen**) und andersrum gesehen sollen die Voraussetzungen auch und gerade für die Knoten im Unterbaum von **Absatztypen und Textgestaltung** gelten.

3. in **Schriftfarbe definieren**: „Wenn Sie Schriftfarben mit Hilfe von CSS Style-Sheet-Angaben definieren wollen, müssen Sie den Abschnitt **Style-Sheets definieren** kennen. Dann können Sie die **CSS-Angabe für Textfarbe** richtig einsetzen."

   Hierin stecken schon verwertbare Informationen über alternative Lernwege – in diesem Fall (und da muß man genau hinsehen!) **CSS-Angabe für Textfarbe** (Beispiel nicht eingezeichnet, weil Style-Sheets nicht Gegenstand des Übungsprogramms sein werden).

4. in **Schriftfarbe definieren**: „Die definierten Textfarben sollten mit der dateiweit definierten **Hintergrundfarbe** kontrastieren."

   Das ist eine hilfreiche Bemerkung mit Angabe eines Referenzabschnitts, jedoch irrelevant für die Voraussetzungsbeziehungen, d. h. sie bekommt keine Kante im Graphen. Ebenso gilt es, Vorgriffe auf schwierigere Kapitel zu ignorieren.

Im Zweifelsfall sollte man einen Link besser nicht zu den Voraussetzungen zählen, denn die Definition von SFBen verlangt, daß bei Unkenntnis der Prämisse *mit Sicherheit* die Konsequenz nicht beherrscht wird. Viel zu leicht

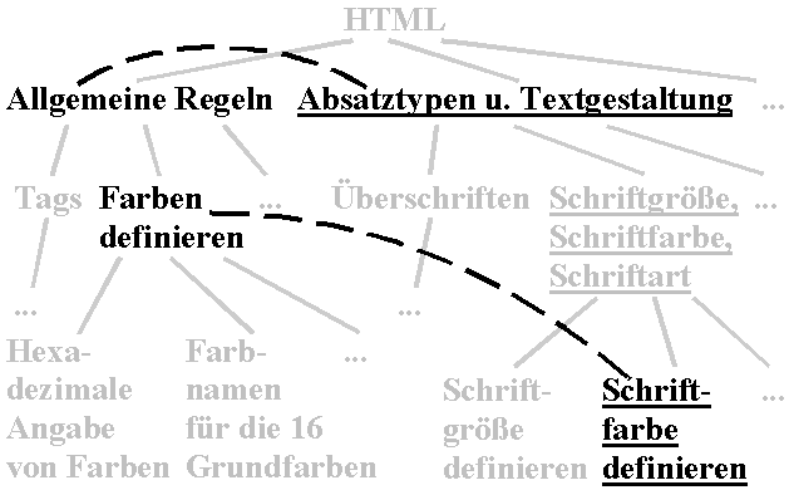


Abbildung 2.6: Querverweise in SELFHTML

kann es passieren, daß man Lernverhalten falsch beurteilt und für das Modell zu strenge Annahmen trifft. SELFHTML (oder ein Lehrbuch allgemein) mag zuerst an dem Thema selbst und erst dann an den Denkprozessen der Schüler orientiert sein.

Dennoch: An einigen offensichtlichen Stellen – wo eine Voraussetzung (Typ 1) zwar genannt, aber nicht mit einem Link versehen ist – ist es sinnvoll, dem Graphen „eigene“ Voraussetzungskanten hinzuzufügen.

Wenn man lernen soll, daß **Start- und Endtags verwenden** zum guten Stil gehört, muß man zunächst einmal wissen, was **Tags in HTML** (eingeschlossen Start- und Endtags) überhaupt sind.

Gleiches gilt für Erweiterungen des **Tabellen definieren** (Voraussetzung) wie **Dicke des Außenrahmens** und Erweiterungen des **Zeilen und Spalten definieren** wie **Zellen in einer Zeile spaltenweise verbinden**.

Mit Grafiken verhält es sich ebenso, die Grundvoraussetzung heißt **Grafik referenzieren (allgemein)**.

Die formale Übersetzung des analysierten Dokumentes (der vollständige Graph wird aus Platzgründen nicht explizit angegeben) in die Sprache der Wissensraumtheorie geschieht im folgenden Abschnitt.

2.2 Aufstellen der Implikationsrelation

Itemmenge Ein Knoten im Baum Abb. 2.4 entspricht i. a. einem Item in der Wissensstruktur.

Ausgenommen ist die Wurzel **HTML**; man kann das komplette Wissen über die Sprache HTML nicht in einer Aufgabe abfragen.

Um die Itemzahl nicht unnötig groß werden zu lassen, nehme ich (zusätzlich zu den aus SELFHTML schon nicht übernommenen Themen) noch weitere Veränderungen an dem Baum vor.

Wo die Inhalte sehr ähnlich bzw. irgendwie anders zusammengehörig oder einzeln wenig umfangreich sind, werden Knoten zusammengefaßt, z. B. das Blatt **Farbnamen für die 16 Grundfarben** mit dem Blatt **Speziellere Farbnamen** verschmolzen.

Informationsverteilung und Dateiorganisation aus dem Unterbaum von **Allgemeine Regeln für HTML** bleibt ein einzelnes, aus der mittleren Ebene gewonnenes Item – also ein Blatt ohne Großeltern! – das den Lernstoff seiner zwei Kinderknoten umfaßt. Es lohnt sich nicht, aus beiden selbst noch Blätteritems zu machen.

Einige Blattbeschriftungen tauchen in verschiedenen Teilbäumen in ähnlicher Form auf, z. B. das „Ausrichten“ von Objekten. Diese Sprachelemente werden in SELFHTML nicht einmal zentral, sondern für jeden Anwendungsfall wieder erklärt.

Ob man aber Überschriften oder Grafiken ausrichtet, beides funktioniert über das `align`-Attribut. In der zu konstruierenden Wissensstruktur kann deshalb ein einziges Item *Ausrichten** daraus gemacht werden. Eine konkrete Anwendung (mit eventuellen Eigenheiten) geht dabei nicht verloren, sie findet sich immer noch in dem jeweiligen (kombinierten) Item des Elternknotens wieder.

Häufige Parallelen zeigen sich in der Behandlung von Tabellen und Grafiken. Beim Auswerten auf der mittleren Ebene wurde *BeschriftenUmfliessen* für Grafik zu einem Inhalt zusammengefaßt, so daß es mit dem weitestgehend entsprechenden Item *BeschriftungAusrichtung* für Tabellen korrespondiert. *Rahmen** und *Textfortsetzung** gelten für Grafik und Tabellen gemeinsam.

Manchmal macht es sogar Sinn, ein Blätteritem aufgrund von vorhandenem Wissen im Elternknoten ganz aufzugeben, und zwar in dem Fall, wo ein mittlerer Knoten nur zwei Kinder hat, die in einer Geschwisterordnung zueinander stehen.

Ein Beispiel: Die Items für **Definitionslisten definieren** und **Liste kompakt darstellen** brauchen kein künstliches Elternitem mit dem Wissen aus beiden, weil die Definition „kompakter“ Definitionslisten das Wissen über die Definition normaler komplett mit einschließt bzw. erweitert (also auch voraussetzt) und somit selbst das Elternitem sein kann (sonst mit diesem inhaltlich äquivalent wäre).

Allgemeiner kann das geschehen, wenn *ein* Blatt existiert, dessen Inhalt mehrere – *alle* – Geschwisterinhalte zusammenfaßt. Dies ist u. a. bei **Vor-**

definierte Angaben kombinieren der Fall, wo gezeigt wird, wie man alle zuvor erläuterten Voreinstellungen für E-Mail-Links in ein Tag schreiben kann. Diesen Inhalt verkörpert dann ebenfalls das Elternitem.

Nach all diesen Überlegungen kommt die Itemmenge Q zustande – aufgelistet in Anhang A.1 gemäß eines Präordnungsdurchlaufs des modifizierten Baumes.

SFBen aus der Baumstruktur Durch die Modifikationen im vorhergehenden Abschnitt wurden aus dem Baum sieben disjunkte Teilbäume (einer für jedes SELFHTML-Hauptkapitel) mit genau einem Knoten pro Item in der Wissensstruktur (ausgenommen die mit „*“ versehenen Items, z. B. *Ausrichten**, welche hier mehr als einmal durch einen Knoten repräsentiert sind).

Die Bäume werden erneut in Präordnung durchlaufen, diesmal ohne die Blätter.

Für jede Wurzel eines Unterbaumes (obere und mittlere Ebene) werden die zugehörigen Voraussetzungsbeziehungen zu ihren Kinderknoten gemäß Tabelle 2.1 in Standardformbehauptungen übersetzt.

Baumbeziehung	SFB(en)	Logikausdruck
s ist einziges Kind von **r**	$(\{s\}, r) \in I$	$\neg s \Rightarrow \neg r$
s, **t** sind Kinder von **r**; unbedingte Lernvoraussetzungen	$(\{s\}, r), (\{t\}, r) \in I$	$\neg s \vee \neg t \Rightarrow \neg r$
s, **t** sind Kinder von **r**; alternative Lernwege	$(\{s, t\}, r) \in I$	$\neg s \wedge \neg t \Rightarrow \neg r$

Tabelle 2.1: Übersetzungsregeln Baumstruktur in SFBen

„Unbedingte Voraussetzungen“ bedeutet, daß beide Wissensinhalte (Kinderknoten) in dem übergeordneten Wissen (Elternknoten) enthalten sind (Normalfall); „alternative Lernwege“, daß eine Aufgabe zu der Konsequenz auf zwei verschiedene Arten mit jeweils dem Wissen eines der Prämissenelemente gelöst werden kann (z. B. *Schriftgroesse/Normalschriftgroesse* oder *hexadezimal/Farbnamen*).

Natürlich gilt die Verallgemeinerung auf n Kinderknoten ($n \geq 1$). Dann entsteht zu der Wurzel eine Menge von SFBen, eine vielelementige Prämisse oder eine Mischung aus beidem, was durch komplexe Logikterme mit sowohl ODER- als auch UND-Verknüpfungen beschrieben werden kann.

Desweiteren sollte eine Kind-Eltern-Beziehung dort unübersetzt bleiben, wo rein intuitiv das Voraussetzungsverhältnis zu schwach ist (beispielswei-

se ist **Guter HTML-Stil** keine zwingende Voraussetzung für die Beherrschung der Programmierung von Hypertextseiten in dem Sinne, daß man bei Unkenntnis andere Inhalte nicht mehr lernen kann) oder wenn die vielen Prämissen nicht in einer einzigen übergeordneten Aufgabe vereinbar erscheinen (wobei man allerdings mit der Verkleinerung von schwierigeren Aufgaben fortgeschrittenen Lernern Möglichkeiten zum Abkürzen des Diagnose- und Lernvorgangs nimmt).

Durch Anwendung des Verfahrens entsteht die erste Hälfte der Implikationsrelation I (s. Anhang A.1).

Abb. 2.7 stellt einen Ausschnitt aus I (analog zu Abb. 2.4) als einen UND/ODER-Graphen dar. Insgesamt handelt es sich schon nicht mehr um einen Baum bzw. Bäume, da die mit „*“ markierten Items (*Ausrichten** usw.) nur noch jeweils einmal als Knoten und mit Kanten zu Inhalten aus unterschiedlichen Kapiteln auftreten.

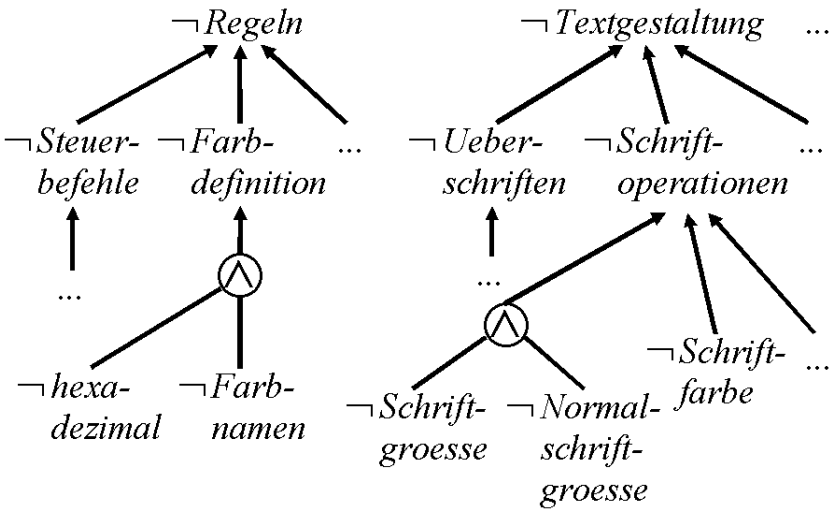


Abbildung 2.7: UND/ODER-Graph aus der Baumstruktur

SFBen aus Geschwisterordnung Sämtliche „horizontalen“ Kanten können direkt als „unbedingte Lernvoraussetzungen“ in SFBen umgesetzt werden. Tabelle 2.2 gibt die entsprechende Übersetzungsregel an.

Die Erweiterung t der Definition s darf kein „*“- Item sein! Diese beinhalten allein das Wissen über ein allgemein einsetzbares Attribut, nicht über ein spezielles Tag, auf welches sich das Attribut bezieht.

Mehrstufige Geschwisterordnungen werden durch mehrfaches Anwenden der Tabelle übersetzt.

Reihenfolge	SFB(en)	Logikausdruck
t erweitert Geschwisterwissen **s**; gemeinsamer Elternknoten **r**	$(\{s\}, t) \in I$ $(\{s\}, r) \notin I$	$\neg s \Rightarrow \neg t$

Tabelle 2.2: Übersetzungsregel Geschwisterordnung in SFBen

Beim Übersetzen entsteht Redundanz: In der ersten Hälfte von I sind SFBen enthalten, die sich nach Berücksichtigung von Geschwisterordnungen auch transitiv ergeben würden und somit in $I^{\diamond}$ enthalten wären. Sie können – sofern sie nicht Argument einer UND-Verknüpfung sind – wie in Tabelle 2.2 angegeben aus I entfernt werden.

Der UND/ODER-Graph in Abb. 2.8 illustriert das Auflösen von entstandenen Redundanzen und wie so aus „horizontalen“ Beziehungen neue Hierarchien werden. Als Beispiel dienen wieder die *Nummerlisten* (vgl. Abb. 2.5; *Nummertyp* ist ein aus **Liste alphabetisch definieren** und **Liste römisch definieren** zusammengesetzter Itemknoten).

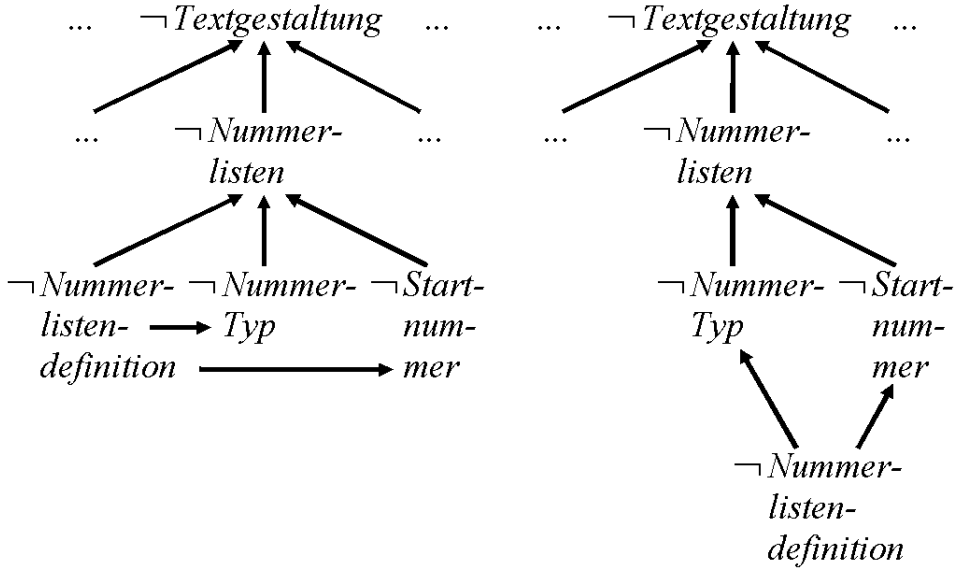


Abbildung 2.8: UND/ODER-Graphen mit Geschwisterordnung

Zusammengefaßt sorgt Teil zwei des Verfahrens über das Hinzufügen und Herausnehmen von SFBen für einige Umstellungen in der ersten Hälfte der

Implikationsrelation I (vgl. Anhang A.1).

SFBen aus Querverweisen Der (entsprechend der Itemmenge modifizierte) Graph Abb. 2.6 wird in Präordnung entlang der ursprünglichen Baumkanten durchlaufen. Eine besondere Bedeutung kommt diesmal den Blättern zu, denn die Implikationen werden vorwiegend aus Links in den instruktionalen Texten gewonnen.

Für jeden durchlaufenen Knoten ist zu prüfen, ob er einen Querverweis (neue, gestrichelte Kante) enthält. Im Dokument (Erläuterungen auf Blätterebene bzw. der einführende Satz zu einer Linkliste „Sinnvolle Vorkenntnisse zum Verständnis...“) liest man die zugehörige Formulierung nach. Gemäß Tabelle 2.3 übersetzt man diese in SFBen, was den UND/ODER-Graphen (Abb. 2.7 und 2.8) um neue Kanten erweitert.

Verweiskante	SFB(en) und Logikausdruck
r verweist auf **t**; Formulierungstyp 1	$(\{t\}, r) \in I$ $\neg t \Rightarrow \neg r$
r verweist auf **Allgemeine Regeln**, hat im Baum die Blätter $\mathbf{r}_i$ $(i = 1, 2, \ldots, n)$; Spezialfall Typ 2	$(\{Regeln\}, r_i) \in I$ für alle $i \in \{1, 2, \ldots, n\}$ (wenn $\{Regeln\}$ nicht bereits durch eine andere Prämisse von r_i transitiv Voraussetzung wird) $(\{Sonderzeichen\}, Regeln)$, $(\{Farbdefinition\}, Regeln)$, $(\{Dateikonventionen\}, Regeln) \notin I$ $\neg Regeln \Rightarrow \neg r_i$
s verweist auf **t** mit gemeinsamem Ziel **r**; Formulierungstyp 3	$(\{s, t\}, r) \in I$ $(\{s\}, r), (\{t\}, r) \notin I$ $\neg s \wedge \neg t \Rightarrow \neg r$

Tabelle 2.3: Übersetzungsregeln Querverweise in SFBen

Für den Formulierungstyp 1 ist die Umsetzung einfach; die zu verbindenden Itemknoten stehen im Text.

Typ 2 gibt zwar nur schwammige Informationen, so daß eigentlich jeder Einzelfall auseinandergenommen werden müßte. Nach genauerem Hinsehen hat sich aber gezeigt, daß es sich bei allen Voraussetzungen dieses Typs (innerhalb des betrachteten Ausschnitts von SELFHTML) um denselben Fall handelt: ein Link zu **Allgemeine Regeln für HTML**, und dieser steht am Anfang jedes der übrigen Kapitel.

Man kann daher allen Blätteritems dieser Kapitel die *Regeln* als eine Voraussetzung (eine SFB) zuordnen, ohne deren Beherrschung jedes einzelne nicht gelernt werden kann. Für die Nachfolgerknoten (Eltern und Großeltern) gilt die Voraussetzung dann transitiv.

Die Einschränkung in Klammern schließt für diese Regel die Vorwegnahme transitiv ohnehin gültiger Implikationen aus. Die meisten aus Formulierungstyp 1 gewonnenen Prämissen werden ja selbst von den *Regeln* abhängig sein.

Zur Lösung des Problems, daß *Regeln* mit all seinen Kinderknoten eine zu große Voraussetzung für so viele andere wäre, wurde der „Trick“ angewandt, alle Implikationen zu diesem Item aus I zu entfernen, die für die übrigen nicht absolut zwingend sind (im ganzen drei). Das richtet in diesem Spezialfall keinen größeren Schaden an.

Es brauchen dann beim Übersetzen von Formulierungen des Typs 1 keine Items „unter“ *Regeln* mehr als Prämissen berücksichtigt werden (wie etwa *Grundgeruest* für *Titel*). Sie gelten schon transitiv für die übrigen Items.

Ebenso muß dieselbe Prämisse nicht für alle Elemente einer Geschwisterordnung definiert werden (wie *Linkaufbau* für die Items 117 – 119), beim ersten genügt sie.

Eine geringe Restredundanz wird sich aufgrund mangelnder Überschaubarkeit des Gesamtgraphen jedoch vermutlich nicht vermeiden lassen.

Beim Formulierungstyp „Alternative“ (3) schließlich müssen bestehende SFBen geändert, nämlich ihre Prämissen erweitert werden.

Ein Beispiel ist die Ergänzung von

$$(\{\mathit{Grafikbeschriften}\}, \mathit{BeschriftenUmfliessen}) \quad \text{(Item 139)}$$

um die Alternative *TextGrafik* (Item 105, Kinderknoten des eigentlich querverwiesenen Items *blindeTabellen* – eine Präzisiereung der Voraussetzung ist häufig angebracht!):

$$(\{\mathit{Grafikbeschriften}, \mathit{TextGrafik}\}, \mathit{BeschriftenUmfliessen}) \in I$$

(Im Klartext bedeutet das: Man kann statt der üblichen Beschriftung von

Grafiken mit dem Attribut `align = {top, middle, bottom}` auch mittels einer *blinden Tabelle* Text neben die Grafik setzen.)

Es ist darauf zu achten, daß die alten SFBen aus I entfernt werden, so daß keine Widersprüche auftreten.

Die Anwendung des dritten Teils des Übersetzungsverfahrens vollständigt die Implikationsrelation I um ihre zweite Hälfte (Anhang A.1).

In Abb. 2.9 sind die zusätzlichen Kanten (vgl. Abb. 2.6) dem vorläufigen UND/ODER-Graphen aus Abb. 2.7 hinzugefügt worden. Entsprechend der Regel für Typ 2 wurde die Kante $\neg$*Farbdefinition* $\Rightarrow \neg$*Regeln* entfernt.

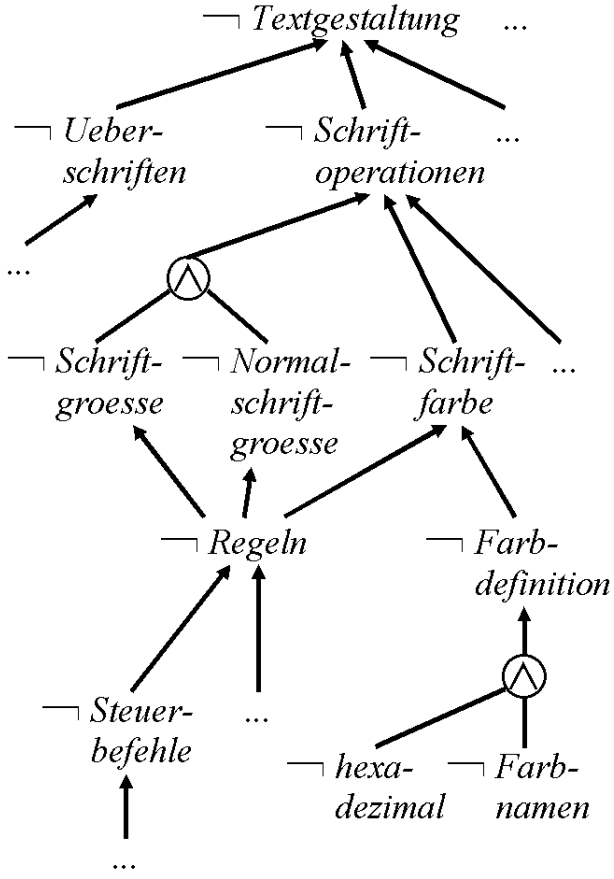


Abbildung 2.9: UND/ODER-Graph einschließlich Querverweisen

Die Wissensstruktur hat eine komplexe Gestalt angenommen. Aus dem ursprünglichen Baum der Höhe 3 ist ein Graph mit vielen Ebenen und Verzweigungen geworden.

Die gesamte Relation I, welche alle drei Teilergebnisse vereint, ist im

Anhang A.1 nochmal komplett dargestellt – sortiert nach den Konsequenzen, so daß zu jedem Item sämtliche Prämissen (im Graphen die eingehenden Kanten) auf einen Blick ersichtlich sind.

Die toolunterstützte Berechnung der Basis im nächsten Abschnitt erfolgt konsequenzenweise. Insofern ist diese Form der Auflistung eine Arbeitserleichterung.

2.3 Berechnung der Basis des Wissensraumes

Tool Es wird praktisch unmöglich, den Wissensraum bzw. dessen Basis per Hand zu berechnen. Deshalb verwende ich als Hilfsmittel das von der Abt. Mathematische Psychologie entwickelte „KQ_Win“.

An sich handelt es sich dabei um ein Programm zur Befragung von Experten, man kann aber auch Items und Standardformbehauptungen direkt eingeben (Fragestrategie “question by expert”).

Das Programm errechnet daraus die Basis in Matrixform und speichert sie in eine Datei, die vom Lehrsystem AdAsTra eingelesen werden kann.

Teilen der Itemmenge Die Verwendung von KQ_Win bringt eine Komplikation mit sich: Es kann maximal eine Itemmenge der Größe $|Q| = 100$ verarbeitet werden!

Q muß deshalb partitioniert und die Teilmengen Q_1 und Q_2 getrennt eingegeben werden. Die zwei Einzelergebnisse werden dann in eine gemeinsame Matrix-Datei geschrieben, die – auf einem anderem Weg – um die ausgelassenen Voraussetzungsbeziehungen zwischen beiden Mengen ergänzt wird.

Welches ist zu diesem Zweck die günstigste Aufteilung?

Die erste Idee ist, die Itemmenge irgendwo in der Mitte zu spalten. So ergeben sich zwei Matrizen B_1 und B_2, die – nach Umordnen der Zeilen gemäß der Itemnumerierung – auf der Quasi-Hauptdiagonalen (es ist ja keine wirklich quadratische Matrix) jeweils Einsen enthalten, ansonsten Nullen und Zweien. Fügt man die beiden Hälften dann gemäß Schema 2.1 zusammen, bleibt diese Eigenschaft erhalten, und die 144 Spalten sind nach Itemnummern sortiert.

$$\begin{pmatrix} B_1 & . \\ . & B_2 \end{pmatrix} \tag{2.1}$$

Prinzipiell müssen jetzt nur noch die Lücken mit Nullen bzw. mit Zweien für die noch nicht berücksichtigten Implikationen gefüllt werden – doch näheres dazu weiter unten.

Damit möglichst wenig nachzutragen ist, muß die Itemmenge an einem Punkt getrennt werden, wo möglichst wenige Kanten im Graphen verloren ge-

hen. Ohne Querverweise und Kanten, an denen „*“-Items beteiligt sind, enthält der UND/ODER-Graph sieben disjunkte Bäume (vgl. Abb. 2.7). Erhält man die entsprechenden Teil-Itemmengen, so wird ein Großteil der SFBen bei den zwei Programmdurchläufen berücksichtigt.

Unter dieser Forderung bleiben zwei Möglichkeiten der Aufspaltung von Q, so daß Teilmengen mit ≤ 100 Wissensinhalten entstehen: vor oder nach den Items aus dem Kapitel **Absatztypen und Textgestaltung**. In Zahlen: vor Item 45 oder nach Item 81.

Ich ordne das Kapitel der ersten Hälfte von Items zu, weil die nachträglichen Einträge in beiden Fällen überschaubar wären und somit die gleichmäßigere Aufteilung den Zuschlag erhalten kann. Einerseits spart man so für die größere der zwei Mengen (den “worst case”) Zeit- und Speicherbedarf beim Programmlauf, und andererseits hat sie für die Darstellung der Basis eine handhabbarere Form.

$$\begin{aligned} Q_1 &:= \{Regeln, \ldots, Linienhoehe\} \subset Q, \quad |Q_1| = 81 \\ Q_2 &:= \{Tabellen, \ldots, Maps\} \subset Q, \quad |Q_2| = 63 \end{aligned}$$

Eingeben der Items KQ_Win wird jeweils eine „Parameterdatei“ (Beschreibung in [Kal94], S. 74 – 80) übergeben, in welche die allgemeinen Formulierungen zur Befragung des Benutzers sowie die Items mit Nummern und Bezeichnungen eingetragen werden.

Die sprachliche Gestaltung der Fragen ist hier unbedeutend, denn die gültigen Implikationen stehen ja schon fest. Es wird einfach die Standardfrage gestellt: „Stellen Sie sich eine Person vor, die P nicht beherrscht. Kann man dann daraus schließen, daß die Person auch q nicht beherrscht? Ja oder nein?“

Als Nummern und Bezeichnungen für P bzw. q werden jene aus Anhang A.1, **Items** verwendet.

Eingeben der SFBen Für jedes Item aus der aufgelisteten Implikationsrelation (Anhang A.1, I **komplett**) werden einfach nacheinander alle SFBen in KQ_Win eingetragen: Zuerst wählt man die Konsequenz aus der Liste der Items, stellt danach die Prämisse zusammen, und schließlich bestätigt man die daraus generierte Frage.

Partitionsübergreifende SFBen, die ja in keiner der beiden Befragungen eingegeben werden können, weil entweder Prämisse oder Konsequenz fehlt, werden gesammelt, um später nachgetragen zu werden.

Während der Befragungsprozedur stehen einem Statistiken und Teilansichten des entstehenden UND/ODER-Graphen zur Verfügung. Angegeben wird u. a. die Größe der aktuellen Basis. Zu Beginn enthält sie für jedes Item

einen (minimalen) Wissenszustand, das entspricht einer Zeile in der Matrix. Ihr Umfang wächst mit jedem Eingeben einer mehrelementigen Prämisse (Lernalternativen).

Nach Beendigung der Befragungen ergibt sich:

$$|Q_1| = 81, \quad |I_1| = 110, \quad |\mathcal{B}_1| = 101$$
$$|Q_2| = 63, \quad |I_2| = 99, \quad |\mathcal{B}_2| = 65$$

Für die Umgebung einer aktuell ausgewählten Konsequenz kann ein Teilgraph – mit der Möglichkeit zur Weiterverfolgung entlang der Implikationskanten – angezeigt werden. Abb. 2.10 gibt einen Graphen aus der KQ_Win-Befragung wieder, welcher Abb. 2.9 entspricht.

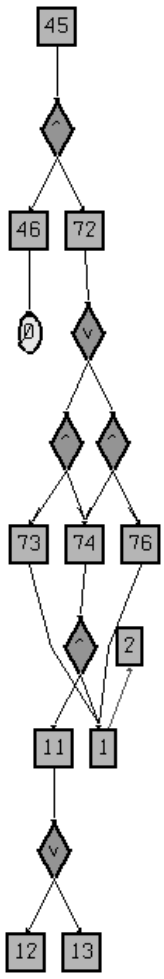


Abbildung 2.10: UND/ODER-Graph in KQ_Win

Die graphische Veranschaulichung von logischen Ausdrücken in KQ_Win weicht von der in der Studienarbeit ab, beide Darstellungsformen sind jedoch äquivalent.

Zum einem trifft der KQ_Win-Graph positive Aussagen:

$$Farbdefinition \Rightarrow hexadezimal \vee Farbnamen$$

anstatt

$$\neg hexadezimal \wedge \neg Farbnamen \Rightarrow \neg Farbdefinition$$

Zudem werden komplexe Logikausdrücke anders vereinfacht (vgl. [Fre99], S. 53 – 59). Alternativen – in KQ_Win die ODER-Knoten! – kommen, entsprechend der Zeilenschreibweise in der Ergebnismatrix, im Graphen zuerst bzw. stehen im Logikterm außen:

$$Schriftoperationen \Rightarrow (Schriftgroesse \wedge Schriftfarbe) \vee (Schriftfarbe \wedge Normalschriftgroesse)$$

anstatt

$$((\neg Schriftgroesse \wedge \neg Normalschriftgroesse) \vee \neg Schriftfarbe) \Rightarrow \neg Schriftoperationen$$

Abb. 2.11 hält eine Bildschirmansicht aus der KQ_Win-Befragung fest. Die SFB wurde, wie gewohnt, negativ formuliert.

Beide berechneten Basismatrizen B_1 und B_2 finden sich in Anhang A.2.

Integration der Ergebnisse B_1 und B_2 werden nach dem Schema 2.1 in die vorläufige Gesamtmatrix eingefügt. Die Lücken sind erstmal Nullmatrizen, mit der Bedeutung: „Q_1 und Q_2 stehen in keiner Beziehung zueinander."

$$\begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} \tag{2.2}$$

Ziel ist es nun, die Nullen an den Stellen durch Zweien zu ersetzen, wo Items aus der einen Partition Voraussetzung für solche aus der anderen sind. Damit entstehen gemischte Basiselemente.

Als erstes werden die zuvor ausgelassenen SFBen per Hand nachgetragen. Allgemein geschieht dies folgendermaßen:

> Sei $n(a)$ die Nummer eines Items a.
> Für jede Konsequenz q arbeite die noch ausstehenden SFBen der Reihe nach ab.

Für jede SFB $(\{p_1, \ldots, p_k\}, q)$:

Für jede Zeile von B, wo in der $n(q)$-ten Spalte eine 1 steht (für q minimale Wissenszustände):

Füge $k-1$ Kopien der Zeile ein (d. h. bei einelementiger Prämisse keine Kopien). Für jedes Prämissenelement p_i $(i = 1, \ldots, k)$:

Trage in die i-te der k Zeilen in der $n(p_i)$-ten Spalte eine 2 ein (p_i ist Voraussetzung für q; p_i ist Element des für q minimalen Wissenszustands).

Die während der KQ_Win-Befragung gesammelten Implikationen sind:

16. $(\{Mailadresse\}, Credits)$

28. $(\{Alternativtext\}, Groessenangabe)$
$(\{GrafikBreiteHoehe\}, Groessenangabe)$

38. $(\{WWW\}, Weiterleitung)$

84. $(\{Regeln\}, Tabellendefinition)$

91. $(\{Breite^*\}, TabellenHoeheBreite)$

92. $(\{Regeln\}, Hoehe^*)$

94. $(\{Ausrichten^*\}, Zellenausrichtung)$

99. $(\{Hintergrundfarbe^*\}, Farbtabellen)$
$(\{Hintergrundbild^*\}, Farbtabellen)$

100. $(\{Ausrichten^*\}, BeschriftungAusrichtung)$

101. $(\{Regeln\}, Tabellenueberschrift)$

102. $(\{Regeln\}, Textfortsetzung^*)$

106. $(\{Breite^*\}, Seitenraender)$
$(\{Bereiche\}, Seitenraender)$

107. $(\{Hintergrundfarbe^*\}, Farbflaechen)$

114. $(\{Regeln\}, Linkaufbau)$

136. $(\{Regeln\}, Referenzieren)$

138. $(\{Breite^*\}, GrafikBreiteHoehe)$

139. $(\{Ausrichten^*\}, BeschriftenUmfliessen)$
$(\{Absatzdefinition\}, BeschriftenUmfliessen)$

140. $(\{Regeln\}, Grafikbeschriften)$

Mehrelementige Prämissen treten hierin gar nicht auf, wohl aber mehrere für einen Wissensinhalt (Item 139) minimale Wissenszustandszeilen.

Noch ist die resultierende Matrix keine Wissensraumbasis. Sie muß in einem zweiten Schritt bezüglich verallgemeinerter Transitivität abgeschlossen werden – eine Aufgabe, die zuvor von KQ_Win übernommen wurde. Verallgemeinerte Reflexivität gilt bereits per Definition (der Matrix).

Ich führe den Abschluß mit dem Programm "surmise-closure" von Cord Hockemeyer, Institut für Psychologie an der Universität Graz, durch. Mathematisch liegt diesem [Hoc98], Theorem 3 zugrunde.

Der Basisabschluß-Algorithmus dürfte gerade in Verbindung mit dem Nachtragen von Items und Voraussetzungsbeziehungen in eine existierende Wissensstruktur von größerer praktischer Relevanz sein. Man denke an die thematische Erweiterung adaptiver Lernprogramme im nachhinein, für diesen Kurs etwa um den Umfang weiterer SELFHTML-Kapitel.

Auch kleine Fehler – ausgelassene Voraussetzungsbeziehungen – in einer ermittelten Wisensstruktur lassen sich mit Hilfe des Werkzeugs schnell ausbessern, ohne eine umfangreiche Expertenbefragung wiederholen zu müssen.

Die manuell eingetragenen Zweien produzieren also eine Menge neuer, transitiv gültiger Voraussetzungen. In erster Linie handelt es sich dabei um partitionsübergreifende Beziehungen; über mehrere Folgerungsschritte „hin und zurück" kommen aber auch wieder partitionsinterne Implikationen hinzu, z. B.

$$\underline{\neg Regeln} \Rightarrow \neg Mailadresse \Rightarrow \underline{\neg Credits} \quad (\underline{\neg 1} \Rightarrow \neg 127 \Rightarrow \underline{\neg 16}),$$

was dann zu Veränderungen in B_1 und B_2 (von nun an B_{11} und B_{22}) führt.

Durch UND-Verknüpfungen in den (per Hand oder Abschlußberechnung) ergänzten Implikationen entstehen darüber hinaus zusätzliche Wissenszustandszeilen, so daß die (nach Items, für welche die Zeilen minimal sind, sortierte) Ergebnismatrix

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{2.3}$$

(s. Anhang A.2), gegenüber der Matrix nach Schema 2.2 eine größere Dimension aufweist – vergleiche Tabelle 2.4 mit Tabelle 2.5.

B repräsentiert die Basis $\mathcal{B}$ zum Wissensgebiet HTML und stellt die mathematische Grundlage des adaptiven Kurses dar.

Matrix	# Zeilen	# Spalten
B_1	101	81
B_2	65	63
0 oben rechts	101	63
0 unten links	65	81
insgesamt	166	144

Tabelle 2.4: Dimensionen der Basismatrizen nach der Expertenbefragung

Matrix	# Zeilen	# Spalten
B_{11}	101	81
B_{22}	69	63
B_{12}	101	63
B_{21}	69	81
B	170	144

Tabelle 2.5: Dimension der Basismatrix nach der Integration

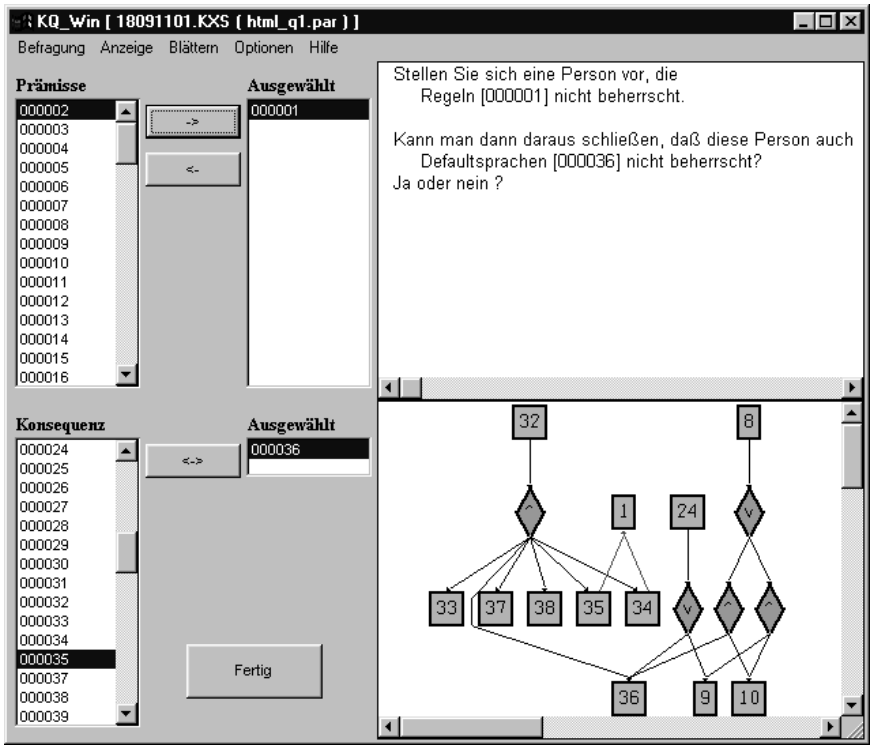


Abbildung 2.11: Screenshot der KQ_Win-Befragung

Kapitel 3

Aufgabengestaltung

3.1 Planung aus psychologischer Sicht

Anforderungen Sinn der Wissensdiagnose ist es, möglichst zutreffende Wissenszustände festzustellen. Sinn der Übung ist die gezielte Erweiterung dieses Wissens. Beides geschieht durch eine Präsentation von Aufgaben, welche wiederum Wissensinhalte repräsentieren. Um in Diagnose und Übung sichere Aussagen treffen zu können, gelten Anforderungen zum einen an den Aufgabeninhalt und im weiteren an die Aufgabenform.

Aufgabe(n) und Item müssen sich inhaltlich entsprechen. Eine Aufgabe muß genau das Wissen abdecken, welches der Wissensinhalt verkörpert. Sie darf nicht zu schwierig geraten, da sonst die Hürde zu nachfolgenden Übungsaufgaben zu hoch läge. Und sie darf keine Lücken aufweisen, sonst würden im Test Inhalte von Voraussetzungsitems ausgelassen werden.

Das Abtesten dieses Wissens muß so fair wie möglich vonstatten gehen. Leichtsinnsfehler sollen beim Bearbeiten der Aufgaben ebenso vermieden werden wie Erfolge durch Raten. Es gibt in der Beurteilung schließlich nur „richtig“ und „falsch“ bzw. „Item bekannt“ und „Item nicht bekannt“.

Generell ist es denkbar und sinnvoll, einem Item eine Menge von alternativen Aufgaben zuzuordnen, die jede den Inhalt auf eine andere Weise vermittelt. Da dies aber in den Algorithmen von AdAsTra (Version 2) noch nicht berücksichtigt ist und ohnehin den Umfang der Studienarbeit sprengen würde, gilt hier die Beschränkung auf ein 1:1-Verhältnis von Wissensinhalt und Aufgabe.

AdAsTra legt den Schwerpunkt auf das Üben von Wissensinhalten, nicht deren sprachliche Vermittlung. Trotzdem darf die instruktionale Komponente nicht vernachlässigt werden. Der Dialog mit dem Lerner über die Hilfestellungen und das Feedback zu seinen/ihren Antworten sollte so ausführlich

sein, daß der Grundbedarf an Erklärungen gedeckt ist. Eine Hilfestellungen sollte das Wissen darstellen, welches für den Lernschritt vom geforderten Vorwissen (Voraussetzungen aufgrund der Wissensstruktur) bis zur Lösung der Aufgabe vonnöten ist.

Sieht man von der Gelegenheit ab, jederzeit eine neue Diagnose durchzuführen, so berücksichtigt AdAsTra nicht das Vergessen von Wissen. Ebenso fehlt eine Nachschlagefunktion. Um trotzdem nicht bedingungsloses Auswendiglernen einzufordern, sollten in den Hilfetexten zu Items, die Wissen kombinieren (mittlere und obere Gliederungsebene von SELFHTML), die benötigten HTML-Schlüsselwörter erneut angegeben werden.

Didaktik Es wäre ein ungeheurer Aufwand nötig, um für jede der 144 Aufgaben, ein eigenes Konzept zu erarbeiten und dieses dann in die Sprache des Autorensystems umzusetzen.

Deshalb sollen zunächst einige wenige Grundtypen von Übungsaufgaben definiert werden, so daß im eigentlichen Entwurf für ein Item nur der passende Aufgabentypus gefunden und dieser mit Inhalt gefüllt zu werden braucht. In der Implementierungsphase entspricht das Schablonen-Dateien, von denen man vor dem Erstellen einer Aufgabe eine als Grundgerüst lädt.

Der Benutzer stößt dann im Test- und Übungsbetrieb immer wieder auf vertraute Bearbeitungsformen und kann sich so leichter an die Bedienung des Programmes gewöhnen.

Franz Schanda klassifiziert in [Sch95], S. 77 – 89 Aufgaben in Computer-Lernprogrammen nach Aufgabenstellung und Eingabemöglichkeiten. In den insgesamt sechs Typbeschreibungen finden sich Hinweise zur didaktisch vernünftigen Gestaltung der Interaktion.

- *Auswahlaufgaben*

  Fragen mit zwei möglichen Antworten (ja/nein) sind die primitivsten (1 aus 2). Multiple-Choice-Fragen, bei denen eine Antwort unter mehreren richtig ist (1 aus n), bieten schon mehr und sind programmiertechnisch noch leicht zu behandeln. Komplexere, praxisorientierte Aufgabenstellungen werden möglich, wenn unter den vorgegebenen Antworten mehrere richtige auszuwählen sind (k aus n). Das Schwierigkeitsniveau einer solchen Aufgabe hängt in erster Linie von der Qualität der *Distraktoren* ab (der falschen Antworten, die aus Ablenkungsgründen mit zur Auswahl stehen).

  Hier ist es trotz $\sum_{k=0}^{n} \binom{n}{k}$ möglichen Antwortkombinationen (für $n = 7$ Antwortvorgaben sind das 128) recht einfach, dem User *differenziertes Feedback* zu geben (detailliert auf seine Fehler und richtigen Antworten

einzugehen): Man schreibt zu jedem wählbaren Antwortpunkt einen kurzen erklärenden Text, wieso er zu den richtigen bzw. falschen gehört. Wenn der Lerner seine „Kreuze“ gesetzt hat, teilt man ihm/ihr mit, wie viele seiner/ihrer Auswahlen richtig waren, wie viele falsch und wie viele von den richtigen Antwortpunkten er/sie vergessen hat. Als Begründung listet man ihm/ihr die Beschreibungen zu den einzelnen Vorgaben, sortiert nach richtigen, falschen und ausgelassenen, auf.

Mögliche Frage: „Welche der folgenden Aussagen über Tags sind zutreffend?“

- *Zuordnungsaufgaben*

  Es geht darum, eine Vielzahl von Objekten nach vorgegebenen gemeinsamen Eigenschaften zu gruppieren. Die Auswahl kann z. B. visuell einsichtig per “drag and drop” geschehen.

  Mögliche Frage: „Welche der angegebenen Tags sind logische oder physische Textauszeichnungen bzw. überhaupt keine?“

- *Reihenfolgeaufgaben*

  Objekte oder Tätigkeiten müssen in eine Reihenfolge gebracht werden. Damit können z. B. Handlungsabläufe geübt werden. Auch hier empfiehlt sich die “drag and drop”-Oberfläche – oder eine schrittweise Fragestellung der Form „Was tut man als erstes?“ usw.

  Mögliche Frage: „Bringe die angegebenen HTML-Kommandos in die richtige Reihenfolge, so daß sie eine Tabelle mit drei Zeilen und zwei Spalten definieren.“

- *Einschätzungsaufgaben*

  Auf einer Skala soll ein Wert gewählt werden. Diese an sich primitive Aufgabe bietet gute Feedbackmöglichkeiten („viel zu wenig“, „etwas zu viel“, ...).

  Mögliche Frage: „Stelle auf den drei Skalen mit den Anteilen für Rot, Grün und Blau die Farbe ‚Türkis‘ in hexadezimaler Darstellung ein.“

- *Texteingaben*

  Zahleneingaben als Ergebnisse von Berechnungen oder das Ausfüllen von Lückentexten zur simplen Abfrage von Begriffen kann man noch gut auf Richtigkeit überprüfen. Differenziertes Feedback zu geben, ist aufgrund der Freiheiten bei der Eingabe jedoch fast unmöglich. In Lückentexten kann man immerhin angeben, welche Felder richtig und

welche falsch ausgefüllt wurden; bei längeren Texteingaben muß man den Lernenden eventuell, wie in einem Lehrbuch, eine Musterlösung zur Selbstkontrolle angeben *(unspezifisches Feedback)*.

Mögliche Aufgabenstellung: HTML-Fragment um fehlende Elemente vervollständigen

- *Übungen*

  Sie dienen zur praktischen Erarbeitung eines Themas und sind wesentlich umfangreicher als die zuvor beschriebenen *Aufgaben*, wo ja nur einzelne Lernziele kurz abgeprüft werden.

  Mögliche Aufgabenstellung: kleine HTML-Seite entwerfen und/oder implementieren

Von den genannten Aufgabentypen bieten sich für den HTML-Kurs zunächst die Texteingabe-Aufgaben an. Diese Eingabeform ist sehr nah an dem, was die Schüler in der Praxis tatsächlich tun werden: HTML-Quelltexte schreiben.

Über Lückentexte – gemeint ist das Vervollständigen von HTML-Fragmenten – lassen sich am besten gelernte Kommandos abfragen. Die Lücken dürfen, wie oben angedeutet, bloß nicht zu groß werden, sonst wird eine akurate Überprüfung schwierig. Manchmal werden mehrere korrekte Textantworten möglich sein. Dann sind alle erlaubten Schreibweisen zu überprüfen (z. B. Zentrieren mittels `<div align=center>` oder einfach `<center>`).

Ganze Programmierübungen werden sich nicht realisieren lassen; man müßte schon einen HTML-Interpreter an das Lehrsystem anbinden.

Nicht alle Items beinhalten das Reproduzieren und Kombinieren von HTML-Tags. Beim Vermitteln der grundlegenden Regeln etwa wird das Verständnis allgemeingültiger Konzepte abgefragt. Hierfür empfehlen sich dann Auswahlaufgaben, die durch längere Frageformulierungen und differenziertes Feedback einen inhaltlich tiefgehenderen „Dialog“ mit dem System ermöglichen.

Natürlich läuft man bei Multiple Choice gefahr, daß ein Schüler die Aufgabe durch Raten löst. Oder daß ein einziger Flüchtigkeitsfehler das Item als „nicht beherrscht“ kennzeichnet, obwohl der Wissensinhalt im großen und ganzen verstanden ist. Aufgrund der besonderen Stellung, welche die richtig/falsch-Entscheidung in adaptiven Lernprogrammen einnimmt, sind Auswahlaufgaben sparsam einzusetzen. Und wenn, dann immer in der Form „k aus n“.

Zuordnungsaufgaben sind ähnlich einsetzbar. Sie sind sogar ausdrucksstärker als Auswahlaufgaben, derart daß als Teilantworten nicht nur „ja“ und „nein“, sondern eine beliebige Anzahl Gruppen vorgegeben werden kann.

An dieser Stelle stößt man jedoch an die Grenzen der Entwicklungsumgebung von AdAsTra, das kein „drag and drop“ unterstützt. Zuordnungsaufgaben sind noch gut über *Listboxes* realisierbar; Reihenfolgeaufgaben kommen dagegen weniger in Frage. Das Zusammenstellen von Sprachelementen sollte, anders als das Trainieren von Handlungsabläufen, schon visuell und nicht durch schrittweises Fragen geschehen.

Bleiben die Einschätzungsaufgaben. Mit numerischen Werten hat man es in HTML nur in wenigen Fällen zu tun – beim Farbcode oder bei Größenangaben. Und dort ist das direkte Eingeben der Zahlenwerte gefordert. Schieberegler machen hier eher in Verknüpfung mit Texteingaben Sinn. Man könnte in einer Aufgabe zuerst den Wert auf einer Skala anfordern und dann das zugehörige Kommando direkt eingeben lassen: Abb. 3.1.

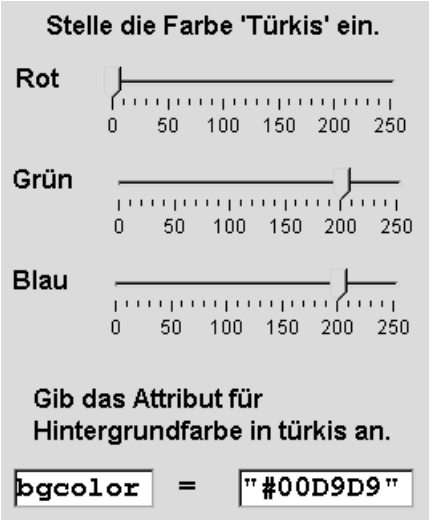


Abbildung 3.1: Einschätzungsaufgabe mit Texteingaben kombiniert

Überhaupt ermöglichen Kombinationen von Aufgabentypen viel komplexere Formen der Interaktion – erst recht unter Hinzunahme weiterer Gestaltungselemente von AdAsTra wie dem Einbinden von Sounddateien oder *Image Maps* (Bilder, in denen man Bereiche anklicken kann).

Insgesamt orientiert sich der Aufgaben(typen)entwurf an folgenden didaktischen Grundprinzipien:

- Auswahl der dem Problem angemessenen Frageform, Kombination von Frageformen; bei Auswahl- und Zuordnungsaufgaben Formulieren anspruchsvoller Distraktoren; Wahl einer einsichtigen grafischen Repräsentation für Zordnungsaufgaben

- Vorsehen einer Lösung, die – gerade bei freieren Frageformen wie Einschätzungsaufgaben und Texteingaben – alle erlaubten Antworten erkennen kann

- Einsetzen des differenzierten Feedbackkonzepts, wo immer sich das machen läßt – bei Texteingaben bestenfalls Einzelbewertung der Lücken ohne Begründung

- Abstimmen der Hilfetexte auf die Aufgabenlösung

Kognitionspsychologie Es gibt zwar einige Veröffentlichungen über die didaktischen Aspekte von Computer-Lernprogrammen sowie Qualitätschecklisten ([Sch95], S. 101 – 110 und Anhang), die einen qualitativ hochwertigen Entwurf gewährleisten sollen. Was jedoch schwieriger zu finden ist, ist eine psychologische Fundierung – kognitionspsychologische Untersuchungen, die aufzeigen, wie speziell mit multimedialer Unterstützung effektiv gelernt wird.

Welche Schlüsse sind etwa aus der Entdeckung der *Mehrspeicherstruktur* des Gedächtnisses im Hinblick auf das Behalten von Informationen zu ziehen?

Das *Kurzzeitgedächtnis (Arbeitsgedächtnis)* kann gleichzeitig etwa sieben kleinste Einheiten *(Chunks)* für 15 – 30 Sekunden aufnehmen ([Ede96], S. 254; [Her94], S. 62 – 64). Durch Gruppieren können Begriffe zu Chunks zusammengefaßt werden. Die Gedächtnisleistung ist besser, wenn die Chunks bereits bekannt und voneinander stark verschieden sind.

Während des Übens mit AdAsTra wird der Lerner sich den instruktionalen Hilfetext durchlesen, diesen wegklicken und dann die Aufgabe mit der Hilfestellung im Kurzzeitgedächtnis lösen. Es ist also darauf zu achten, allerhöchstens sieben einzelne, sauber getrennte und allgemeinverständliche Chunks in einem Hilfetext zu präsentieren.

Das eigentliche Ziel beim Lernen ist die Aufnahme von Wissen in das *Langzeitgedächtnis*. Es soll nicht wieder vergessen werden und in Anwendungssituation (HTML-Programmieren) abrufbar sein. Behalten und Anwenden des Gelernten werden erleichtert, wenn die Lernsituation der Abrufsituation ähnlich ist, diese simuliert ([Ede96], S. 255).

Der wichtigste Aufgabentyp wird somit sein, dem Lerner einen Entwurf eines Hypertextabschnitts zu zeigen, zu dem er/sie dann den entsprechenden HTML-Code einzugeben hat. Damit werden *Reproduktionsleistungen (Recall)* trainiert ([Ede96], S. 255), denn die HTML-Tags und ihre Anwendung müssen exakt wiedergegeben werden.

Eine einfachere Gedächtnisleistung ist das *Wiedererkennen (Recognition)*, welches vor allem in Auswahlaufgaben gefragt ist. Diese sollten nicht zum

Üben des konkreten Programmierens verwendet werden, sondern dort, wo eher abstrakte Ideen („Was ist guter HTML-Stil?“) abgefragt werden.

So wichtig das Auswendiglernen von HTML-Befehlen sein mag, es sollte *sinnvoll* geschehen, das Wissen nicht mechanisch „eingetrichtert“ werden. Sonst setzt schnell Vergessen ein (nach der Kurve von Ebbinghaus; [Ede96], S. 250).

Sinnvolles Lernen bedeutet beispielsweise, daß Tagabkürzungen wie `<b>` oder `<td>`, deren Bedeutung sich auch ein englischkundiger Lerner nicht sofort erschließen kann, im Hilfetext ausgeschrieben werden: “bold” bzw. “table data”.

Das sinnvolle Gegenmodell zum „Eintrichtern“ ist das „Verankern“ von Neugelerntem im Vorwissen *(Assimilation)*. Großer Vorteil beim Entwickeln eines adaptiven Kurses: Das Vorwissen eines Lerners ist bekannt!

Ein Mittel, Vergessen von Gelerntem entgegenzuwirken, wäre das Nennen – ohne neuerliches Erklären – der Prämissen des Wissensinhaltes im Hilfetext (mit Itemnamen, HTML-Schlüsselwörtern und Anknüpfungspunkten). Auch Analogien, wie etwa zwischen Tabellen und Grafiken, können dort Erwähnung finden.

Eine weitere interessante Theorie der Gedächtnispsychologie ist die der *dualen Kodierung* nach Paivio (s. [Ede96], S. 220f). Wissen kann im Gedächtnis *aussagenartig* (sprachlich) oder *analog* (bildhaft) repräsentiert sein.

Die Präsentation im multimedialen Lernprogramm sollte so gewählt sein, daß sie der natürlichen Art der Speicherung entspricht und somit leicht übernommen werden kann (vgl. [Ede96], S. 238). Es stellt sich also die Frage, an welchen Stellen Bilder verwendet werden sollten und wo Text sinvoller ist.

Text ist geeigneter, abstrakte Ideen (z. B. browserunabhängiges Programmieren) zu vermitteln, denn diese sind vorwiegend aussagenartig repräsentiert ([Ede96], S. 231). Bilder passen zu konkreten, anschaulichen Inhalten (z. B. der räumlichen Gestaltung einer Tabelle), welche der analogen Kodierung entsprechen.

Wenn unter Bildern auch Teilansichten von Hypertextseiten zu verstehen sind, die formatierten Text zeigen, dann sollte im HTML-Kurs ausgiebig Gebrauch von Abbildungen gemacht werden. Fast alles, was dort vermittelt werden soll, ist konkretes Wissen.

Schanda unterscheidet *Abbilder* von *logischen Bildern* ([Sch95], S. 94 und 97) – Diagrammen. Diese zeichnen sich dadurch aus, das sie eine eigene Sprache verwenden.

Für HTML kommen Sprachen der Informatik in Frage. *Syntaxdiagramme* ([Bro92], S. 78f) können Informatikern schnell vermitteln, wie komplexe Sprachkonstrukte, etwa das Beschreiben von Tabellen mit Zeilen, Spalten usw., funktionieren. Um aber allgemein verständlich zu sein, sollten diese nie

ohne zusätzliche natürlichsprachliche Erläuterung dastehen.

Generell sind Bilder (und Animationen) der Lernleistung zuträglicher als reine Texterklärungen ([Lew97], S. 225). Sie fördern Motivation, Verstehen und Aufmerksamkeit ([Ede96], S. 237).

In Maßen verwendet lösen Bilder Gefühle aus, wirken dekorativ und wecken die Neugiermotivation. Nach Paivio müssen Bilder „zur geistigen Verarbeitung herausfordern, um lernwirksam zu sein“ ([Sch95], S. 94f).

Am hilfreichsten sind Bilder mittlerer Komplexität ([Lew97], S. 109). Bilder sollen nicht unbedingt realistisch sein, sondern die wesentlichen Lerninhalte und ihre Zusammenhänge herausstellen, wobei unwichtige Details bewußt herausgelassen werden ([Sch95], S. 94).

Es braucht nicht immer eine vollständig gefüllte Hypertextseite gezeigt werden. Schematische Darstellungen mit eingetragenen Erläuterungen sind verständlicher. Um das Definieren von Seitenrändern mit Hilfe einer blinden Tabelle zu demonstrieren, genügt eine Seite voller Textzeilen, und hineingezeichnet die Abmessungen von Tabelle und Rand.

Andererseits muß dem angehenden HTML-Programmierer auch mal eine Website gezeigt werden, wie er/sie sie in der Realität zu gestalten hat (vgl. Röntgenbilder-Beispiel in [Sch95], S. 96).

Illustrationen können in ihrer Komplexität dem Vowissen angepaßt werden. HTML-Experten haben bereits gefestigte *mentale Modelle* (vgl. [Ede96], S. 252f) entwickelt. Sie werden durch neue schematische Darstellungsformen eher durcheinandergebracht, Anfänger dagegen durch zu viele Informationen überfordert ([Lew97], S. 111)

Daraus entsteht die Idee, für die Aufgaben „unter“ *Regeln* nur einfache Abbildungen bzw. für die Items, die viel Wissen zusammenfassen (aus der oberen Gliederungsebene von SELFHTML) nur realistische Darstellungen zu verwenden.

Damit Bilder die Aufmerksamkeit des Lernenden auf sich ziehen bzw. nach ihrem Studium von sich weglenken können, müssen sie vom Text leicht unterscheidbar sein, etwa in Hintergrundfarbe und Schriftstil.

Nach den Bildern könnte man als nächstes Animationen und Ton untersuchen. Beides werde ich aber nur in Aufgaben benutzen, die auch die Verwendung solcher Gestaltungselemente in HTML-Seiten zum Thema haben: Einbinden animierter Grafiken, Hintergrundmusik und das Unterstützen nichtvisueller Browser.

Diese Betrachtungen geben lediglich einen kleinen Ausschnitt kognitionspsychologischer Untersuchungen der multimedialen Lehre und computergestellter Übungsaufgaben wieder. Neben Gedächtnis- und Wahrnehmungspsychologie können weiter Forschungsergebnisse zu Aufmerksamkeit, Sprache usw. einfließen, und sie werden es vermutlich in Zukunft vermehrt tun.

Die kognitionspsychologischen Überlegungen fließen in die Gestaltung von Aufgabenstellung und Erklärung ein:

- Allgemeine Verständnisfragen werden vorwiegend sprachlich als Auswahlaufgaben gestellt, konkrete Sprachelemente als Texteingabeaufgaben mit Abbildung eines zu implementierenden Hypertextentwurfs.
  Die Illustrationen sind der Schwierigkeit der Aufgabe angemessen, verdeutlichen die wesentlichen Iteminhalte und setzen den Problemlöseprozeß in Gang.

- Im Hilfetext werden Prämissenelemente und Analogien genannt sowie das im Item neu enthaltene, zur Aufgabenlösung benötigte Wissen in höchstens sieben übersichtlich geordneten Einheiten erläutert.
  Die Erklärung ist für den Lerner inhaltlich nachvollziehbar und bei Bedarf um ein kleines Beispiel, bei komplexen HTML-Konstrukten durch ein Syntaxdiagramm ergänzt.

Software-Ergonomie Losgelöst von Lernaspekten wird Software schon länger psychologisch auf ergonomische Merkmale hin untersucht. Hier ist Kognitionspsychologie direkt in Lehrbücher wie Michael Herczegs [Her94], in Checklisten und sogar international vereinbarte Normen eingeflossen. Die Software-Ergonomie umfaßt u. a. das Bildschirmdesign und die Gestaltung von Mensch-Computer-Dialogen.

Ersteres wird sowohl bei Herczeg ([Her94], S. 71 – 84) als auch bei Schanda behandelt ([Sch95], S. 90ff).

Die Hauptschriftart im Lernprogramm sollte serifenlos sein, damit der Text auf dem im Vergleich zu bedrucktem Papier unscharf wirkenden Monitor deutlich lesbar ist. Es sollten nicht übermäßig viele Schriftarten und -stile vorkommen. Vorzuziehen ist dunkle Schrift auf hellem Grund, u. a. weil man daran von gedrucktem Text her gewöhnt ist. Ob der nötige Helligkeitskontrast erreicht wird, kann mit einem monochromen Monitor getestet werden.

Farben übermitteln bereits Bedeutungen, z. B. steht Rot für „Alarm“ und Grün signalisiert „in Ordnung“. Rot und Blau sollten nach Möglichkeit nicht gleichzeitig verwendet werden, weil das Auge nicht auf beide fokussiert werden kann. Ohnehin dürfen die Farben nicht zu grell sein.

Die gewählten Bildschirmfarben und Schriften[1] für den HTML-Übungskurs finden sich in Tabelle 3.1. Jedem der sieben Kapitel aus SELFHTML (Wurzeln der sieben Bäume) wird eine Farbe zugeordnet, so daß der Lerner nicht völlig die Orientierung verliert. Immerhin bestimmt die Reihenfolge der

[1] „Schriftart“ meint die gesamte Schriftartenfamilie. Arial (Windows) steht stellvetretend für Helvetica (Unix) u. a.

Lernabschnitte ein interner Algorithmus, der vermutlich häufig von Kapitel zu Kapitel springt.

Bildschirmelement	Hintergrund	Schriftfarbe	Schriftart
Kapitelüberschrift	Kapitelfarbe hell	Kapitelfarbe dunkel	**Arial (Bild)**
Itemüberschrift	Kapitelfarbe hell	schwarz	Arial groß
Frage	Kapitelfarbe hell	schwarz	Arial
Hilfe	Kapitelfarbe hell	schwarz	Arial
Abbildungen	beige	schwarz	**Arial fett**
Hypertext	weiß	schwarz	Roman
HTML-Code	hellgrau	schwarz	`Courier`
Feedback „richtig“	dunkelgrün	weiß	Arial
Feedback „falsch“	dunkelrot	weiß	Arial

Tabelle 3.1: Bildschirmaussehen

Kapitel	Farbe	Hintergrund	Schrift
Grundregeln	Blau	(150, 200, 230)	(0, 30, 130)
Dateiangaben	Lila	(210, 200, 230)	(60, 0, 100)
Globaldefinitionen	Türkis	(120, 200, 200)	(0, 90, 90)
Textgestaltung	Gelb	(240, 240, 190)	(140, 120, 0)
Tabellen	Braun	(220, 160, 110)	(80, 40, 0)
Links	Rot	(220, 140, 140)	(140, 30, 30)
Grafik	Grün	(140, 200, 140)	(0, 100, 40)
Allgemeines	Grau	(210, 210, 210)	(40, 40, 40)

Tabelle 3.2: Kapitelfarben

Neben dem generellen Aussehen will auch der Aufbau der Bildschirme wohlüberlegt sein. Wie gruppiert man zusammengehörige Informationen, so daß sie auch als Einheit wahrgenommen werden? Dafür können die *Gestaltgesetze* der Wahrnehmungspsychologie als Grundlage dienen (s. Abb. 3.2; vgl. [Her94], S. 58 – 60).

Wo das Autorensystem eine ungünstige räumliche Gruppierung sinnverwandter Elemente vorgibt, etwa einen zu großen Abstand zwischen prinzipiell nebeneinanderstehenden Infomationen, werde ich diese mit einer gemeinsamen, von der Umgebung geringfügig abweichenden Hintergrundfarbe unterlegen („gemeinsame Region“).

Beschreibende Texte zu übereinander- bzw. nebeneinanderstehenden Grafikelementen, werde ich in derselben Reihenfolge übereinander bzw. nebeneinander anordnen. Um die interaktiven Elemente der Aufgaben herauszuheben, werden alle Aufforderungen zu einer Aktion, z. B. Texteingeben, **fett** gedruckt.

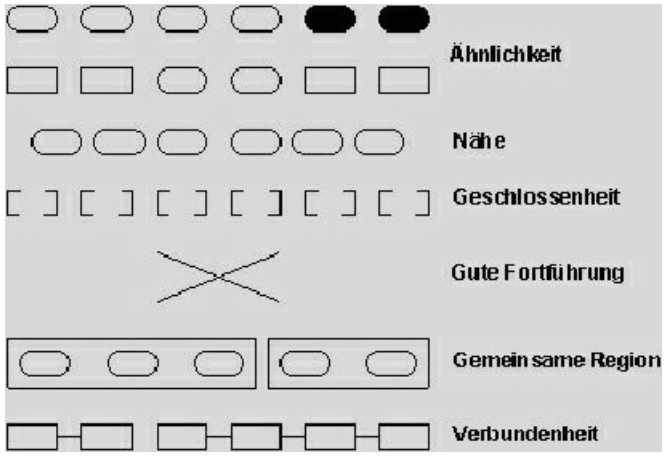


Abbildung 3.2: Einige Gestaltgesetze

Weitere Regeln für den Bildschirmaufbau stehen bei Herczeg auf S. 77ff. Dort findet sich auch eine Klassifikation von Mensch-Computer-Dialogen. AdAsTra bietet als Interaktionselemente Checkboxes und Listboxes für *Menüs* ([Her94], S. 93f) sowie Eingabefelder für *Formulare* ([Her94], S. 101) an; dazu Schieberegler, deren Verwendung sich nicht so recht einordnen läßt.

Auch für Menüs und Formulare finden die oben erläuterten Gestaltungsregeln Verwendung: Begrenzung auf 7 Menüpunkte, Gruppieren von Einheiten, Anordnen gemäß der Gestaltgesetze.

Bei Menüs ist auf eine durchdachte Darbietung der Auswahlmöglichkeiten zu achten (Listbox, Checkbox oder Kombination). Sie finden in Auswahl- und Zuordnungsaufgaben Verwendung.

Formulare zeichnen sich durch eine sehr freie Form der Eingabe aus. Deshalb sollte zumindest der erwartete „Datentyp“ erkennbar sein – z. B. an der Feldbreite (wie die gesuchte Eingabe), Schriftart (`Courier New` für HTML-Kommandos), Hintergrundfarbe (für Zahleneingaben durchgängig gelb) oder zusätzlichen Angaben (Rahmentext um die Lücken). Formulare kommen in Texteingabeaufgaben vor.

Ergonomische Erkenntnisse geben also Hinweise für die Oberflächengestaltung der Aufgabentypen:

- Abfragen der Antwort bei Auswahlaufgaben über Checkbox-Menüs, bei Zuordnungsaufgaben über Listbox-Menüs, Formulare für Texteingaben und Schieberegler für Einschätzungsaufgaben; Bildschirmdesign anhand von Tabelle 3.1; logische Gruppierung von Informationen durch Ausnutzen der Gestaltgesetze (ausgenommen Distraktoren)

- Kennzeichnen der Aussage von (Teil-) Feedback durch die Hintergrundfarben Grün (richtig) und Rot (falsch)

- Entwerfen der Hilfebildschirme gemäß Tabelle und Gestaltgesetzen

Motivationspsychologie Selbst der didaktisch bestaufbereitetste Lernstoff bringt wenig Nutzen, wenn den Schülern das Interesse fehlt, ihn sich anzueignen. Wie motiviert man Lernende, sich dem Thema HTML in computerpräsentierter Form zu nähern? Wie fördert man bereits vorhandene Motivation?

Ich gehe hier kurz auf vier unterschiedliche Formen von Motivation[2] ein und versuche jeweils zu klären, wie mit AdAsTra entworfene Übungsaufgaben solcherart motivierte Personen zur Beschäftigung mit ihnen anregen können.

- *Neugiermotivation.*

  Ein neugiermotivierter Lerner zeigt einen Drang zur Erforschung von Programm und Lerngebiet (natürliches *Explorationsbedürfnis*), um dabei immer wieder Neues zu entdecken. Dabei muß eine „optimale Inkongruenz“ ([Ede96], S. 362) in der Mitte zwischen neuen und bekannten Dingen gefunden werden, weil zuviel Neues wiederum abschreckt.

  Adaptive CBT-Kurse, jedenfalls solche unter dem derzeitigen AdAsTra, laufen stark programmgesteuert ab und bieten wenig Gelegenheit zum selbstinitiierten Explorieren.

  Andererseits findet ein ständiger Austausch der Aufgaben statt. Diese sollten abwechslungsreich gestaltet sein und öfter etwas neues bieten. Eine Beschränkung auf Auswahl- und Texteingabeaufgaben könnte für Langeweile sorgen, deshalb werden erweiterte Einschätzungsaufgaben – gerade Schieberegler laden zum Herumprobieren ein – sowie an einigen Stellen Zuordnungsaufgaben mit Listboxes Verwendung finden.

  Bei vier grundsätzlichen Interaktionsformen kann es bleiben. Ihre wiederkehrende Struktur wird mit der Zeit zu dem bekannten Gerüst, in das Neues eingebettet werden kann, so daß es neugierig aufgenommen wird. Dieses wirklich Neue muß dann der Aufgabeninhalt hergeben.

  Dabei könnte es sich um einen neuen HTML-Abschnitt wie z. B. **Verweise – Hyperlinks** handeln. Es liegt an den Aufgaben der grundlegenden Items eines weiterführenden Kapitels, Faszination an dem neuen Konzept zu vermitteln. Das Neugierbedürfnis von Lernern, die eine

[2] nach der Vorlesung „Motivationspsychologie“ von Prof. J. Hinz, Seminar für Psychologie, Sommersemester 2000

Programmiersprache grundsätzlich als eher trockenes Thema empfinden, mag dagegen durch Einkleiden der Aufgaben in immer wieder neue interessante oder lustige Anwendungsbeispiele gestillt werden.

Zu guter Letzt können und sollen, wie oben erläutert, Illustrationen das intellektuelle Interesse des Lernenden wecken.

- *Leistungsmotivation.*

  Gerade in unserer Gesellschaft werden durch Erfolg und Mißerfolg starke emotionale Reaktionen hervorgerufen. Das bedeutet jedoch nicht, daß das *Leistungsmotiv* automatisch bei allen stark ausgeprägt wäre. Man kann zwischen *hoch* und *niedrig leistungsmotivierten* Menschen unterscheiden (nach [Ede96], S 375ff).

  Hoch leistungsmotivierte Lerner streben eine möglichst gute *Bewertung* ihrer Aufgabenlösungen an.

  In einem adaptiven Kurs, in dem Lösungen entweder richtig oder falsch ohne genauere Abstufungen sind, bleibt als Möglichkeit zum „Loben“ guter Leistungen nur die Angabe, daß eine gelöste Aufgabe (objektiv, nicht gemessen am Vorwissen) besonders schwierig war.

  Konkret werde ich im Feedback zu einer richtig beantworteten Frage deren Schwierigkeit, visualisiert in Form von 0 bis 5 Sternen, angeben. Als Maß verwende ich die Anzahl der Voraussetzungen eines Items: Falls es für ein Item q genau einen minimalen Wissenszustand s in der Basis $\mathcal{B}$ gibt, dann $|s|$; bei n minimalen Wissenszuständen s_i $(i = 1, \ldots, n)$ den Durchschnitt $\lfloor \frac{\sum_{i=1}^{n} |s_i|}{n} \rfloor$. Die Umrechnung zeigt Tabelle 3.3.

# Voraussetzungen	Schwierigkeit
1	0
2 bis 8	*
9, 10	**
11, 12	***
13 bis 17	****
18 bis 44	*****

Tabelle 3.3: Itemschwierigkeiten

Um komplexere *Anregung durch Wettbewerb*, etwa „Lernspiele“, „Mitführen einer Punktewertung“ oder das „Versinnbildlichen des Lernfortschritts und der Lücken“ ([Sch95], S. 99f), zu realisieren, verfügt AdAs-

Tra (noch) nicht über die benötigten Darstellungs- und Interaktionsformen.

Für hoch leistungsmotivierte Menschen dürfte sich der Umgang mit adaptiven Lernprogrammen grundsätzlich angenehm gestalten, da sie von Natur aus (subjektiv) mittelschwere Herausforderungen aufsuchen, und genau die präsentiert ihnen ein adaptiver Kurs mit Berücksichtigung des Vorwissens.

Niedrig leistungsmotivierte Lerner streben danach, ein Scheitern und den damit verbundenen Erhalt negativen Feedbacks zu verhindern.

Für diesen Lernertyp genügt prinzipiell die Unterscheidung nach richtiger und falscher Lösung in Form von Feedback in roter oder grüner Signalfarbe. Ziel ist es dann, auf eine Vermeidung des roten Feedbacks hinzuarbeiten.

Unter Verwendung des Schandaschen differenzierten Feedbackkonzeptes (für Auswahlaufgaben) können bei insgesamt falscher Beantwortung neben den falschen auch die richtigen Teile wiederholt werden, so daß das Mißerfolgserlebnis nicht ganz so dramatisch ausfällt. Das Feedback zu jeder Teilfrage muß dann allerdings in beiden Farben vorbereitet werden (vgl. [Sch95], S. 89).

Niedrig Leistungsmotivierte sind mitunter nicht in der Lage, Aufgaben von (subjektiv) angemessener Schwierigkeit auswählen – was zwar die Software für sie erledigt, sie aber vielleicht nicht erkennen.

Es besteht die Gefahr, daß sie Mißerfolge überhöht schwierigen Aufgaben zuschreiben (*external-stabile Kausalattribution*; s. [Ede96], S. 378f) oder gar generellem eigenen Unvermögen (*internal-stabile* Attribution). Beides könnte zu einem Abbruch des Lernens mit dem Programm führen. Motivationsfördernd wäre dagegen, auf ein stärkeres Anstrengen des Lerners hinzuwirken (*internal-variabel*).

Die Feedbacktexte sind demnach so zu formulieren, daß einem Abschieben der Verantwortung entgegengewirkt wird und der Benutzer geneigt ist, dieselbe Frage erneut mit mehr Einsatz zu bearbeiten.

- *Flow.*

  Flow ist intensives Erleben *intrinsischer* (von innen kommender) Motivation. Intrinsisch hoch Motivierte finden ihre Belohnung im Lernprozeß selbst. Sie haben Spaß am Lösen der Aufgaben – sei es durch die Interaktion, Freude am Programmieren, dem Miterleben des eigenen Fortschritts oder, wie oben erläutert, dem Stillen von Neugier. Im

günstigsten Fall kann sich ein intrinsisch motivierter Lerner so intensiv mit dem Programm beschäftigen, daß er/sie alles um sich herum vergißt – ein „Flow-Erlebnis“.

Wiederum entsteht die wirkungsvollste Spannung bei (subjektiv) mittelschweren, d. h. dem Können einer Person angemessenen Aufgaben. Freude am Tun kann intensiv erlebt werden, wenn *Handlungsanforderungen* und *Fähigkeiten* als etwa gleich stark wahrgenommen werden. Zu hohe Anforderungen führen zu *Sorge*, zu niedrige lassen *Langeweile* aufkommen ([Csi99], S. 75f und Abb. 1).

Adaptive Lernprogramme bieten vom Grundprinzip her beste Voraussetzungen für das Erleben intrinsischer Motivation, denn die Schwierigkeit des Lernstoffes ist auf die Fähigkeiten des Lerners abgestimmt.

Im Kontext des Lernens gewinnt das Flow-Modell eine dynamische Komponente hinzu. Die Fähigkeiten eines Lerners verbessern sich ja kontinuierlich. Hebt man parallel dazu im gleichen Maße die Anforderungen an, dann dürfte ein Flow-Zustand nicht nur andauern, sondern kontinuierlich in seiner Komplexität wachsen (vgl. [Csi99], S. 79 und Abb. 2).

Um dies zu ermöglichen, müßte das Tutorprogramm in kleinen Abständen die Anforderungen erhöhen. Der Übungsalgorithmus müßte schrittweise einen „Pfad“ durch den Wissensraum bis zum größten Wissenszustand hin berechnen.

AdAsTra führt gegenwärtig nur den ersten Schritt über eine Voraussetzungsebene aus; danach muß ein neuer Test absolviert werden, um weiterzukommen.

Proportional zu den Anforderungen (Fragen) sollen die Lerner ihre Fähigkeiten steigern können. Das Verhältnis von einem Hilfetext zur zugehörigen Frage muß also möglichst ausgeglichen sein. Simple Aufgaben mit umfangreichen Erklärungen würden zu Langeweile führen bzw. dürftige Erklärungen für komplexe Aufgaben zu Sorge. Wird die richtige Balance getroffen, werden die Lernenden schnell vorankommen und dabei ihren eigenen Lernfortschritt spüren.

Freude am Tun entsteht nicht aus bloßem Konsumieren von Text. Schüler sollen Gelerntes „kontinuierlich ausprobieren und anwenden“ können ([Sch95], S. 98). Selbständiges, freies Handeln soll gefördert werden.

Dies ist bei AdAsTra gewährleistet, denn der Schwerpunkt liegt schon automatisch auf dem Lösen von Übungsaufgaben. Die anfangs vor-

gestellten Aufgabenklassen sowie die grundlegenden AdAsTra-Gestaltungselemente bieten allerdings für sich genommen noch keinen allzu großen Handlungsspielraum. Evenentuell können die Aufgabenstellungen durch geschickte Kombination verschiedener Interaktionsformen variabler gestaltet werden.

Warum z. B. sollte man in Abb. 3.1 unbedingt nach der Farbe Türkis fragen? Genauso kann der Benutzer sich selbst eine Farbe aussuchen – über die Schieberegler oder als eine Variante durch Anklicken auf einer Farbkreis-Image-Map – die dann bei korrekter Definition im Feedback auch dargestellt wird. Das führt zu einem kombinierten Aufgabentyp mit Wahlmöglichkeiten.

Generell ist darauf zu achten, daß die Aufgabenstellungen ein Konzentrieren auf den wesentlichen Item-Inhalt fördern, damit Freude im Lernen auch auf den Lerngegenstand bezogen wird.

- *Selbstverwirklichung.*

  Das Bild vom sich selbstverwirklichenden Menschen ist ein humanistisches ([Ede96], S. 372ff). Selbstverwirklicher sind schöpferisch tätig und bestrebt, ihr persönliches Potential zu entdecken und freizusetzen. Im Gegensatz zu Motivationsformen, bei denen ein defizitäres Bedürfnis befriedigt wird, ist hierbei die Rede von *Wachstumsmotivation* (vgl. Maslows Bedürfnispyramide in [Ede96], S. 373).

  Von besonderer Bedeutung für die Theorie der Selbstverwirklichung sind die *Inhalte* der Motive eines Individuums. Bei den bisherigen drei Motivationsformen wurde nur festgestellt, daß Lernende ein Neugierbedürfnis, ein Leistungsbedürfnis oder ein Freudebedürfnis haben – Neugier worauf? Leistung worin? Freude wodurch?

  Selbstverwirklicher lernen HTML, um mit den neu erworbenen Kenntnissen persönlich zu wachsen, insbesondere aber, um mittels dieser ihre eigenen Ideen kreativ in die Tat umzusetzen. Nicht nur um abwechslungsreich zu wirken, werden im Kurs Fragen mit praxisbezogenem Inhalt gestellt, die in den komplexeren Aufgaben als kleine Geschichte formuliert sein können.

  Auch Schanda empfiehlt, unter dem Schlagwort *Anregung durch Praxisnähe*, eine „praxisbezogene Einkleidung von Aufgaben und praxisnahe Beispiele“ ([Sch95], S. 99). Ein Gesamtrahmen mit einem einzelnen Anwendungsgebiet (s. [Sch95], S. 100), etwa das durchgängige Gestalten des Webauftritts einer Autovermietung, wäre meines Erachtens jedoch zu speziell und könnte an dem Thema nicht interessierte Lerner

eher abschrecken.

Die Aufgaben müssen vielmehr kreativitätsfördernd gestaltet sein, so daß Lernende ihre Ideen einfließen lassen können, idealerweise auch ihre bevorzugten Anwendungsgebiete. Das ist in einem vorrangig programmgesteuerten Kurs schwierig zu realisieren. Die Benutzer treffen keine Entscheidungen über ihren Lernweg, die verwendeten Materialien, die Aufgabenform – erst recht nicht über den Inhalt der Aufgaben. Für Schanda wäre damit das wichtigste Mittel zur Aktivierung verfehlt ([Sch95], S. 68f und 98).

Trotzdem versuche ich, in einigen Aufgaben auf höherer Schwierigkeitsebene Ansätze kreativen Gestaltens zu ermöglichen. Per Auswahl über Menüs (Listboxes) wird ein Thema, ein Design o. ä. ausgesucht werden können, worauf das Itemwissen dann per Eingeben des HTML-Codes angewendet wird. Bei korrekter Antwort wird der aus den gewählten Elementen zusammengesetzte Seitenausschnitt im Feedback angezeigt.

Die Anwendung motivationspsychologischer Theorien auf computergestützte Lernprogramme erbringt eine letzte Liste von Ideen für das Definieren der Aufgabentypen.

- Verwenden vier grundlegender Fragetypen: Auswählen, Zuordnen, Einschätzen (Map oder Schieberegler) mit Texteingabe sowie Implementieren eines abgebildeten Entwurfs – stellenweise mit erweitertem Handlungsspielraum oder kreativer Auswahl von Inhalt und Hypertext-Gestaltungselementen; Anwenden des HTML-Wissens auf Praxisbeispiele, ohne vom eigentlichen Thema abzulenken

- Programmieren dynamischer Lösungs-Scripts für Aufgaben mit erhöhtem Handlungsspielraum; Abdecken aller Kombinationsmöglichkeiten bei kreativen Aufgaben

- im Feedback Anzeigen der Schwierigkeit bei richtiger Antwort; Aufmuntern zu Wiederholung und mehr Anstrengung bei falscher Antwort; grünes und rotes Einfärben auch der Teile differenzierten Feedbacks; Darstellen der verschiedenen Ergebnisse bei kreativen Aufgaben und solchen mit erweitertem Handlungsspielraum

- Herausheben neuer Konzepte in den Hilfetexten

- Wahren des Gleichgewichts zwischen Frage und Hilfestellung

3.2 Entwurf der Aufgabentypen

Typenkonstruktion Ergebnis der didaktischen und psychologischen Planung sind vier grundlegende Fragetypen, wovon es bei den Einschätzungsaufgaben eine Variante mit Schieberegler sowie eine mit Image Map gibt.

Welche Typen können mit einem erweiterten Handlungsspielraum ausgestattet werden? Am ehesten kommen die Einschätzungsaufgaben in Frage. Die den übrigen Fragetypen zugeordneten Interaktionsformen sind hierfür zu einfach (Menüs) oder zu komplex (Texteingabe). Sämtliche Einschätzungsaufgaben erhalten daher diese Wahlmöglichkeit und heißen von nun an *Umsetzungsaufgaben*.

Welche Typen sollen um die kreative Auswahl erweitert werden? Einzig die *Implementierungsaufgaben* sind vom Umfang und der Interaktionsform her nahe genug am realen Programmieren, so daß es Sinn macht, sie mit praxisnahem Inhalt zu füllen. Der mit der kreativen Zusammenstellung verbundene größere Bearbeitungsaufwand und die Nähe zum Webdesign kommen dann aber erst für fortgeschrittenere Lerner in Frage – sprich: ab den schwierigeren Items. So ergeben sich auch für diesen Typ zwei Varianten.

Typendefinition Die sechs resultierenden Aufgabentypen mit ihrer Frage, ihrer Lösung (Algorithmus in Pseudocode), ihrem Feedback und ihrer Hilfe finden sich im folgenden tabellarisch aufgelistet.

1. Auswahlaufgabe: Tabelle 3.4
2. Zuordnungsaufgabe: Tabelle 3.5
3. Umsetzungsaufgabe Schieberegler: Tabelle 3.6
4. Umsetzungsaufgabe Image Map: Tabelle 3.7
5. Implementierungsaufgabe vorgegeben: Tabelle 3.8
6. Implementierungsaufgabe kreativ: Tabelle 3.9

Entwurf des Kurses Die Aufgabengestaltung orientiert sich an dem Rahmen, den die sechs Aufgabentypen vorgeben, sowie an den Erkenntnissen aus der Planungsphase.

Anhang B.1 listet alle 144 Aufgaben mit jeweils der Angabe des Typs, dem Frage- und dem Hilfeinhalt auf.

Das Aussehen von Lösung und Feedback wird durch die konkrete Fragestellung und den Aufgabentyp (mit gelegentlichen Anpassungen) gemäß der Tabellen bestimmt.

Frage	„Welche der folgenden Aussagen sind zutreffend?“ ◦ wahre Aussage `[w1]` ◦ falsche Aussage `[f1]` ◦ falsche Aussage `[f2]` ◦ wahre Aussage `[w2]`
Lösung	**if** `w1` **and** `w2` **and not** `f1` **and not** `f2` **then** `feed_richtig` **else** `feed_falsch`; **if** `w1` **then** `feed_w1` (`gruen`) **else** `feed_w1` (`rot`) **fi**; **if not** `f1` **then** `feed_f1` (`gruen`) **else** `feed_f1` (`rot`) **fi**; **if not** `f2` **then** `feed_f2` (`gruen`) **else** `feed_f2` (`rot`) **fi**; **if** `w2` **then** `feed_w2` (`gruen`) **else** `feed_w2` (`rot`) **fi** **fi**
Feedback	`feed_richtig`: „*** Prima, alle Antworten sind richtig!“ −→ Item bekannt! `feed_falsch`: „Einige Antworten waren falsch. Du kannst später eine neue Übung starten und dich wieder an dieser Aufgabe versuchen. Denke dann nochmal genauer über die Aussagen nach.“ −→ Item nicht bekannt! `feed_w1` (`farbe`): Begründung, warum die erste wahre Aussage zutrifft `feed_w2` (`farbe`): Begründung, warum die zweite wahre Aussage zutrifft `feed_f1` (`farbe`): Begründung, warum die erste falsche Aussage nicht zutrifft `feed_f2` (`farbe`): Begründung, warum die zweite falsche Aussage nicht zutrifft
Hilfe	„Zur Lösung der Aufgabe helfen Kenntnisse über. . . “ „Sie funktioniert ähnlich wie. . . “ „Folgende Inhalte kommen erstmals vor:“ (evtl. Beispiel, Bild) • ggf. Erläuterung eines neuen Konzeptes • Erklärung zur zweiten falschen Aussage • Erklärung zur zweiten wahren Aussage • Erklärung zur ersten wahren Aussage • Erklärung zur ersten falschen Aussage

Tabelle 3.4: Aufgabentyp 1 – Auswahlaufgabe

Frage	„Gib für jedes Element die Gruppe an, zu der es gehört.“ • 1. Element [`element1`] Gruppe 1 / Gruppe 2 / Gruppe 3 • 2. Element [`element2`] Gruppe 1 / Gruppe 2 / Gruppe 3 • 3. Element [`element3`] Gruppe 1 / Gruppe 2 / Gruppe 3
Lösung	**if** `element1` = “`Gruppe 2`” **and** `element2` = “`Gruppe 3`” **and** `element3` = “`Gruppe 1`” **then** `feed_richtig` **else** `feed_falsch`**;** **if** `element1` = “`Gruppe 2`” **then** `feed_element1` (`gruen`) **else** `feed_element1` (`rot`) **fi;** **if** `element2` = “`Gruppe 3`” **then** `feed_element2` (`gruen`) **else** `feed_element2` (`rot`) **fi;** **if** `element3` = “`Gruppe 1`” **then** `feed_element3` (`gruen`) **else** `feed_element3` (`rot`) **fi fi**
Feedback	`feed_richtig`: „*** Prima, alle Antworten sind richtig!“ $\longrightarrow$ Item bekannt! `feed_falsch`: „Einige Antworten waren falsch. Du kannst später eine neue Übung starten und dich wieder an dieser Aufgabe versuchen. Denke dann nochmal genauer über die Elemente nach.“ $\longrightarrow$ Item nicht bekannt! `feed_element1` (`farbe`): Begründung, warum das 1. Element zu Gruppe 2 gehört, und nicht zu Gruppe 1 oder Gruppe 3 `feed_element2` (`farbe`): Begründung für das 2. Element `feed_element3` (`farbe`): Begründung für das 3. Element
Hilfe	„Zur Lösung der Aufgabe helfen Kenntnisse über...“ „Sie funktioniert ähnlich wie...“ „Folgende Inhalte kommen erstmals vor:“ (evtl. Beispiel, Bild) • Erläuterung des neuen Konzepts dieser Gruppen • Erklärung des 1. Elements • Erklärung des 2. Elements • Erklärung des 3. Elements

Tabelle 3.5: Aufgabentyp 2 – Zuordnungsaufgabe

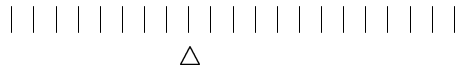
Frage	„Stelle zunächst einen Wert ein.“ m x n  „Vervollständige nun das angegebene HTML-Fragment. Verwende den zuvor eingestellten Wert [x] in HTML-Schreibweise [xhtml].“ Lückentext [text1] _____ [einhtml] Lückentext [text2]
Lösung	Sei δ der Toleranzabstand für ein „richtiges“ Umsetzen von x. Sei δ' der Abstand, bis zu welchem eine Umsetzung „zumindest in die richtige Richtung geht“. **if** text1 richtig **and** text2 richtig **and** xhtml$-\delta \leq$ einhtml $\leq$ xhtml$+\delta$ **then** feed_richtig **else** feed_falsch; **if** einhtml $<$ xhtml$-\delta'$ **then** feed_sehr_klein **fi;** **if** xhtml$-\delta' \leq$ einhtml $<$ xhtml$-\delta$ **then** feed_klein **fi;** **if** xhtml$-\delta \leq$ einhtml $\leq$ xhtml$+\delta$ **then** feed_ok **fi;** **if** xhtml$+\delta <$ einhtml $\leq$ xhtml$+\delta'$ **then** feed_gross **fi;** **if** xhtml$+\delta' <$ einhtml **then** feed_sehr_gross **fi** **fi**
Feedback	feed_richtig: „*** Prima, die Antwort ist richtig!“; Darstellen des Hypertext-Ergebnisses unter Verwendung des Wertes x $\longrightarrow$ Item bekannt! feed_falsch: „Deine Lösung enthält Fehler. Du kannst später eine neue Übung starten und dich wieder an dieser Aufgabe versuchen. Denke dann nochmal genauer darüber nach.“ $\longrightarrow$ Item nicht bekannt! feed_ok: „Den eingestellten Wert hast du richtig umgesetzt.“ feed_sehr_klein: „Der Wert in deinem HTML-Code ist viel kleiner als der auf der Skala.“ feed_klein: „... kleiner als ...“ feed_gross: „... größer als ...“ feed_sehr_gross: „... viel größer als ...“
Hilfe	„Zur Lösung der Aufgabe helfen Kenntnisse über...“ „Sie funktioniert ähnlich wie...“ „Folgende Inhalte kommen erstmals vor:“ (evtl. Beispiel, Bild) • ggf. Erläuterung eines neuen Konzeptes • Erklärungen zum Lückentext • Erklärung zu dem numerischen Wert

Tabelle 3.6: Aufgabentyp 3 – Umsetzungsaufgabe Schieberegler

<table>
<tr><td>Frage</td><td>„Wähle zunächst einen Bereich aus der Abbildung.“
<table><tr><td>$(oben, links)$</td><td>$(oben, links + \varepsilon)$</td></tr><tr><td>$(oben + \delta, links)$</td><td>$(oben + \delta, links + \varepsilon)$</td></tr></table>
„Vervollständige nun das angegebene HTML-Fragment. Verwende dem ausgewählten Bereich [x1, x2] entsprechende Werte in HTML-Schreibweise [xhtml1 und xhtml2].“
Lückentext [text1] ___ [einhtml1]
___ [einhtml2] Lückentext [text2]</td></tr>
<tr><td>Lösung</td><td>if text1 richtig and text2 richtig and xhtml1 $-\frac{1}{2}\delta <$ einhtml1 $<$ xhtml1 $+\frac{1}{2}\delta$ and xhtml2 $-\frac{1}{2}\varepsilon <$ einhtml2 $<$ xhtml2 $+\frac{1}{2}\varepsilon$
then feed_richtig
else feed_falsch;
if einhtml1 $\leq$ xhtml1 $-\frac{1}{2}\delta$ then feed_einhtml1_klein fi;
if einhtml2 $\leq$ xhtml2 $-\frac{1}{2}\varepsilon$ then feed_einhtml2_klein fi;
if xhtml1 $-\frac{1}{2}\delta <$ einhtml1 $<$ xhtml1 $+\frac{1}{2}\delta$ and xhtml2 $-\frac{1}{2}\varepsilon$
$<$ einhtml2 $<$ xhtml2 $+\frac{1}{2}\varepsilon$ then feed_ok fi;
if xhtml1 $+\frac{1}{2}\delta \leq$ einhtml1 then feed_einhtml1_gross fi;
if xhtml2 $+\frac{1}{2}\varepsilon \leq$ einhtml2 then feed_einhtml2_gross fi
fi</td></tr>
<tr><td>Feedback</td><td>feed_richtig: „*** Prima, die Antwort ist richtig!“; Darstellen des Ergebnisses unter Verwendung von $(x_1, x_2) \longrightarrow$ Item bekannt!
feed_falsch: „Deine Lösung enthält Fehler. Du kannst später eine neue Übung starten und dich wieder an dieser Aufgabe versuchen. Denke dann nochmal genauer darüber nach.“ $\longrightarrow$ Item nicht bekannt!
feed_ok: „Den ausgewählten Bereich hast du richtig umgesetzt.“
feed_einhtml1_klein: „Der erste Wert in deinem HTML-Code ist zu klein, um den ausgewählten Bereich zu repräsentieren.“
feed_einhtml2_klein: „... zweite Wert ... zu klein ...“
feed_einhtml1_gross: „... erste Wert ... zu groß ...“
feed_einhtml2_gross: „... zweite Wert ... zu groß ...“</td></tr>
<tr><td>Hilfe</td><td>„Zur Lösung der Aufgabe helfen Kenntnisse über...“
„Sie funktioniert ähnlich wie...“
„Folgende Inhalte kommen erstmals vor:“ (evtl. Beispiel, Bild)
• ggf. Erläuterung eines neuen Konzeptes
• Erklärungen zum Lückentext
• Erklärungen zu den numerischen Werten</td></tr>
</table>

Tabelle 3.7: Aufgabentyp 4 – Umsetzungsaufgabe Image Map

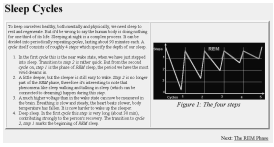
Frage	„Es ist deine Aufgabe, den abgebildeten Hypertext-Entwurf zu implementieren.“ (evtl. mit einer Handlung versehen)  „Vervollständige dazu das angegebene HTML-Fragment.“ HTML-Code mit Lücken [`text1` bis `textn`]
Lösung	**if** `text1` richtig **and** ... **and** `textn` richtig (Zulassung von Alternativen!) **then** `feed_richtig` **else** **if** `text1` richtig **then** `farbe1` := `gruen` **else** `farbe1` := `rot` **fi;** ⋮ **if** `textn` richtig **then** `farben` := `gruen` **else** `farben` := `rot` **fi;** `feed_falsch` (`farbe1`, ..., `farben`) **fi**
Feedback	`feed_richtig`: „*** Prima, die Antwort ist komplett richtig!“ ⟶ Item bekannt! `feed_falsch` (`farbe1`, ..., `farben`): „Deine Lösung enthält Fehler. Du kannst später eine neue Übung starten und dich wieder an dieser Aufgabe versuchen. Denke dann nochmal genauer darüber nach.“; Wiedergeben des ausgefüllten HTML-Codes mit farbigem Markieren der richtigen und falschen Antworten ⟶ Item nicht bekannt!
Hilfe	„Zur Lösung der Aufgabe helfen Kenntnisse über...“ „Sie funktioniert ähnlich wie...“ „Folgende Inhalte kommen erstmals vor:“ (evtl. Beispiel, Bild) • ggf. Erläuterung eines neuen Konzeptes • Erklärungen zum Lückentext

Tabelle 3.8: Aufgabentyp 5 – Implementierungsaufgabe vorgegeben

Frage	„Wähle als erstes ein Themengebiet, zu welchem du eine Website aufbauen möchtest.“(evtl. mit (einer) Handlung(en) versehen) „Ich nehme das Thema...“ [`gebiet`] Themengebiet 1 / Themengebiet 2 „Wähle als nächstes ein passendes Design.“ „Ich nehme das Design...“ [`design`] Design 1 / Design 2 „Implementiere nun deinen Entwurf, indem du das angegebene HTML-Fragment vervollständigst.“ HTML-Code mit Lücken [`text1` bis `textn`]
Lösung	Fallunterscheidung: Formulieren einer eigenen Lösung für jede der (hier 4) möglichen Wahlkombinationen. Angegeben ist der Fall `gebiet` = **“**`Themengebiet 1`**”** **and** `design` = **“**`Design 1`**”**. **if** `text1` richtig **and** ... **and** `textn` richtig (Zulassung von Alternativen!) **then** `feed_richtig_gebiet1_design1` **else if** `text1` richtig **then** `farbe1` := `gruen` **else** `farbe1` := `rot` **fi;** ⋮ **if** `textn` richtig **then** `farben` := `gruen` **else** `farben` := `rot` **fi;** `feed_falsch` (`farbe1`, ..., `farben`) **fi**
Feedback	`feed_richtig_gebiet1_design1`: „*** Prima, die Antwort ist komplett richtig!“; Abbildung des Ergebnisses mit dem ersten Themengebiet und dem ersten Design −→ Item bekannt! `feed_richtig_gebiet1_design2`, `feed_richtig_gebiet2_design1`, `feed_richtig_gebiet2_design2`: analog `feed_falsch` (`farbe1`, ..., `farben`): „Deine Lösung enthält Fehler. Du kannst später eine neue Übung starten und dich wieder an dieser Aufgabe versuchen. Denke dann nochmal genauer darüber nach.“; Wiedergeben des ausgefüllten Codes mit farbigem Markieren der richtigen und falschen Antworten −→ Item nicht bekannt!
Hilfe	„Zur Lösung der Aufgabe helfen Kenntnisse über...“ „Sie funktioniert ähnlich wie...“ „Folgende Inhalte kommen erstmals vor:“ (evtl. Beispiel, Bild) • ggf. Erläuterung eines neuen Konzeptes • Erklärungen zum Lückentext (unabhängig vom Design)

Tabelle 3.9: Aufgabentyp 6 – Implementierungsaufgabe kreativ

3.3 Implementierung mit AdAsTra/Prola 2

Vorgang Das Lehr- und Autorensystem AdAsTra verwaltet mehrere Domains gleichzeitig. Es muß ein neuer Kurs „HTML“ hinzugefügt werden, der im wesentlichen aus der Menge seiner Items sowie der Basisdatei besteht. Zu einem Item gehört eine Spalte in der Basismatrix und seine Aufgabendatei.

Die Aufgaben werden in der AdAsTra-internen Programmiersprache Prola 2 definiert (“PROblem LAnguage”, beschrieben in [Sch99], S. 93 – 112[3]; Prola 1 in [Lud95], Teil I, S. 19 – 29). Für Frage, Feedback und Hilfe stellt AdAsTra eine codegenerierende Entwicklungsoberfläche zur Verfügung; die Lösung wird direkt in Prola 2 implementiert.

Die Fragestellung wird aus einer Anzahl von Gestaltungselementen wie Text, Schiebereglern, Eingabefeldern usw. aufgebaut. Ich verwende die Komponenten wie in Tabelle 3.4 – 3.9 spezifiziert. Überschrift ist jeweils eine Grafik, die angibt, in welchem Themenbereich man sich befindet. Zu den bisherigen sieben kommt noch ein achter hinzu, der die kapitelübergreifenden „*“-Items umfaßt. Unter der Hauptüberschrift wird der Itemname angegeben. Es kommen selbsterstellte Abbildungen zum Einsatz sowie in Aufgaben zu Hintergrundmusik und nichtvisuellen Browsern Ton.

Die Lösung erzeugt als einziger der vier Aufgabenteile keine Bildschirmausgabe. Man schreibt eine Prola-2-Funktion, welche die Benutzerantwort überprüft und das entsprechende Feedback aufruft. Bei der Beurteilung von komplexen Eingaben ist mitunter eine gewisse Unschärfe angebracht – wie in Aufgabe 3 (Toleranzabstand δ) oder wenn man in Eingabefeldern alle Varianten von Groß- und Kleinschreibung durchgehen läßt (Anwenden der unterstützenden Prola-Funktion `LOWERCASE`), weil HTML, zumindest was die Tagnamen betrifft, nicht “case sensitive” ist.

Die Feedbackerstellung geht ähnlich der Prozedur für die Frage vonstatten, jedoch ohne interaktive Elemente und mit dem Zusatz, daß für jede Rückmeldefunktion, auch Teilfeedback, ein Ergebniswert festgelegt wird, der angibt, ob die Antwort des Lerners *insgesamt* als richtig (`CORRECT`) oder falsch (`WRONG`) bewertet wird.

Das Hilfefenster ist ebenfalls eine Zusammenstellung von grafischen Gestaltungselementen – gänzlich ohne Rückgabewerte und Interaktivität, sondern ausschließlich aus Text, Grafik und eventuell Ton bestehend.

Das Implementieren und Übersetzen besonders der Aufgabentypen mit größerem Handlungsspielraum sowie nachfolgend des 144 Items umfassenden Kurses ist auch ein Härtetest für die noch junge Software AdAsTra. Ent-

[3] ab AdAsTra 2.0.3/2.0.4 nochmals erweitert, u. a. um die Ausgabe von Variableninhalten sowie Regeln für die implizite Typumwandlung

deckte Bugs und Verbesserungsvorschläge sind in die aktuellen Versionen eingeflossen oder werden in künftigen berücksichtigt.

In zwei Punkten muß die Implementierung von der Spezifikation abweichen, weil AdAsTra dynamische Lösungen und Feedback derzeit nur so weit zuläßt, wie es um Textausgaben geht. Farbdefinitionen und Schriftattributen können noch keine Parameter übergeben werden.

Als Kompromißlösung werden für Aufgaben mit vergrößertem Handlungsspielraum (Typen 3 und 4) Feedbacks mit ausgewählten repräsentativen Fällen vorbereitet. Das Markieren falscher Lückentexteingaben (Typen 5 und 6) erfolgt nicht mittels roter Textfarbe, sondern durch Erweiterung der betreffenden Strings um `" <- Fehler!"`

Die Algorithmen der einfachen Aufgabentypen (1 und 2) können trotz Einschränkungen verlustlos umgesetzt werden.

Typenschablonen Die Dateien der sechs implementierten Aufgabenprototypen dienen als wiederverwendbare Schablonen zur Erstellung der eigentlichen Aufgaben.

Als Beispiel einer Aufgabendatei sei der Prola-Code der Umsetzungsaufgabe mit Schieberegler (Typ 3) angegeben.

- Frageprozedur:

```
QUESTION()
{
 VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
  FGCOLOR(RGB(0,0,0)),BGCOLOR(RGB(220,140,140)),ALIGN(LEFT),
  IMAGE(FILE("ueberschrift.gif")),
  PG("Itemname",FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(18))),
  PG(" "),
  PG("Stelle zunaechst einen Wert ein.",FONT(FAMILY("Arial"),
   STYLE(BOLD),SIZE(14))),
  SLIDER(NAME("x"),MINIMUM(0),MAXIMUM(100),MINOR(5),MAJOR(10),
   ORIENTATION(HORIZONTAL),FONT(FAMILY("Arial"),STYLE(PLAIN),
    SIZE(14)),FGCOLOR(RGB(0,0,0)),ALIGN(LEFT)),
  PG(" "),
  PG("Vervollstaendige nun das angegebene HTML-Fragment. Verwende
   den zuvor eingestellten Wert in HTML-Schreibweise.",
   FONT(FAMILY("Arial"),STYLE(BOLD),SIZE(14))),
  HORIZONTAL(FONT(FAMILY("Courier New"),STYLE(PLAIN),SIZE(16)),
   BGCOLOR(RGB(190,190,190)),ALIGN(LEFT), PG("Text"),
   INPUT(NAME("text1"),SIZE(10)), PG("Text"),
   INPUT(NAME("einhtml"),SIZE(5),BGCOLOR(RGB(190,180,95))),
   PG("Text"), INPUT(NAME("text2"),SIZE(10)), PG("Text")
  )
 );
}
```

- Lösungsprozedur:

```
SOLUTION()
{
xhtml:=x; m:=0; n:=100; mhtml:=m; nhtml:=n; delta:=5;
deltastrich:=25;
if (LOWERCASE(text1) == "richtig") and
 (LOWERCASE(text2) == "richtig") and (xhtml-delta <= einhtml)
 and (xhtml+delta >= einhtml)
 then FEED_RICHTIG();
 if (x == m) then FEED_M();
  else
   if (x == n) then FEED_N();
    else FEED_NORMAL();
   endif;
 endif;
 else
  FEED_FALSCH();
  if (xhtml-deltastrich > einhtml) then FEED_SEHR_KLEIN();
  endif;
  if (xhtml-deltastrich <= einhtml) and (xhtml-delta > einhtml)
   then FEED_KLEIN();
  endif;
  if (xhtml-delta <= einhtml) and (xhtml+delta >= einhtml)
   then FEED_OK();
  endif;
  if (xhtml+delta < einhtml) and (xhtml+deltastrich >= einhtml)
   then FEED_GROSS();
  endif;
  if (xhtml+deltastrich < einhtml) then FEED_SEHR_GROSS();
  endif;
endif;
}
```

- Feedbackfunktionen (richtig):

```
FEED_RICHTIG()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(0,80,0)),ALIGN(LEFT),
   HORIZONTAL(
    IMAGE(FILE("stern.gif")),
    IMAGE(FILE("stern.gif")),
    IMAGE(FILE("stern.gif"))
   ),
   PG("Prima, die Antwort ist richtig!"),
   PG(" ")
  );
  RESULT := CORRECT;
```

```
}

FEED_M()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(0,80,0)),ALIGN(LEFT),
   PG("Hypertext-Ergebnis mit m")
  );
  RESULT := CORRECT;
}

FEED_N()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(0,80,0)),ALIGN(LEFT),
   PG("Hypertext-Ergebnis mit n")
  );
  RESULT := CORRECT;
}

FEED_NORMAL()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(0,80,0)),ALIGN(LEFT),
   PG("Hypertext-Ergebnis mit Durchschnittswert")
  );
  RESULT := CORRECT;
}
```

- Feedbackfunktionen (falsch):

```
FEED_FALSCH()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(100,0,0)),ALIGN(LEFT),
   PG("Deine Loesung enthaelt Fehler."),
   PG("Du kannst spaeter eine neue Uebung starten und dich wieder ",
   "an dieser Aufgabe versuchen. Denke dann nochmal genauer darueber
   nach."),
   PG(" ")
  );
  RESULT := WRONG;
}

FEED_OK()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(0,80,0)),ALIGN(LEFT),
   PG("Den eingestellten Wert hast du richtig umgesetzt.")
```

```
  );
  RESULT := WRONG;
}

FEED_SEHR_KLEIN()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(100,0,0)),ALIGN(LEFT),
   PG("Der Wert in deinem HTML-Code ist viel kleiner als der auf der
    Skala.")
  );
  RESULT := WRONG;
}

FEED_KLEIN()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(100,0,0)),ALIGN(LEFT),
   PG("Der Wert in deinem HTML-Code ist kleiner als der auf der
    Skala.")
  );
  RESULT := WRONG;
}

FEED_GROSS()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(100,0,0)),ALIGN(LEFT),
   PG("Der Wert in deinem HTML-Code ist groesser als der auf der
    Skala.")
  );
  RESULT := WRONG;
}

FEED_SEHR_GROSS()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
   FGCOLOR(RGB(255,255,255)),BGCOLOR(RGB(100,0,0)),ALIGN(LEFT),
   PG("Der Wert in deinem HTML-Code ist viel groesser als der auf
    der Skala.")
  );
  RESULT := WRONG;
}
```

- Hilfeprozedur:

```
HELP()
{
  VERTICAL(FONT(FAMILY("Arial"),STYLE(PLAIN),SIZE(14)),
```

```
    FGCOLOR(RGB(0,0,0)),BGCOLOR(RGB(220,140,140)),ALIGN(LEFT),
    PG("Zur Loesung der Aufgabe helfen Kenntnisse ueber..."),
    PG("Sie funktioniert aehnlich wie..."),
    PG(" "),
    PG(" "),
    PG("Folgende Inhalte kommen erstmals vor:"),
    PG(" "),
    PG("Erlaeuterung eines neuen Konzeptes"),
    PG(" "),
    PG("Erklaerungen zum Lueckentext"),
    PG(" "),
    PG("Erklaerung zu dem numerischen Wert")
   );
}
```

Programmierung des Kurses Zur Implementierung jeder der 144 Aufgaben wird der Code der zugehörigen Typenschablone übernommen, gegebenenfalls um Antwortvorgaben, Lücken usw. ergänzt bzw. gekürzt und schließlich mit dem textuellen und grafischen Inhalt aus Anhang B.1 gefüllt.

Als Beispiel eines einsatzbereiten Items sind Screenshots der Aufgabe 8 in Abb. 3.3 – 3.5 zu sehen.

Die endgültigen Prola-Skripte und alle übrigen benötigten Dateien werden auf einer CD-ROM gesammelt. Ihr Inhalt ist in Anhang B.2 aufgelistet.

Release Notes und Ausblick Zum Zeitpunkt der Fertigstellung des Dokumentes waren erst die Aufgaben der Items 1 bis 29 (die **Grundregeln**) implementiert. Mit dieser Menge lief AdAsTra Version 2.0.7 nicht stabil. Jedoch konnten die Items in kleineren Mengen für sich erfolgreich getestet werden.

Ich habe deshalb als vorläufige Implementierung des Kurses die Domains

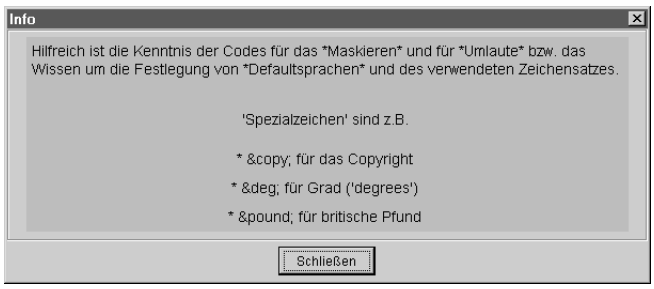


Abbildung 3.3: Screenshot einer Hilfestellung in AdAsTra

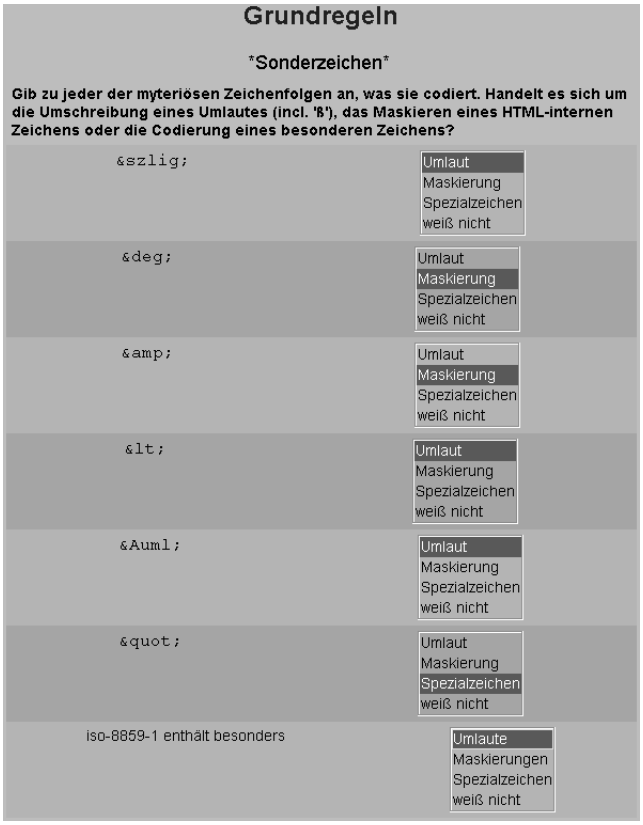


Abbildung 3.4: Screenshot einer Frage in AdAsTra

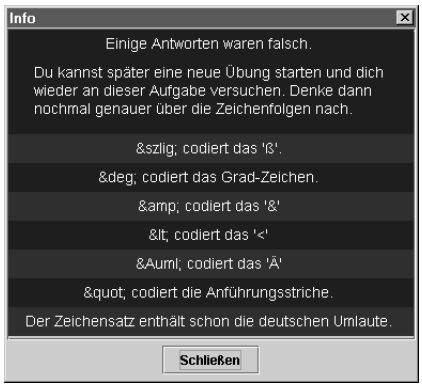


Abbildung 3.5: Screenshot eines Feedbacks in AdAsTra

- „HDemo“ (6 austauschbare **Grundregeln**-Items ohne Voraussetzungsbeziehungen),
- „Grundregeln“ (die 29 **Grundregeln**-Items mit entsprechend kleiner Basis)
- und „HTML“ (vollständige Itemmenge und Basis, jedoch nur Platzhalter-Aufgaben[4] ab Item 29)

erstellt.

Es ist also noch umfangreiche und gründliche Implementierungs- bzw. Testarbeit zu leisten, bevor der Kurs tatsächlich in der Praxis eingesetzt werden kann.

Im Vorfeld der Studienarbeit wurde angedacht, eine Standalone-Variante von AdAsTra zu schaffen. Ohne Entwicklungsumgebung und speziell auf die Benutzer abgestimmt, könnte das Lehrsystem mit dem Kurs “HTML” zusammen an Lernende und Lehrende abgegeben werden.

Weiterhin dürfte ein Einsatz des fertigen Kurses im neuen, Internetbasierten AdAsTra, ebenfalls von Ralf Schreiber, in Frage kommen.

[4] Alle Grafiken und Sounddateien sind fertig.

Literaturverzeichnis

[Alb99] Josef Lukas, Dietrich Albert. Knowledge Structures: What They Are and How They Can be Used in Cognitive Psychology, Test Theory, and the Design of Learning Environments. In Dietrich Albert and Josef Lukas, editors, *Knowledge Spaces: Theories, Empirical Research, Applications*, chapter 1, pages 3–12. Lawrence Erlbaum Associates, Mahwah, NJ, 1999.

[Bro92] Manfred Broy. *Informatik. Eine grundlegende Einführung. Teil I. Problemnahe Programmierung.* Springer, Heidelberg, 1992.

[Csi99] Mihaly Csikszentmihalyi. *Das flow-Erlebnis. Jenseits von Angst und Langeweile: im Tun aufgehen.* Klett-Cotta, Stuttgart, 1999. 7. Auflage.

[Dow93] Cornelia E. Dowling. Applying the Basis of a Knowledge Space for Controlling the Questioning of an Expert. *Journal of Mathematical Psychology*, 37(1):21–48, 1993.

[Ede96] Walter Edelmann. *Lernpsychologie.* Beltz, Weinheim, 1996. 5., vollständig überarbeitete Auflage.

[Fre99] Olaf Freitag. Graphische Darstellung von Expertenurteilen. Diplomarbeit, TU Braunschweig, Juli 1999.

[Her94] Michael Herczeg. *Software-Ergonomie. Grundlagen der Mensch-Computer-Kommunikation.* Addison-Wesley, Bonn, 1994.

[Hoc98] Cornelia E. Dowling, Cord Hockemeyer. Computing the Intersection of Knowledge Spaces Using Only Their Bases. In Cornelia E. Dowling, Fred S. Roberts and Peter Theuns, editors, *Recent Progress in Mathematical Psychology. Psychophysics, Knowledge, Representation, Cognition, and Measurement*, chapter 8, pages 133–141. Lawrence Erlbaum Associates, Mahwah, NJ, London, 1998.

[Kal94] Rainer Kaluscha. Ein effizienter Algorithmus zur Expertenbefragung. Diplomarbeit, TU Braunschweig, September 1994.

[Lew97] Doris Lewalter. *Lernen mit Bildern und Animationen.* Waxmann, Münster, 1997.

[Lud95] Andreas H. Ludwig. Eine themenorientierte Programmiersprache für die Präsentation und Auswertung computergestützter Test- und Übungsaufgaben. Diplomarbeit, TU Braunschweig, September 1995.

[Mü98] Stefan Münz. SELFHTML 7.0. `http://www.teamone.de/selfhtml`, April 1998.

[Nef99] Stefan Münz, Wolfgang Nefzger. *HTML 4.0 Referenz.* Franzis, Feldkirchen, 1999.

[Sch95] Franz Schanda. *Computer-Lernprogramme: Wie damit gelernt wird. Wie sie entwickelt werden. Was sie im Unternehmen leisten.* Beltz, Weinheim, 1995.

[Sch99] Ralf Schreiber. Portierung und Erweiterung der adaptiven CBT-Software AdAsTra. Studienarbeit, TU Braunschweig, Dezember 1999.

Anhang A

Darstellungen der Wissensstruktur

A.1 SFBen

Items

1. *Regeln*
2. *Steuerbefehle*
3. *Tags*
4. *Verschachtelung*
5. *Attribute*
6. *Grundgeruest*
7. *Editieren*
8. *Sonderzeichen*
9. *Umlaute*
10. *Maskieren*
11. *Farbdefinition*
12. *hexadezimal*
13. *Farbnamen*
14. *KommentareCredits*
15. *Kommentare*
16. *Credits*
17. *Dateikonventionen*
18. *Dateinamen*
19. *Defaultdatei*
20. *Organisation*
21. *Stil*
22. *Browser*
23. *zweckentfremden*
24. *Zeichen*
25. *Endtags*
26. *Linktexte*
27. *Grafikeinsatz*
28. *Groessenangabe*
29. *Aufloesung*
30. *Dateiangaben*
31. *Titel*
32. *Metaangaben*

33. *Beschreibung*
34. *Auslesen*
35. *Stichwortsprache*
36. *Defaultsprachen*
37. *Originaladresse*
38. *Weiterleitung*
39. *dateiweit*
40. *Farben*
41. *Hintergrundfarbe**
42. *Textfarbe*
43. *Hintergrundbild**
44. *Hintergrundmusik*
45. *Textgestaltung*
46. *Ueberschriften*
47. *Ueberschriftdefinition*
48. *Ausrichten**
49. *Absaetze*
50. *Absatzdefinition*
51. *Umbrueche*
52. *ErzwingenUmbruch*
53. *VerhindernUmbruch*
54. *Leerzeichen*
55. *ErlaubenUmbruch*
56. *Nummerlisten*
57. *Nummerlistendefinition*
58. *Nummertyp*
59. *Startnummer*
60. *Aufzaehllisten*
61. *Aufzaehllistendefinition*
62. *Definitionslisten*
63. *Definitionslistendefinition*
64. *ZitateAdressen*
65. *Zitat*
66. *Adresse*
67. *praeformatiert*
68. *Formatierungsdefinition*
69. *Breite**
70. *logisch*
71. *physisch*
72. *Schriftoperationen*
73. *Schriftgroesse*
74. *Schriftfarbe*
75. *Schriftart*
76. *Normalschriftgroesse*
77. *Bereiche*
78. *Trennlinien*
79. *Liniendefinition*
80. *Nichtschattierung*
81. *Linienhoehe*
82. *Tabellen*

83. *Tabellenaufbau*
84. *Tabellendefinition*
85. *ZeilenSpalten*
86. *Spaltenvordefinition*
87. *RahmenGitterRand*
88. *Rahmen**
89. *Gitter*
90. *Rand*
91. *TabellenHoeheBreite*
92. *Hoehe**
93. *Zellenumbruch*
94. *Zellenausrichtung*
95. *Vertikalausrichten*
96. *Verbinden*
97. *Zeilenverbinden*
98. *Spaltenverbinden*
99. *Farbtabellen*
100. *BeschriftungAusrichtung*
101. *Tabellenueberschrift*
102. *Textfortsetzung**
103. *blindeTabellen*
104. *Mehrspaltigkeit*
105. *TextGrafik*
106. *Seitenraender*
107. *Farbflaechen*
108. *nichtvisuell*
109. *Zellenbeschreibung*
110. *Kopfzellenreferenz*
111. *Kategorien*
112. *Zusammenfassung*
113. *Links*
114. *Linkaufbau*
115. *Anker*
116. *Projektlinks*
117. *gleichesVerzeichnis*
118. *anderesVerzeichnis*
119. *andererAnker*
120. *andereAdressen*
121. *WWW*
122. *FTP*
123. *Gopher*
124. *Telnet*
125. *Newsgroups*
126. *EMail*
127. *Mailadresse*
128. *Empfaenger*
129. *Subject*
130. *Body*
131. *Download*
132. *Dateien*

133. *eigeneDateien*

134. *Grafik*

135. *Einbinden*

136. *Referenzieren*

137. *Alternativtext*

138. *GrafikBreiteHoehe*

139. *BeschriftenUmfliessen*

140. *Grafikbeschriften*

141. *Abstand*

142. *Grafiklinks*

143. *EinbindenGrafiklink*

144. *Maps*

SFBen aus der Baumstruktur

1. $(\{Steuerbefehle\}, Regeln)$
   $(\{Grundgeruest\}, Regeln)$
   $(\{Editieren\}, Regeln)$
   $(\{Sonderzeichen\}, Regeln)$
   $(\{Farbdefinition\}, Regeln)$
   $(\{Dateikonventionen\}, Regeln)$

2. $(\{Tags\}, Steuerbefehle)$
   $(\{Verschachtelung\},$
   $Steuerbefehle)$
   $(\{Attribute\}, Steuerbefehle)$

8. $(\{Umlaute\}, Sonderzeichen)$
   $(\{Maskieren\}, Sonderzeichen)$

11. $(\{hexadezimal, Farbnamen\},$
    $Farbdefinition)$

14. $(\{Kommentare\},$
    $KommentareCredits)$
    $(\{Credits\},$
    $KommentareCredits)$

17. $(\{Dateinamen\},$
    $Dateikonventionen)$
    $(\{Defaultdatei\},$
    $Dateikonventionen)$

21. $(\{Browser\}, Stil)$
    $(\{zweckentfremden\}, Stil)$
    $(\{Zeichen\}, Stil)$
    $(\{Endtags\}, Stil)$
    $(\{Linktexte\}, Stil)$
    $(\{Grafikeinsatz\}, Stil)$
    $(\{Groessensangabe\}, Stil)$
    $(\{Aufloesung\}, Stil)$

30. $(\{Titel\}, Dateiangaben)$
    $(\{Metaangaben\},$
    $Dateiangaben)$

32. $(\{Beschreibung\},$
    $Metaangaben)$
    $(\{Auslesen\}, Metaangaben)$
    $(\{Stichwortsprache\},$
    $Metaangaben)$
    $(\{Defaultsprachen\},$
    $Metaangaben)$
    $(\{Originaladresse\},$
    $Metaangaben)$
    $(\{Weiterleitung\},$
    $Metaangaben)$

39. $(\{Farben\}, dateiweit)$
    $(\{Hintergrundbild^*\}, dateiweit)$
    $(\{Hintergrundmusik\},$
    $dateiweit)$

40. $(\{Hintergrundfarbe^*\}, Farben)$
    $(\{Textfarbe\}, Farben)$

45. ({*Ueberschriften*}, *Textgestaltung*)
({*Absaetze*}, *Textgestaltung*)
({*Umbrueche*}, *Textgestaltung*)
({*Nummerlisten*}, *Textgestaltung*)
({*Aufzaehllisten*}, *Textgestaltung*)
({*Definitionslisten*}, *Textgestaltung*)
({*ZitateAdressen*}, *Textgestaltung*)
({*praeformatiert*}, *Textgestaltung*)
({*logisch*}, *Textgestaltung*)
({*physisch*}, *Textgestaltung*)
({*Schriftoperationen*}, *Textgestaltung*)
({*Bereiche*}, *Textgestaltung*)
({*Trennlinien*}, *Textgestaltung*)

46. ({*Ueberschriftdefinition*}, *Ueberschriften*)
({*Ausrichten**}, *Ueberschriften*)

49. ({*Absatzdefinition*}, *Absaetze*)
({*Ausrichten**}, *Absaetze*)

51. ({*ErzwingenUmbruch*}, *Umbrueche*)
({*VerhindernUmbruch*, *Leerzeichen*}, *Umbrueche*)
({*ErlaubenUmbruch*}, *Umbrueche*)

56. ({*Nummerlistendefinition*}, *Nummerlisten*)
({*Nummertyp*}, *Nummerlisten*)
({*Startnummer*}, *Nummerlisten*)

60. ({*Aufzaehllistendefinition*}, *Aufzaehllisten*)

62. ({*Definitionslistendefinition*}, *Definitionslisten*)

64. ({*Zitat*}, *ZitateAdressen*)
({*Adresse*}, *ZitateAdressen*)

67. ({*Formatierungsdefinition*}, *praeformatiert*)
({*Breite**}, *praeformatiert*)

72. ({*Schriftgroesse*, *Normalschriftgroesse*}, *Schriftoperationen*)
({*Schriftfarbe*}, *Schriftoperationen*)
({*Schriftart*}, *Schriftoperationen*)

77. ({*Ausrichten**}, *Bereiche*)

78. ({*Liniendefinition*}, *Trennlinien*)
({*Nichtschattierung*}, *Trennlinien*)
({*Linienhoehe*}, *Trennlinien*)
({*Breite**}, *Trennlinien*)
({*Ausrichten**}, *Trennlinien*)

82. ({*Tabellenaufbau*}, *Tabellen*)
({*RahmenGitterRand*}, *Tabellen*)
({*TabellenHoeheBreite*}, *Tabellen*)
({*Zellenausrichtung*}, *Tabellen*)
({*Verbinden*}, *Tabellen*)
({*Farbtabellen*}, *Tabellen*)
({*BeschriftungAusrichtung*}, *Tabellen*)

83. ({*Tabellendefinition*},
Tabellenaufbau)
({*ZeilenSpalten*},
Tabellenaufbau)
({*Spaltenvordefinition*},
Tabellenaufbau)

87. ({*Rahmen**},
RahmenGitterRand)
({*Gitter*}, *RahmenGitterRand*)
({*Rand*}, *RahmenGitterRand*)

91. ({*Hoehe**},
TabellenHoeheBreite)
({*Breite**},
TabellenHoeheBreite)
({*Zellenumbruch*},
TabellenHoeheBreite)

94. ({*Ausrichten**},
Zellenausrichtung)
({*Vertikalausrichten*},
Zellenausrichtung)

96. ({*Zeilenverbinden*}, *Verbinden*)
({*Spaltenverbinden*},
Verbinden)

99. ({*Hintergrundfarbe**},
Farbtabellen)
({*Hintergrundbild**},
Farbtabellen)

100. ({*Tabellenueberschrift*},
BeschriftungAusrichtung)
({*Ausrichten**},
BeschriftungAusrichtung)
({*Textfortsetzung**},
BeschriftungAusrichtung)

103. ({*Mehrspaltigkeit*},
blindeTabellen)
({*TextGrafik*}, *blindeTabellen*)
({*Seitenraender*},
blindeTabellen)
({*Farbflaechen*},
blindeTabellen)

108. ({*Zellenbeschreibung*},
nichtvisuell)
({*Kopfzellenreferenz*},
nichtvisuell)
({*Kategorien*}, *nichtvisuell*)
({*Zusammenfassung*},
nichtvisuell)

113. ({*Linkaufbau*}, *Links*)
({*Anker*}, *Links*)
({*Projektlinks*}, *Links*)
({*andereAdressen*}, *Links*)
({*EMail*}, *Links*)
({*Download*}, *Links*)
({*eigeneDateien*}, *Links*)

116. ({*gleichesVerzeichnis*},
Projektlinks)
({*anderesVerzeichnis*},
Projektlinks)
({*andererAnker*}, *Projektlinks*)

120. ({*WWW*}, *andereAdressen*)
({*FTP*}, *andereAdressen*)
({*Gopher*}, *andereAdressen*)
({*Telnet*}, *andereAdressen*)
({*Newsgroups*},
andereAdressen)

126. ({*Mailadresse*}, *EMail*)
({*Empfaenger*}, *EMail*)
({*Subject*}, *EMail*)
({*Body*}, *EMail*)

131. ({*Dateien*}, *Download*)

134. ({*Einbinden*}, *Grafik*)
({*GrafikBreiteHoehe*}, *Grafik*)

({*BeschriftenUmfliessen*}, *Grafik*)
({*Abstand*}, *Grafik*)
({*Grafiklinks*}, *Grafik*)
({*Maps*}, *Grafik*)

135. ({*Referenzieren*}, *Einbinden*)
({*Alternativtext*}, *Einbinden*)
({*Rahmen**}, *Einbinden*)

138. ({*Breite**}, *GrafikBreiteHoehe*)
({*Hoehe**}, *GrafikBreiteHoehe*)

139. ({*Grafikbeschriften*}, *BeschriftenUmfliessen*)
({*Ausrichten**}, *BeschriftenUmfliessen*)
({*Textfortsetzung**}, *BeschriftenUmfliessen*)

142. ({*EinbindenGrafiklink*}, *Grafiklinks*)

SFBen aus Geschwisterordnung

2. ({*Tags*}, *Verschachtelung*)
({*Tags*}, *Attribute*)
lösche: ({*Tags*}, *Steuerbefehle*)

32. ({*Beschreibung*}, *Stichwortsprache*)
lösche: ({*Beschreibung*}, *Metaangaben*)

56. ({*Nummerlistendefinition*}, *Nummertyp*)
({*Nummerlistendefinition*}, *Startnummer*)
lösche:
({*Nummerlistendefinition*}, *Nummerlisten*)

78. ({*Liniendefinition*}, *Nichtschattierung*)
({*Liniendefinition*}, *Linienhoehe*)
lösche: ({*Liniendefinition*}, *Trennlinien*)

83. ({*Tabellendefinition*}, *ZeilenSpalten*)
({*ZeilenSpalten*}, *Spaltenvordefinition*)
lösche: ({*Tabellendefinition*}, *Tabellenaufbau*)
lösche: ({*ZeilenSpalten*}, *Tabellenaufbau*)

116. ({*gleichesVerzeichnis*}, *anderesVerzeichnis*)
({*anderesVerzeichnis*}, *andererAnker*)
lösche: ({*gleichesVerzeichnis*}, *Projektlinks*)
lösche: ({*anderesVerzeichnis*}, *Projektlinks*)

126. ({*Mailadresse*}, *Empfaenger*)
({*Mailadresse*}, *Subject*)
({*Mailadresse*}, *Body*)
lösche: ({*Mailadresse*}, *EMail*)

135. ({*Referenzieren*}, *Alternativtext*)
lösche: ({*Referenzieren*}, *Einbinden*)

SFBen aus den Querverweisen

1. **lösche:** ({*Sonderzeichen*}, *Regeln*)
lösche: ({*Farbdefinition*}, *Regeln*)

lösche: ({*Dateikonventionen*}, *Regeln*)

9. ({*Umlaute*, *Defaultsprachen*}, *Sonderzeichen*)
lösche: ({*Umlaute*}, *Sonderzeichen*)

16. ({*Mailadresse*}, *Credits*)

24. ({*Umlaute*, *Defaultsprachen*}, *Zeichen*)

25. ({*Tags*}, *Endtags*)

28. ({*Alternativtext*}, *Groessenangabe*)
({*GrafikBreiteHoehe*}, *Groessenangabe*)

31. ({*Regeln*}, *Titel*)

33. ({*Regeln*}, *Beschreibung*)

34. ({*Regeln*}, *Auslesen*)

36. ({*Regeln*}, *Defaultsprachen*)

37. ({*Regeln*}, *Originaladresse*)

38. ({*WWW*}, *Weiterleitung*)

41. ({*Regeln*}, *Hintergrundfarbe**)
({*Farbdefinition*}, *Hintergrundfarbe**)

42. ({*Regeln*}, *Textfarbe*)
({*Farbdefinition*}, *Textfarbe*)

43. ({*Regeln*}, *Hintergrundbild**)

44. ({*Regeln*}, *Hintergrundmusik*)

47. ({*Regeln*}, *Ueberschriftdefinition*)

48. ({*Regeln*}, *Ausrichten**)

50. ({*Regeln*}, *Absatzdefinition*)

52. ({*Regeln*}, *ErzwingenUmbruch*)

53. ({*Regeln*}, *VerhindernUmbruch*)

54. ({*Regeln*}, *Leerzeichen*)

55. ({*Regeln*}, *ErlaubenUmbruch*)

57. ({*Regeln*}, *Nummerlistendefinition*)

61. ({*Regeln*}, *Aufzaehllistendefinition*)

63. ({*Regeln*}, *Definitionslistendefinition*)

65. ({*Regeln*}, *Zitat*)

66. ({*Regeln*}, *Adresse*)

68. ({*Regeln*}, *Formatierungsdefinition*)

69. ({*Regeln*}, *Breite**)

70. ({*Regeln*}, *logisch*)

71. ({*Regeln*}, *physisch*)

73. ({*Regeln*}, *Schriftgroesse*)

74. ({*Regeln*}, *Schriftfarbe*)
({*Farbdefinition*}, *Schriftfarbe*)

75. ({*Regeln*}, *Schriftart*)

76. ({*Regeln*}, *Normalschriftgroesse*)

79. ({*Regeln*}, *Liniendefinition*)

84. ({*Regeln*}, *Tabellendefinition*)

88. ({*Tabellendefinition*}, *Rahmen**)

89. ({*Tabellendefinition*}, *Gitter*)

90. ({*Tabellendefinition*}, *Rand*)

92. ({*Regeln*}, *Hoehe**)

93. ({*ZeilenSpalten*}, *Zellenumbruch*)

95. ({*ZeilenSpalten*}, *Vertikalausrichten*)

97. ({*ZeilenSpalten*}, *Zeilenverbinden*)

98. ({*ZeilenSpalten*}, *Spaltenverbinden*)

99. ({*Tabellendefinition*}, *Farbtabellen*)

101. ({*Regeln*}, *Tabellenueberschrift*)

102. ({*Regeln*}, *Textfortsetzung**)

104. ({*Tabellenaufbau*}, *Mehrspaltigkeit*)

105. ({*Tabellenaufbau*}, *TextGrafik*)
({*Referenzieren*}, *TextGrafik*)

106. ({*Breite**}, *Seitenraender*)
({*Bereiche*}, *Seitenraender*)
({*Tabellenaufbau*}, *Seitenraender*)

107. ({*Hintergrundfarbe**}, *Farbflaechen*)
({*Tabellenaufbau*}, *Farbflaechen*)

109. ({*ZeilenSpalten*}, *Zellenbeschreibung*)

110. ({*ZeilenSpalten*}, *Kopfzellenreferenz*)

111. ({*ZeilenSpalten*}, *Kategorien*)

112. ({*Tabellendefinition*}, *Zusammenfassung*)

114. ({*Regeln*}, *Linkaufbau*)

115. ({*Linkaufbau*}, *Anker*)

117. ({*Linkaufbau*}, *gleichesVerzeichnis*)

119. ({*Anker*}, *anderesAnker*)

121. ({*Linkaufbau*}, *WWW*)

122. ({*Linkaufbau*}, *FTP*)

123. ({*Linkaufbau*}, *Gopher*)

124. ({*Linkaufbau*}, *Telnet*)

125. ({*Linkaufbau*}, *Newsgroups*)

127. ({*Linkaufbau*}, *Mailadresse*)

132. ({*anderesVerzeichnis*}, *Dateien*)
({*WWW*}, *Dateien*)
({*FTP*}, *Dateien*)

133. ({*Linkaufbau*}, *eigeneDateien*)

136. ({*Regeln*}, *Referenzieren*)

138. ({*Referenzieren*}, *GrafikBreiteHoehe*)

139. ({*Absatzdefinition*}, *BeschriftenUmfliessen*)

140. $(\{Regeln\}, Grafikbeschriften)$
$(\{Grafikbeschriften,$
$TextGrafik\},$
$BeschriftenUmfliessen)$
lösche:
$(\{Grafikbeschriften\},$
$BeschriftenUmfliessen)$

141. $(\{Referenzieren\}, Abstand)$

143. $(\{Linkaufbau\},$
$EinbindenGrafiklink)$
$(\{Referenzieren\},$
$EinbindenGrafiklink)$

144. $(\{Anker\}, Maps)$
$(\{Referenzieren\}, Maps)$

I komplett

1. $(\{Steuerbefehle\}, Regeln)$
$(\{Grundgeruest\}, Regeln)$
$(\{Editieren\}, Regeln)$

2. $(\{Verschachtelung\},$
$Steuerbefehle)$
$(\{Attribute\}, Steuerbefehle)$

4. $(\{Tags\}, Verschachtelung)$

5. $(\{Tags\}, Attribute)$

8. $(\{Maskieren\}, Sonderzeichen)$
$(\{Umlaute, Defaultsprachen\},$
$Sonderzeichen)$

11. $(\{hexadezimal, Farbnamen\},$
$Farbdefinition)$

14. $(\{Kommentare\},$
$KommentareCredits)$
$(\{Credits\},$
$KommentareCredits)$

16. $(\{Mailadresse\}, Credits)$

17. $(\{Dateinamen\},$
$Dateikonventionen)$
$(\{Defaultdatei\},$
$Dateikonventionen)$

21. $(\{Browser\}, Stil)$
$(\{zweckentfremden\}, Stil)$
$(\{Zeichen\}, Stil)$
$(\{Endtags\}, Stil)$
$(\{Linktexte\}, Stil)$
$(\{Grafikeinsatz\}, Stil)$
$(\{Groessensangabe\}, Stil)$
$(\{Aufloesung\}, Stil)$

24. $(\{Umlaute, Defaultsprachen\},$
$Zeichen)$

25. $(\{Tags\}, Endtags)$

28. $(\{Alternativtext\},$
$Groessenangabe)$
$(\{GrafikBreiteHoehe\},$
$Groessenangabe)$

30. $(\{Titel\}, Dateiangaben)$
$(\{Metaangaben\},$
$Dateiangaben)$

31. $(\{Regeln\}, Titel)$

32. $(\{Auslesen\}, Metaangaben)$
$(\{Stichwortsprache\},$
$Metaangaben)$
$(\{Defaultsprachen\},$
$Metaangaben)$
$(\{Originaladresse\},$
$Metaangaben)$
$(\{Weiterleitung\},$
$Metaangaben)$

33. $(\{Regeln\}, Beschreibung)$

34. ({*Regeln*}, *Auslesen*)

35. ({*Beschreibung*},
Stichwortsprache)

36. ({*Regeln*}, *Defaultsprachen*)

37. ({*Regeln*}, *Originaladresse*)

38. ({*WWW*}, *Weiterleitung*)

39. ({*Farben*}, *dateiweit*)
({*Hintergrundbild**}, *dateiweit*)
({*Hintergrundmusik*},
dateiweit)

40. ({*Hintergrundfarbe**}, *Farben*)
({*Textfarbe*}, *Farben*)

41. ({*Regeln*}, *Hintergrundfarbe**)
({*Farbdefinition*},
Hintergrundfarbe*)

42. ({*Regeln*}, *Textfarbe*)
({*Farbdefinition*}, *Textfarbe*)

43. ({*Regeln*}, *Hintergrundbild**)

44. ({*Regeln*}, *Hintergrundmusik*)

45. ({*Ueberschriften*},
Textgestaltung)
({*Absaetze*}, *Textgestaltung*)
({*Umbrueche*}, *Textgestaltung*)
({*Nummerlisten*},
Textgestaltung)
({*Aufzaehllisten*},
Textgestaltung)
({*Definitionslisten*},
Textgestaltung)
({*ZitateAdressen*},
Textgestaltung)
({*praeformatiert*},
Textgestaltung)
({*logisch*}, *Textgestaltung*)
({*physisch*}, *Textgestaltung*)
({*Schriftoperationen*},
Textgestaltung)
({*Bereiche*}, *Textgestaltung*)
({*Trennlinien*}, *Textgestaltung*)

46. ({*Ueberschriftdefinition*},
Ueberschriften)
({*Ausrichten**},
Ueberschriften)

47. ({*Regeln*},
Ueberschriftdefinition)

48. ({*Regeln*}, *Ausrichten**)

49. ({*Absatzdefinition*},
Absaetze)
({*Ausrichten**}, *Absaetze*)

50. ({*Regeln*}, *Absatzdefinition*)

51. ({*ErzwingenUmbruch*},
Umbrueche)
({*VerhindernUmbruch*,
Leerzeichen}, *Umbrueche*)
({*ErlaubenUmbruch*},
Umbrueche)

52. ({*Regeln*},
ErzwingenUmbruch)

53. ({*Regeln*},
VerhindernUmbruch)

54. ({*Regeln*}, *Leerzeichen*)

55. ({*Regeln*}, *ErlaubenUmbruch*)

56. ({*Nummertyp*}, *Nummerlisten*)
({*Startnummer*},
Nummerlisten)

57. ({*Regeln*}, *Nummerlistendefinition*)
58. ({*Nummerlistendefinition*}, *Nummertyp*)
59. ({*Nummerlistendefinition*}, *Startnummer*)
60. ({*Aufzaehllistendefinition*}, *Aufzaehllisten*)
61. ({*Regeln*}, *Aufzaehllistendefinition*)
62. ({*Definitionslistendefinition*}, *Definitionslisten*)
63. ({*Regeln*}, *Definitionslistendefinition*)
64. ({*Zitat*}, *ZitateAdressen*)
    ({*Adresse*}, *ZitateAdressen*)
65. ({*Regeln*}, *Zitat*)
66. ({*Regeln*}, *Adresse*)
67. ({*Formatierungsdefinition*}, *praeformatiert*)
    ({*Breite**}, *praeformatiert*)
68. ({*Regeln*}, *Formatierungsdefinition*)
69. ({*Regeln*}, *Breite**)
70. ({*Regeln*}, *logisch*)
71. ({*Regeln*}, *physisch*)
72. ({*Schriftgroesse*, *Normalschriftgroesse*}, *Schriftoperationen*)
    ({*Schriftfarbe*}, *Schriftoperationen*)
    ({*Schriftart*}, *Schriftoperationen*)
73. ({*Regeln*}, *Schriftgroesse*)
74. ({*Regeln*}, *Schriftfarbe*)
    ({*Farbdefinition*}, *Schriftfarbe*)
75. ({*Regeln*}, *Schriftart*)
76. ({*Regeln*}, *Normalschriftgroesse*)
77. ({*Ausrichten**}, *Bereiche*)
78. ({*Nichtschattierung*}, *Trennlinien*)
    ({*Linienhoehe*}, *Trennlinien*)
    ({*Breite**}, *Trennlinien*)
    ({*Ausrichten**}, *Trennlinien*)
79. ({*Regeln*}, *Liniendefinition*)
80. ({*Liniendefinition*}, *Nichtschattierung*)
81. ({*Liniendefinition*}, *Linienhoehe*)
82. ({*Tabellenaufbau*}, *Tabellen*)
    ({*RahmenGitterRand*}, *Tabellen*)
    ({*TabellenHoeheBreite*}, *Tabellen*)
    ({*Zellenausrichtung*}, *Tabellen*)
    ({*Verbinden*}, *Tabellen*)
    ({*Farbtabellen*}, *Tabellen*)
    ({*BeschriftungAusrichtung*}, *Tabellen*)
83. ({*Spaltenvordefinition*}, *Tabellenaufbau*)
84. ({*Regeln*}, *Tabellendefinition*)

85. ({*Tabellendefinition*},
ZeilenSpalten)

86. ({*ZeilenSpalten*},
Spaltenvordefinition)

87. ({*Rahmen**},
RahmenGitterRand)
({*Gitter*}, *RahmenGitterRand*)
({*Rand*}, *RahmenGitterRand*)

88. ({*Tabellendefinition*},
*Rahmen**)

89. ({*Tabellendefinition*}, *Gitter*)

90. ({*Tabellendefinition*}, *Rand*)

91. ({*Hoehe**},
TabellenHoeheBreite)
({*Breite**},
TabellenHoeheBreite)
({*Zellenumbruch*},
TabellenHoeheBreite)

92. ({*Regeln*}, *Hoehe**)

93. ({*ZeilenSpalten*},
Zellenumbruch)

94. ({*Ausrichten**},
Zellenausrichtung)
({*Vertikalausrichten*},
Zellenausrichtung)

95. ({*ZeilenSpalten*},
Vertikalausrichten)

96. ({*Zeilenverbinden*}, *Verbinden*)
({*Spaltenverbinden*},
Verbinden)

97. ({*ZeilenSpalten*},
Zeilenverbinden)

98. ({*ZeilenSpalten*},
Spaltenverbinden)

99. ({*Hintergrundfarbe**},
Farbtabellen)
({*Hintergrundbild**},
Farbtabellen)
({*Tabellendefinition*},
Farbtabellen)

100. ({*Tabellenueberschrift*},
BeschriftungAusrichtung)
({*Ausrichten**},
BeschriftungAusrichtung)
({*Textfortsetzung**},
BeschriftungAusrichtung)

101. ({*Regeln*},
Tabellenueberschrift)

102. ({*Regeln*}, *Textfortsetzung**)

103. ({*Mehrspaltigkeit*},
blindeTabellen)
({*TextGrafik*}, *blindeTabellen*)
({*Seitenraender*},
blindeTabellen)
({*Farbflaechen*},
blindeTabellen)

104. ({*Tabellenaufbau*},
Mehrspaltigkeit)

105. ({*Tabellenaufbau*}, *TextGrafik*)
({*Referenzieren*}, *TextGrafik*)

106. ({*Breite**}, *Seitenraender*)
({*Bereiche*}, *Seitenraender*)
({*Tabellenaufbau*},
Seitenraender)

107. ({*Hintergrundfarbe**},
Farbflaechen)
({*Tabellenaufbau*},

Farbflaechen)

108. ({*Zellenbeschreibung*},
nichtvisuell)
({*Kopfzellenreferenz*},
nichtvisuell)
({*Kategorien*}, *nichtvisuell*)
({*Zusammenfassung*},
nichtvisuell)

109. ({*ZeilenSpalten*},
Zellenbeschreibung)

110. ({*ZeilenSpalten*},
Kopfzellenreferenz)

111. ({*ZeilenSpalten*}, *Kategorien*)

112. ({*Tabellendefinition*},
Zusammenfassung)

113. ({*Linkaufbau*}, *Links*)
({*Anker*}, *Links*)
({*Projektlinks*}, *Links*)
({*andereAdressen*}, *Links*)
({*EMail*}, *Links*)
({*Download*}, *Links*)
({*eigeneDateien*}, *Links*)

114. ({*Regeln*}, *Linkaufbau*)

115. ({*Linkaufbau*}, *Anker*)

116. ({*andererAnker*}, *Projektlinks*)

117. ({*Linkaufbau*},
gleichesVerzeichnis)

118. ({*gleichesVerzeichnis*},
anderesVerzeichnis)

119. ({*anderesVerzeichnis*},
andererAnker)
({*Anker*}, *andererAnker*)

120. ({*WWW*}, *andereAdressen*)
({*FTP*}, *andereAdressen*)
({*Gopher*}, *andereAdressen*)
({*Telnet*}, *andereAdressen*)
({*Newsgroups*},
andereAdressen)

121. ({*Linkaufbau*}, *WWW*)

122. ({*Linkaufbau*}, *FTP*)

123. ({*Linkaufbau*}, *Gopher*)

124. ({*Linkaufbau*}, *Telnet*)

125. ({*Linkaufbau*}, *Newsgroups*)

126. ({*Empfaenger*}, *EMail*)
({*Subject*}, *EMail*)
({*Body*}, *EMail*)

127. ({*Linkaufbau*}, *Mailadresse*)

128. ({*Mailadresse*}, *Empfaenger*)

129. ({*Mailadresse*}, *Subject*)

130. ({*Mailadresse*}, *Body*)

131. ({*Dateien*}, *Download*)

132. ({*anderesVerzeichnis*},
Dateien)
({*WWW*}, *Dateien*)
({*FTP*}, *Dateien*)

133. ({*Linkaufbau*}, *eigeneDateien*)

134. ({*Einbinden*}, *Grafik*)
({*GrafikBreiteHoehe*}, *Grafik*)
({*BeschriftenUmfliessen*},
Grafik)
({*Abstand*}, *Grafik*)
({*Grafiklinks*}, *Grafik*)
({*Maps*}, *Grafik*)

135. ({*Alternativtext*}, *Einbinden*)
({*Rahmen**}, *Einbinden*)

136. ({*Regeln*}, *Referenzieren*)

137. ({*Referenzieren*},
Alternativtext)

138. ({*Breite**}, *GrafikBreiteHoehe*)
({*Hoehe**}, *GrafikBreiteHoehe*)
({*Referenzieren*},
GrafikBreiteHoehe)

139. ({*Ausrichten**},
BeschriftenUmfliessen)
({*Textfortsetzung**},
BeschriftenUmfliessen)
({*Absatzdefinition*},
BeschriftenUmfliessen)
({*Grafikbeschriften*,
TextGrafik},
BeschriftenUmfliessen)

140. ({*Regeln*}, *Grafikbeschriften*)

141. ({*Referenzieren*}, *Abstand*)

142. ({*EinbindenGrafiklink*},
Grafiklinks)

143. ({*Linkaufbau*},
EinbindenGrafiklink)
({*Referenzieren*},
EinbindenGrafiklink)

144. ({*Anker*}, *Maps*)
({*Referenzieren*}, *Maps*)

A.2 Basisdateien

Matrix B_1

```
81
101
12222220000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
01222000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00210000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00201000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000001220000000000000000000000000000000000000000000000000000000000000000000000000000000000000
22222221020000000000000000000000020000000000000000000000000000000000000000000000000000000000000
00000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000001200000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000001020000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000001220000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000001220000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000
00200000200000000001222222220000000000000000000000000000000000000000000000000000000000000000000
22222220000000000001222222220000002000000000000000000000000000000000000000000000000000000000000
00000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000
00000002000000000000010000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000010000000000200000000000000000000000000000000000000000000000000000000000000
00200000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000001222222200000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000100000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000012222220000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000001000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000100000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000201000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000010000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000001000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000001222220000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000001222220000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000000122000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000000122000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000000010000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000000010000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000000001000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000000001000000000000000000000000000000000000000000000000000000
22222220000000000000000000000000000000001000000000000000000000000000000000000000000000000000000
22222220000000000000000000000000000000000100000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000001222220202222222222222222222222022222
22222220002200000000000000000000000000122222202222222222222222222222022222
22222220002200000000000000000000000000012222220222222222222222222220222222
22222220002200000000000000000000000000001222222022222222222222222220222222
```

```
22222220002020000000000000000000000000000000012222222202222222222222222222222022222
22222220002020000000000000000000000000000000012222222202222222222222222222222022222
22222220002020000000000000000000000000000000012222222202222222222222222222222022222
22222220002020000000000000000000000000000000012222222202222222222222222222222022222
22222220000000000000000000000000000000000000012200000000000000000000000000000000000
22222220000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000100000000000000000000000000000000000000
22222220000000000000000000000000000000000000212000000000000000000000000000000000000
22222220000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000012202000000000000000000000000000000000
22222220000000000000000000000000000000000000120220000000000000000000000000000000000
22222220000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000100000000000000000000000000000000000000
22222220000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000001000000000000000000000000000000000000
22222220000000000000000000000000000000000000012220000000000000000000000000000000000
22222220000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000021000000000000000000000000000000000000
22222220000000000000000000000000000000000000020100000000000000000000000000000000000
22222220000000000000000000000000000000000000001200000000000000000000000000000000000
22222220000000000000000000000000000000000000000010000000000000000000000000000000000
22222220000000000000000000000000000000000000000012000000000000000000000000000000000
22222220000000000000000000000000000000000000000001000000000000000000000000000000000
22222220000000000000000000000000000000000000000000100000000000000000000000000000000
22222220000000000000000000000000000000000000000000012200000000000000000000000000000
22222220000000000000000000000000000000000000000000001000000000000000000000000000000
22222220000000000000000000000000000000000000000000000100000000000000000000000000000
22222220000000000000000000000000000000000000000000000010000000000000000000000000000
22222220000000000000000000000000000000000000000000000001000000000000000000000000000
22222220000000000000000000000000000000000000000000000000100000000000000000000000000
22222220000000000000000000000000000000000000000000000000010000000000000000000000000
22222220002200000000000000000000000000000000000000000000001222000000000000000000000
22222220002200000000000000000000000000000000000000000000001022200000000000000000000
22222220002020000000000000000000000000000000000000000000001222000000000000000000000
22222220002020000000000000000000000000000000000000000000001022200000000000000000000
22222220000000000000000000000000000000000000000000000000000100000000000000000000000
22222220002200000000000000000000000000000000000000000000000010000000000000000000000
22222220002020000000000000000000000000000000000000000000000010000000000000000000000
22222220000000000000000000000000000000000000000000000000000001000000000000000000000
22222220000000000000000000000000000000000000000000000000000000100000000000000000000
22222220000000000000000000000000000000000000000000000000000000010000000000000000000
22222220000000000000000000000000000000000000000000000000000000001000000000000000000
22222220000000000000000000000000000000000000000000000000000000000100000000000000000
22222220000000000000000000000000000000000000000000000000000000000010000000000000000
22222220000000000000000000000000000000000000000000000000000000000001000000000000000
22222220000000000000000000000000000000000000000000000000000000000000100000000000000
22222220000000000000000000000000000000000000000000000000000000000000010000000000000
22222220000000000000000000000000000000000000000000000000000000000000001000000000000
22222220000000000000000000000000000000000000000000000000000000000000000100000000000
22222220000000000000000000000000000000000000000000000000000000000000000010000000000
22222220000000000000000000000000000000000000000000000000000000000000000001000000000
22222220000000000000000000000000000000000000000000000000000000000000000000100000000
22222220000000000000000000000000000000000000000000000000000000000000000000010000000
22222220000000000000000000000000000000000000000000000000000000000000000000001000000
22222220000000000000000000000000000000000000000000000000000000000000000000000100000
22222220000000000000000000000000000000000000000000000000000000000000000000000010000
22222220000000000000000000000000000000000000000000000000000000000000000000000001000
22222220000000000000000000000000000000000000000000000000000000000000000000000000100
22222220000000000000000000000000000000000000000000000000000000000000000000000000010
22222220000000000000000000000000000000000000000000000000000000000000000000000000001
```

Matrix B_2

```
63
65
12222222222222222222000000000000000000000000000000000000000000000000
01222000000000000000000000000000000000000000000000000000000000000000
00100000000000000000000000000000000000000000000000000000000000000000
00210000000000000000000000000000000000000000000000000000000000000000
00221000000000000000000000000000000000000000000000000000000000000000
00200122200000000000000000000000000000000000000000000000000000000000
00200010000000000000000000000000000000000000000000000000000000000000
00200001000000000000000000000000000000000000000000000000000000000000
00200000100000000000000000000000000000000000000000000000000000000000
00220000012200000000000000000000000000000000000000000000000000000000
00000000001000000000000000000000000000000000000000000000000000000000
```

```
00220000000100000000000000000000000000000000000000000000000000000000
00000000000012000000000000000000000000000000000000000000000000000000
00000000000001000000000000000000000000000000000000000000000000000000
00220000000001220000000000000000000000000000000000000000000000000000
00220000000000100000000000000000000000000000000000000000000000000000
00220000000000010000000000000000000000000000000000000000000000000000
00200000000000001000000000000000000000000000000000000000000000000000
00000000000000001220000000000000000000000000000000000000000000000000
00000000000000000100000000000000000000000000000000000000000000000000
00000000000000000010000000000000000000000000000000000000000000000000
02222000000000000001222200000000000000000000000000000200000000000000
02222000000000000000100000000000000000000000000000000000000000000000
02222000000000000000010000000000000000000000000000000200000000000000
02222000000000000000001000000000000000000000000000000000000000000000
02222000000000000000000100000000000000000000000000000000000000000000
00220000000000000000000012222000000000000000000000000000000000000000
00220000000000000000000010000000000000000000000000000000000000000000
00220000000000000000000001000000000000000000000000000000000000000000
00220000000000000000000000100000000000000000000000000000000000000000
00200000000000000000000000010000000000000000000000000000000000000000
00000000000000000000000000001222222222222222222220000000000000000000
00000000000000000000000000000010000000000000000000000000000000000000
00000000000000000000000000000021000000000000000000000000000000000000
00000000000000000000000000000022122200000000000000000000000000000000
00000000000000000000000000000020010000000000000000000000000000000000
00000000000000000000000000000020021000000000000000000000000000000000
00000000000000000000000000000022022100000000000000000000000000000000
00000000000000000000000000000020000122222000000000000000000000000000
00000000000000000000000000000020000010000000000000000000000000000000
00000000000000000000000000000020000001000000000000000000000000000000
00000000000000000000000000000020000000100000000000000000000000000000
00000000000000000000000000000020000000010000000000000000000000000000
00000000000000000000000000000020000000001000000000000000000000000000
00000000000000000000000000000020000000000122220000000000000000000000
00000000000000000000000000000020000000000010000000000000000000000000
00000000000000000000000000000020000000000021000000000000000000000000
00000000000000000000000000000020000000000201000000000000000000000000
00000000000000000000000000000020000000000200100000000000000000000000
00000000000000000000000000000020022002200000001200000000000000000000
00000000000000000000000000000020022002200000000100000000000000000000
00000000000000000000000000000020000000000000000010000000000000000000
00200020002000000000200000000002200000000000000000012222222222222222
02222020002000000000200200000002200000000000000000012222202222222222
00200020000000000000000000000000000000000000000000001220000000000000
00000000000000000000000000000000000000000000000000000100000000000000
00000000000000000000000000000000000000000000000000000210000000000000
00000000002000000000000000000000000000000000000000000201000000000000
00000000000000000002000000000000000000000000000000000120000000000000
02222000000000000002002000000000000000000000000000000200100000000000
00000000000000000000000000000000000000000000000000000010000000000000
00000000000000000000000000000000000000000000000000000200001000000000
00000000000000000000000000000000000000000000000000000200000120000000
00000000000000000000000000000000000000000000000000000200000010000000
00000000000000000000000000000000000000000000000000000200000001000000
```

Matrix B_{11}

```
144
170
12222220000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
01222000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

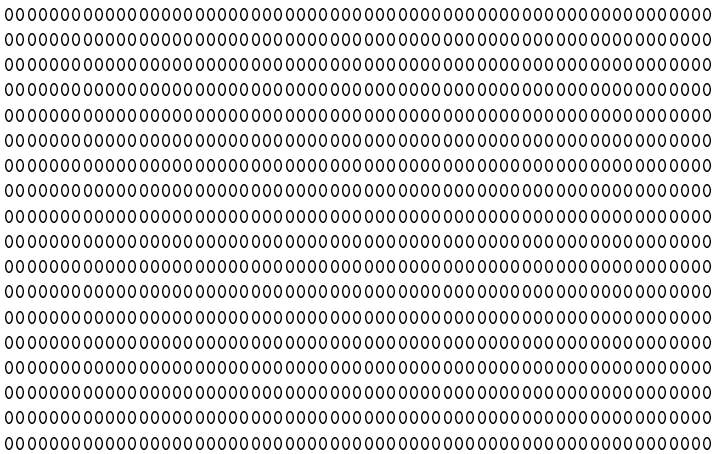
```
00100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00210000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00201000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000001220000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
22222221020000000000000000000000020000000000000000000000000000000000000000000000000000000000000000000000
00000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000001200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000102000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000001220000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000122000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000001222222220000020000000000000000000000000000000000000000000000000000000000000000000000
22222220200000000001222222220000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000002000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000010000000000200000000000000000000000000000000000000000000000000000000000000000000000
00200000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000122222222000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000012000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000012222200000000000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000012222200000000000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000122000000000000000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000122000000000000000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000
22222220002020000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000000
22222220000000000000000000000000000001000000000000000000000000000000000000000000000000000000000000000000
22222220002200000000000000000000000012222222022222222222222222222222022222
22222220002020000000000000000000000012222222022222222222222222222222022222
22222220002020000000000000000000000012222222022222222222222222222222022222
22222220002020000000000000000000000012222222022222222222222222222222022222
22222220002020000000000000000000000012222222022222222222222222222222022222
22222220002200000000000000000000000012222222022222222222222222222222022222
22222220002200000000000000000000000012222222022222222222222222222222022222
22222220000000000000000000000000000012200000000000000000000000000000000000
22222220000000000000000000000000000001000000000000000000000000000000000000
22222220000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000212000000000000000000000000000000000000
```

```
22222220000000000000000000000000000000000000000000000000100000000000000000000000000000000
22222220000000000000000000000000000000000000000000000012202000000000000000000000000000000
22222220000000000000000000000000000000000000000000000120220000000000000000000000000000000
22222220000000000000000000000000000000000000000000000010000000000000000000000000000000000
22222220000000000000000000000000000000000000000000000100000000000000000000000000000000000
22222220000000000000000000000000000000000000000000001000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000012220000000000000000000000000000000000
22222220000000000000000000000000000000000000000000000100000000000000000000000000000000000
22222220000000000000000000000000000000000000000000021000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000201000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000012000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000010000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000120000000000000000000000000000000000000
22222220000000000000000000000000000000000000000000001000000000000000000000000000000000000
22222220000000000000000000000000000000000000000001220000000000000000000000000000000000000
22222220000000000000000000000000000000000000000010000000000000000000000000000000000000000
22222220000000000000000000000000000000000000000100000000000000000000000000000000000000000
22222220000000000000000000000000000000000000001000000000000000000000000000000000000000000
22222220000000000000000000000000000000000000010000000000000000000000000000000000000000000
22222220000000000000000000000000000000000000122000000000000000000000000000000000000000000
22222220000000000000000000000000000000000001000000000000000000000000000000000000000000000
22222220000000000000000000000000000000000010000000000000000000000000000000000000000000000
22222220002200000000000000000000000000000000000000000000000000000000000000001222000000000
22222220002020000000000000000000000000000000000000000000000000000000000000001022200000000
22222220002020000000000000000000000000000000000000000000000000000000000000001222000000000
22222220002200000000000000000000000000000000000000000000000000000000000000001022200000000
22222220000000000000000000000000000000000000000000000000000000000000000000100000000000000
22222220002200000000000000000000000000000000000000000000000000000000000000010000000000000
22222220002020000000000000000000000000000000000000000000000000000000000000010000000000000
22222220000000000000000000000000000000000000000000000000000000000000000001000000000000000
22222220000000000000000000000000000000000000000000000000000000000000000010000000000000000
22222220000000000000000000000000000000000000000000000020000000000000000000000000000010000
22222220000000000000000000000000000000000000000000000200000000000000000000020000000001222
22222220000000000000000000000000000000000000000000000000000000000000000000000000000000100
22222220000000000000000000000000000000000000000000000000000000000000000000000000000000210
22222220000000000000000000000000000000000000000000000000000000000000000000000000000000201
```

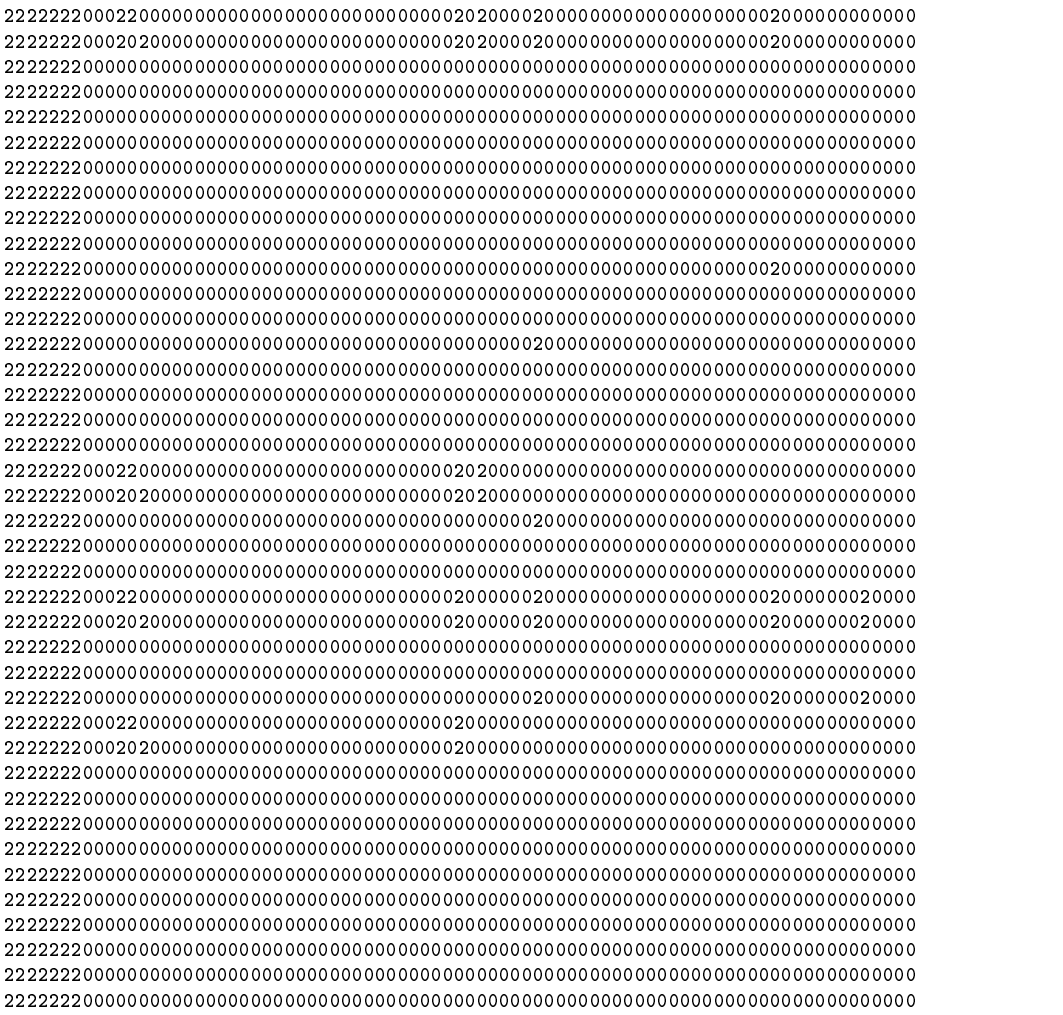
Matrix B_{12}

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

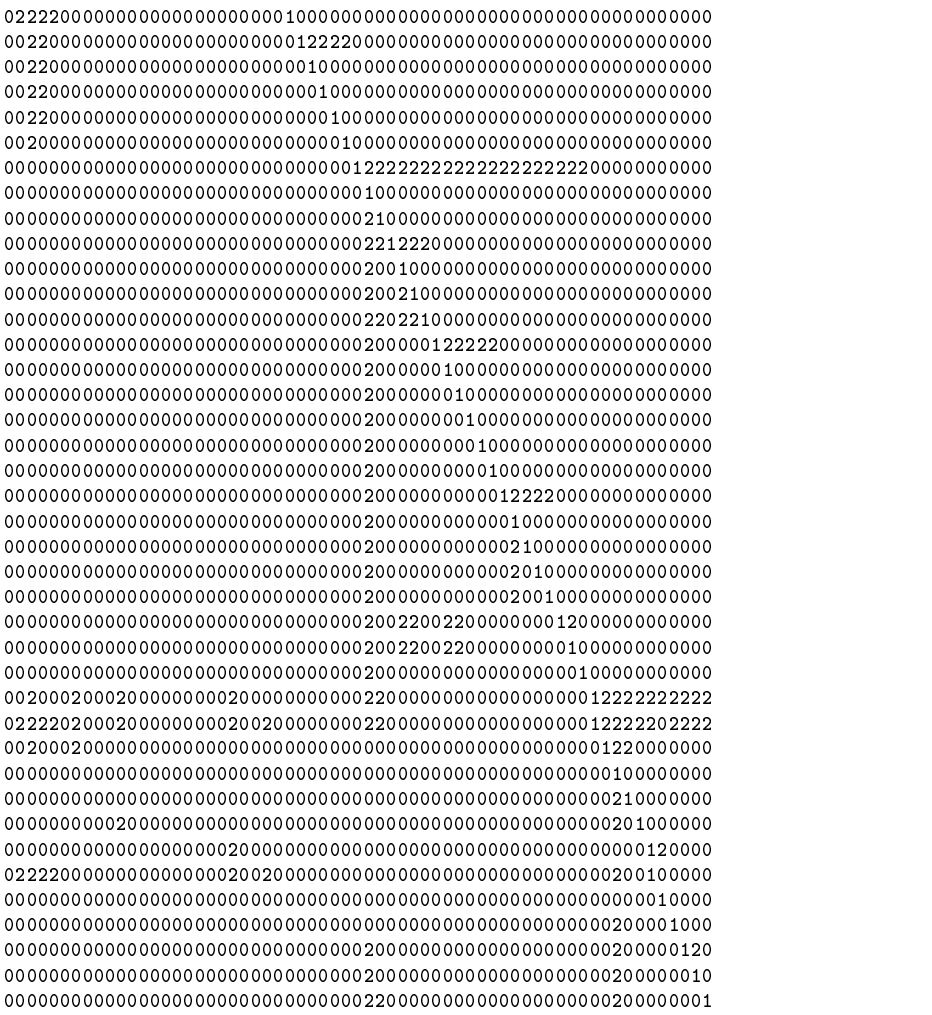
```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000002000000000000000000000000000000000000000000000000000222000000
00000000002000000000000000000000000000000000000000000000000000222000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000002000000000000000000000000000000000000000000000000000222000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000002000000200000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000002000000200000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000002000000200000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
```



Matrix B_{21}



```
22222220000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```



Anhang B

Aufgaben des HTML-Kurses

B.1 Aufgabenliste

Grundregeln

1. *Regeln* (Typ 1)

   Spüre in der folgenden HTML-Datei Schreib- und Denkfehler des Autors auf. Lösche für diese Zeilen die Markierungen.

   - `<index.html>`
   - `<HEAD>`
   - `<title>Irregular Verbs </title>`
   - `The verbs are taken from Oxford Dictionary.`
   - `</head>`
   - `<body text=#000000 bgcolor=#F9D2B0 link=#000099 vlink=#663366 alink=#FF0000>`
   - `<black font>`
   - `<b>Previous: Verbs starting with <i>a</i/b>`
   - `<u>Verbs starting with <i>b</i></u>`
   - `be was been`
   - `were`
   - `begin began begun`
   - `blow blew blown`
   - `build built built`
   - `<b>Next: Verbs starting with <i>c</b></i>`
   - `</black font>`
   - `</BODY>`
   - `</index.html>`

   Untersuche den Quellcode dahingehend, ob HTML-**Steuerbefehle**, unabhängig von ihrer Bedeutung, richtig geschrieben und kombiniert wurden. Stimmt das **Gründgerüst**? Hat der Programmierer beim **Editieren** falsche Annahmen über die Interpretierung durch den Browser getroffen?

2. *Steuerbefehle* (Typ 1)

   Welche HTML-Ausdrücke sind korrekt?
   (`<font>`, `<b>` und `<i>` sind Tags.)

   - `<b><font size=4 color=red>Achtung! Bis hierher und nicht weiter. </b></font>`
   - `<font color=red size=4> <b>Achtung! <i>Bis hierher und nicht weiter. </i></b></font>`
   - `<font size=5 color=black<b>>Achtung! Bis hierher und nicht nicht weiter.<</b>/font>`

- `<font size=5 warningcolor><b><i> Achtung!</i> Bis hierher und nicht weiter. </b></font>`
- `<font color=red><font size=4><b>Achtung!</b> Bis hierher und nicht weiter.</font size=4>`
- `<b><black font>Achtung! Bis hierher und nicht weiter.</font></b>`

Sieh dir jeden der HTML-Ausdrücke genau an: Sind **Attribute** und die **Verschachtelung** von Tags richtig eingesetzt worden?

3. *Tags* (Typ 1)

   Welche der folgenden Aussagen über Tags sind zutreffend?

   - Zu manipulierender Text steht im Gültigkeitsbereich zwischen einleitendem und abschließendem Tag.
   - Zwar hat jeder Befehl sein einleitendes und sein abschließendes Tag, jedoch kann letzteres oft weggelassen werden.
   - Befehle werden durch Tags eingeleitet. Das Ende markiert ein Semikolon ;
   - Die Tagnamen werden in spitze Klammern `<...>` eingefaßt, für abschließende Tags mit einem Schrägstrich vor dem Namen `</...>`

   Eine HTML-Datei enthält den Text der Hypertext-Seite, die man in einem Browser darstellen möchte. Mit den Kommandos der HTML kann der Text manipuliert werden, so daß er am Bildschirm z. B. farbig, als Hyperlink oder mit Grafiken versehen angezeigt wird.

   `<...>` *Gültigkeitsbereich* `</...>`

   - Allgemein bestehen Kommandos aus einem einleitenden und einem abschließenden „Tag“.
   - Diese umklammern den Gültigkeitsbereich, welcher den Text enthält, auf den das Kommando sich auswirken soll.
   - Das Tag ist der Name des Befehls (Groß- oder Kleinbuchstaben), eingefaßt in spitze Klammern `<...>` Im abschließenden Tag kommt davor ein Schrägstrich `</...>`
   - Ein paar Befehle, die nur zwischen Textabschnitten vorkommen, stehen alleine, d. h. ohne abschließendes Tag, da.

4. *Verschachtelung* (Typ 1)

   Welche der Ausdrücke mit Schachtelung sind korrekt?
   (`<h1>`, `<i>` und `<u>` sind Tags.)

   - `<u><i>Gesucht:</u> Diplom-Informatiker</i>`
   - `<h1<i>>Gesucht: Diplom-Informatiker <</i>/h1>`
   - `<h1><i>Gesucht:</i> Diplom-Informatiker</h1>`
   - `<h1><i><u>Gesucht: Diplom-Informatiker </u></i></h1>`

   Hilfreich ist die Kenntnis von **Tags**.

   Das ist neu:

   `<...>`*Text Ebene 1*
   `<...>`*Text Ebene 2*`</...>`
   Text Ebene 1`</...>`

   - HTML-Befehle können beliebig tief geschachtelt werden.
   - Die Tags auf tieferer Ebene stehen dann im Gültigkeitsbereich der Tags auf höherer Ebene.

- So können Teile von manipuliertem Text weiter verändert werden.
- Beispielsweise könnte man ein Wort in kursivem Text zusätzlich noch unterstreichen lassen.

5. *Attribute* (Typ 1)

Welche der Ausdrücke mit Attributen sind korrekt?
(`<font>` und `<table>` sind Tags.)

- `<table border>`
  `...</table>`
- `<table align=left>`
  `...</table>`
  - `<font blacksmallplain>`
    `...</font>`
  - `<color=white font>`
    `...</font>`
  - `<font size=2>`
    `...</font size=2>`
- `<font color=black size=2>`
  `...</font>`

Hilfreich ist die Kenntnis von **Tags**.

Das ist neu:

<*Tagname Attribut1 Attribut2*>
...</*Tagname*>

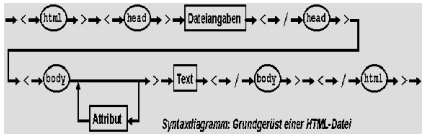
- Bei vielen Tags kann man die Grundbedeutung um zusätzliche Effekte erweitern.
- Man schreibt dazu ein Attribut bzw. mehrere Attribute mit ins einleitende Tag, siehe oben.
- Zu jedem Tag gibt es eine begrenzte Menge an verwendbaren Attributen.
- Manche Attribute sind für eine Vielzahl von Tags gültig, z. B. Ausrichtung oder Hintergrundfarbe.
- Ein Attribut kann ein einzelnes Wort sein, z. B. `border` für Tabellen mit Linien.
- Ein Attribut kann eine Wertzuweisung sein: `align=left` steht für linksbündige, `align=right` für rechtsbündige Ausrichtung.

6. *Grundgeruest* (Typ 5)

Ergänze den Lückentext, so daß das Grundgerüst einer HTML-Datei entsteht.

<html>
<head>
Kopf mit Dateiangaben
</head>
<body *Attribute*>
Rumpf mit dem Dokumenttext
</body>
</html>

Die Datei, welche den HTML-Code erhält, ist eine ganz normale Textdatei. Damit sie von Browsern interpretiert werden kann, muß sie nach einem vorgegebenen Schema gegliedert sein.



Syntaxdiagramm: Grundgerüst einer HTML-Datei

- Den Rahmen der Datei bilden die Zeilen `<html>` am Anfang sowie `</html>` am Ende.
- Dazwischen stehen zwei Sektionen: der Kopf und der Rumpf.
- Der Kopfinhalt ist in die Zeilen `<head>` und `</head>` eingefaßt. Im Kopf stehen Angaben zur Datei, z. B. der Titel des Dokuments.
- Der Rumpfinhalt ist in die Zeilen `<body>` und `</body>` eingefaßt. Im Rumpf steht der eigentliche Text und die HTML-Kommandos.

- Die Rumpfdeklaration kann zusätzlich um Angaben über den Dokumenttext ergänzt werden: `<body` *Attribute*`>` Beispielsweise kann eine Hintergrundfarbe definiert werden.

7. *Editieren* (Typ 1)

   Was läßt sich über das Editieren der HTML-Datei sagen?

   - Es macht immer Sinn, sich um eine übersichtliche Gestaltung des Quellcodes zu bemühen.
   - What you see is what you get: So wie Text in der Datei formatiert ist, erscheint er im Browserfenster.
   - Längere Zeilen werden automatisch am Rand umgebrochen.
   - Zur Textausgabe benutzt man das `<print>`-Tag.
   - Mit der Tabulatortaste eingerückter Text wird im Browser trotzdem an den Zeilenanfang gestellt.
   - Zeilenumbrüche müssen explizit per HTML-Befehl angezeigt werden. Ebenso Absätze. Abstände zwischen zwei Wörtern werden, wenn nicht extra gekennzeichnet, auf ein Leerzeichen verkleinert.

   Eine HTML-Datei enthält im wesentlichen den Text, der im Browser angezeigt werden soll.

   - Einfachen Text kann man direkt in die Datei schreiben, ohne irgendwelche Befehle drumherum.
   - Allerdings wird der Text vom Webbrowser anders formatiert, als ihn der Programmierer im Editor vor sich sieht.
   - Zeilenumbrüche können für jeden Betrachter unterschiedlich gesetzt sein – abhängig von der Auflösung, Fensterbreite, individuellen Programmeinstellungen usw.
   - Absätze, zusätzliche Leerzeilen und größere Abstände als ein Leerzeichen müssen gesondert im Quellcode ausgewiesen werden.
   - Tabulatoren in der Datei haben keine Auswirkungen.
   - Natürlich ist es trotzdem ratsam, den eigenen Quelltext der Übersicht halber ordentlich zu formatieren.

8. *Sonderzeichen* (Typ 2)

   Gib zu jeder der mysteriösen Zeichenfolgen an, was sie codiert. Handelt es sich um die Umschreibung eines Umlautes (incl. „ß“), das Maskieren eines HTML-internen Zeichens oder die Codierung eines besonderen Zeichens?

   - `ß` — Umlaut / Maskierung / Spezialzeichen / weiß nicht
   - `°` — Umlaut / Maskierung / Spezialzeichen / weiß nicht
   - `&` — Umlaut / Maskierung / Spezialzeichen / weiß nicht
   - `<` — Umlaut / Maskierung / Spezialzeichen / weiß nicht
   - `Ä` — Umlaut / Maskierung / Spezialzeichen / weiß nicht

- `"` | Umlaut / Maskierung / Spezialzeichen / weiß nicht |
- `iso-8859-1` enthält besonders | Umlaute / Maskierungen / Spezialzeichen / weiß nicht |

(Von den zwei Umlaut-Codes einerseits und dem Zeichensatz andererseits genügt es, eines zu wissen und bei der Aternative mit „weiß nicht“ zu antworten.)

Hilfreich ist die Kenntnis der Codes für das **Maskieren** und für **Umlaute** bzw. das Wissen um die Festlegung von **Defaultsprachen** und des verwendeten Zeichensatzes.
„Spezialzeichen“ sind z. B.

- `©` für das Copyright
- `°` für Grad (“degrees”)
- `£` für britische Pfund

9. *Umlaute* (Typ 5)

Herr Möllemann ist furchtbar stolz auf seine selbsterstellte Homepage.

Ich heiße Jürgen Möllemann. Über mich gäbe es mächtig viel zu erzählen. Wenn ich doch bloß wüßte, wo ich anfangen soll...

Überglücklich sendet er die Adresse an Kollegen in der ganzen Welt. Bei Mr Clinton ruft der Text nur Irritation hervor, denn sein Browser zeigt ein ziemliches Kauderwelsch.

Schaffe dem Abhilfe, indem du die drei Sätze im folgenden Lückentext mit international übersetzbaren Codes anstatt der Umlaute (und „ß“) wiedergibst.

```
Ich hei&szlig;e J&uuml;rgen
M&ouml;llemann. &Uuml;ber
mich g&auml;be es
m&auml;chtig viel zu
erz&auml;hlen. Wenn ich doch
blo&szlig; w&uuml;&szlig;te,
wo ich anfangen soll...
```

Der vom Betriebssystem verwendete Zeichensatz ist von der Landessprache eines Users abhängig. Deutsche Umlaute im HTML-Text werden deshalb nicht überall auf der Welt korrekt dargestellt. Um sicherzugehen, daß die selbstgeschriebenen Texte international lesbar sind, ersetzt man in HTML Umlaute durch von Browsern übersetzbare Umschreibungen – ähnlich dem **Maskieren** von HTML-internen Zeichen.

- `ä` steht für ä (a-Umlaut)
- `ö` steht für ö (o-Umlaut)
- `ü` steht für ü (u-Umlaut)
- `Ä` steht für Ä (A-Umlaut)
- `Ö` steht für Ö (O-Umlaut)
- `Ü` steht für Ü (U-Umlaut)
- `ß` steht für ß („sz“)

10. *Maskieren* (Typ 5)

Setze folgende Zeile in HTML um. Maskiere alle darin auftretenden Sonderzeichen.

<<<“Cremig & Fruchtig”>>>

Vervollständige dazu den Lückentext.

```
&lt;&lt;&lt;&quot;Cremig &amp;
Fruchtig&quot;&gt;&gt;&gt;
```

Zeichen, die selbst schon zur Sprache HTML gehören, kann man an manchen Stellen nicht einfach so im Text verwenden. Um Konflikte zu vermeiden, muß man sie durch Umschreibungen ersetzen – „maskieren“. Die Codes sehen ähnlich aus wie die der **Umlaute**.

- `&` steht für & ("ampersand", „Kaufmanns-Und")
- `"` steht für " ("quote", „Zitat")
- `<` steht für < ("lesser than", „kleiner als")
- `>` steht für > ("greater than", „größer als")

11. *Farbdefinition* (Typ 4)

Such dir zuerst eine von vier Farben aus.

black (0,0,0)	navy (0,0,128)
green (0,128,0)	teal (0,128,128)

Trage nun deine Farbe in HTML-Notation ein (als Attributwert ohne Anführungszeichen).

`color=` _____

Codiere die Farbe entweder **hexadezimal** oder verwende die Schreibweise mit **Farbnamen**.

12. *hexadezimal* (Typ 3)

Zuerst, überlege dir eine Farbe und stelle ihren Rot-, Grün- und Blauanteil ein.

0 255
Rot: | | | | | | | | | | | | | | | |

0 255
Grün: | | | | | | | | | | | | | | | |

0 255
Blau: | | | | | | | | | | | | | | | |

Gib die Farbe nun in hexadezimaler HTML-Notation an (als Attributwert ohne Anführungszeichen). Dazu ist es notwendig, die eingestellten dezimalen Farbanteile umzurechnen.

`color=` _ _ _ _ _ _

In vielen Kommandos kann als Ergänzung eine Farbe angegeben werden, z. B. um farbige Schrift darzustellen. Über verschiedengroße Anteile an Rot, Grün und Blau kann man alle denkbaren Farben zusammenmischen.

- Der Ausdruck für eine Farbmischung beginnt mit einem Rautezeichen.
- Angehängt werden die zweistelligen Werte für Rot, Grün und Blau: `#RRGGBB`
- Die hexadezimalen Farbanteile reichen von minimal `00` (dezimal 0) bis maximal `FF` (dezimal 255).
- Man erhält die linke Ziffer eines Hexadezimalwertes, indem man die Dezimalzahl durch 16 teilt. Die rechte Stelle ist der Rest der Division.
- Dabei entstehen Ziffern über 9: `A` für 10, `B` für 11, `C` für 12, `D` für 13, `E` für 14, `F` für 15.

13. *Farbnamen* (Typ 4)

Wähle zuerst eine von sechs Farben aus.

yellow	olive	teal
purple	navy	blue

Gib nun den HTML-Namen deiner Farbe an.

`color=` _____

Von 16 Grundfarben versteht jeder Browser die Namen.

- `black` (schwarz)
  `gray` (grau)
  `silver` (silber/hellgrau)
  `white` (weiß)
- `maroon` (dunkelrot)
  `red` (rot)
  `fuchsia` (pink)
  `purple` (violett)
- `lime` (hellgrün)
  `green` (grün)
  `olive` (oliv)
  `yellow` (gelb)

- `navy` (dunkelblau)
  `blue` (mittelblau)
  `aqua` (hellblau)
  `teal` (türkis)

"Netscape" und dem "Internet Explorer" sind noch viele weitere Farbnamen bekannt. Alternativ kann man sich selbst Farben definieren (in Hexadezimalschreibweise).

14. *KommentareCredits* (Typ 6)

Edgar Alan Poetry hat ein wundervolles Liebesgedicht für seine Freundin geschrieben. Ab Mai 2001 wird er es der ganzen Welt auf seiner Internetseite präsentieren und per E-Mail ("ed@poetry.co.uk") Meinungen von Lesern einholen. Besorgt ist er darüber, daß jemand seine Ergüsse klauen oder darin herumpfuschen könnte. Edgar fügt also seiner Seite Credit-Informationen mit Copyright-Angabe hinzu. Er gibt auch an, daß er ein Hintergrundbild von "Web Art" verwendet, wobei er die Firma noch um Erlaubnis bitten muß.
Welches Format für die Credits würdest du Edgar vorschlagen?

Text (c) MM/JJJJ by Name Nur Weitergabe des Originals erlaubt! Hintergrundbild (c) by Firma Mail-Adresse: Mail-Adresse als Mail-Link

Ein Gedicht geschrieben von Name als Mail-Link. Das Copyright liegt bei Name. Der Hintergrund stammt von Firma. Letztes Update im Monatsname JJJJ.

„Ich nehme das Format. . . "

oben unten

Gib nun die relevanten Daten aus der Aufgabenstellung im von dir gewählten Format ein – und zwar so wie sie in einer HTML-Datei stehen würden, d. h. mit E-Mail-Referenz.
Füge vor die Angaben zum Hintergrundbild eine Kommentarzeile mit dem Inhalt „Web Art mailen" ein.
`<br>` markiert das Ende einer Textzeile.

```
_____ <br>
_____ <br>
<!--Web Art mailen-->
_____ <br>
_____
```

Zum Lösen der Aufgabe sollte man Wissen zum einen über **Kommentare** (`<!--...-->`) und zum anderen über **Credits** besitzen.

15. *Kommentare* (Typ 1)

In welchen Fällen sind Kommentare richtig benutzt werden?

- ```
  <!--erst schwarze Schrift
  dann rote Schrift//-->
  <font color=black>
  Bericht 1999</font>
  <font color=red>
  Bericht 2000</font>
  ```
- ```
  <!--schwarze Schrift-->
  <font color=black>
  Bericht 1999</font>
  <!--rote Schrift-->
  <font color=red>
  Bericht 2000</font>
  ```
- ```
  <font color=<!--rot-->
  red>Bericht 2000</font>
  ```
- ```
  <a href="r99.htm">
  Bericht 1999</a><br>
  <a href="r00.htm">
  Bericht 2000</a><br>
  <!--<a href="r01.htm">-->
  Bericht 2001<!--</a>-->
  ```

Wie in den meisten Sprachen können in HTML Teile des Quellcodes mit

Kommentaren versehen werden, die der Browser dann überliest.

- Man schreibt eine Kommentarzeile in `<!--...-->`
- Man schreibt einen Kommentarblock, der über mehrere Zeilen geht, in `<!--...//-->`
- Kommentare können sich auch auf Teile von Zeilen beschränken und zwischen HTML-Kommandos stehen bzw. diese umschließen und damit „auskommentieren".
- Kommentare können nicht innerhalb von Kommandos verwendet werden.

16. *Credits* (Typ 6)

Selbst Boris Becker ist mittlerweile „drin" im Internet. Du erhältst von ihm den Auftrag, in seinem Namen eine Website mit lustigen Tennisanekdoten zu erstellen. Leserbriefe sollen direkt an "becker@aol.com" gehen. Am 19. Juni 2001 bist du endlich damit fertig; es fehlen nur noch die Credits am Ende der Seite.
Wähle zuerst ein passendes Format dafür.

Autor: Name als Mail-Link Erstelldatum: TT.MM.JJ Hinweis: Weitergabe nur in unmodifizierter Form!

Text von Name. Das Copyright liegt beim Autor. Fragen und Anregungen bitte an Mail-Adresse als Mail-Link. Letztes Update am TT. Monatsname JJJJ.

„Ich nehme das Format..."

oben unten

Gib nun die relevanten Daten aus der Aufgabenstellung im von dir gewählten Format ein – und zwar so wie sie in einer HTML-Datei stehen würden, d. h. mit E-Mail-Referenz. `<br>` markiert das Ende einer Zeile.

_____ `<br>`
_____ `<br>`
_____ `<br>`
_____

Für das Eingeben der Lösung solltest du dich mit **Mailadressen** auskennen (`<a href="mailto:...">`).
Sinnvolle Informationen am Ende einer Hypertextseite:

- Wer ist der verantwortliche Autor?
- Wer hält die Urherberechte an den Inhalten und Komponenten? Einschränkungen für Benutzung und Weitergabe.
- Möglichkeiten, dem Autor oder Webmaster per E-Mail Rückmeldungen zukommen zu lassen – z. B. durch Unterlegen des Autorennamens oder der Mailadresse mit einem entsprechenden Link.
- Wann fand die letztere Aktualisierung statt?

17. *Dateikonventionen* (Typ 1)

Oh je! Du wurdest zu Franziskas virtueller Geburtstagsparty eingeladen, hast aber die genaue WWW-Adresse vergessen. Angenommen, die richtige URL war

`http://www.parties.de/franziska/begruessung.html`

und der Parties-Provider hätte sich auf `begruessung.html` als Defaultdatei festgelegt. Über welche Eingaben würdest du Franziska auch erreichen?

- ○ `http://www.parties.de/`

`franziska/Begruessung.html`

- `http://www.parties.de/franziska/begruessung.jpg`
- `http://www.parties.de/franziska/begrüßung.html`
- `http://www.parties.de/franziska/begruess.htm`

- `http://www.parties.de/franziska/`

Zur Lösung hilft die Kenntnis der Regeln für **Dateinamen** sowie das Wissen um die Funktion der **Defaultdatei**.

18. *Dateinamen* (Typ 1)

Welche der folgenden Aussagen über Dateinamen sind zutreffend?

- Namen bestehen generell aus 8 Zeichen + 3 für die Extension.
- `index.html` hat die gleiche Bedeutung wie `INDEX.HTML`

- Die Zeichen `*`, `?`, `ä`, `ö`, `ü`, `ß` oder das Leerzeichen gehören nicht in Dateinamen.
- Man kann die Hauptdatei auf `.html` oder `.htm` enden lassen. Die verwendeten Grafiken haben je nach Grafikformat die Extension `.gif` oder `.jpg`

Eine Website besteht generell aus einer Vielzahl von Dateien. Damit deren Zusammenspiel auf jedem Rechner funktioniert, sind einige Regeln zum Benennen der Dateien einzuhalten.

- Auf Leerzeichen, Umlaute und Sonderzeichen ist zu verzichten – mit Ausnahme des nützlichen Unterstreichungszeichens _
- Groß-/Kleinschreibung in Dateinamen spielt im Internet eine Rolle!

- Namen sollten, wenn möglich, nur einen Punkt und damit nur eine Extension enthalten. Aus Rücksicht auf die älteren PC-Betriebssysteme kann man sich auf Namen mit 8 + 3 Zeichen beschränken.
- Für HTML-Quelldateien ist die Extension `.html` oder `.htm` zu verwenden. Grafiken enden abhängig von ihrem Format auf `.gif` oder `.jpg`

19. *Defaultdatei* (Typ 5)

Die Homepage des Studenten mit der Usernummer y0003231 an der Technischen Uni in Braunschweig hat die Adresse
`http://www.tu-bs.de/~y0003231/index.html`
`index.html` wurde vom Rechenzentrum (dem Provider) als Name der Defaultdatei bestimmt.
Was kann man also im Browser abkürzend eingeben, um den Student im WWW zu besuchen?
`http://www.tu-bs.de/~y0003231`
(alternativ mit / am Ende)

Folgendes ist neu:

- Ein Provider kann einen Dateinamen vorgeben, der die Startseiten seiner Websites bezeichnet, z. B. `index.html`
- Die entsprechende Datei wird dann automatisch in den Browser geladen, wenn man als Besucher den Pfad zu der Website einträgt: Man kann den Defaultdateinamen weglassen.

20. *Organisation* (Typ 1)

Was läßt sich zur Organisation eines Webauftritts sagen?

- Man erstellt eine Reihe von einzelnen HTML-Dateien, die jede

für sich eine einzelne Seite bzw. einen Gliederungsabschnitt beschreiben.

- Für die Leser ist es angenehm, wenn möglichst viele Informationen auf der Hauptseite stehen.

- Bei gemäß der Gliederung verketteten Seiten werden Hyperlinks nicht nur „vorwärts“, sondern auch „zurück“ eingefügt.

- Dateien, die über Querverweise verbunden sind, kopiert man i.a. ins selbe Verzeichnis.

Wie kann man eine Hypertext-Seite vom Entwurf bis zum Betrieb ordentlich organisieren?

- Erstmal gliedert man die Informationen des Angebotes übersichtlich in viele kleine Einheiten – bei Bedarf über mehrere Strukturieungsebenen.

- Die Einheiten werden sinnvoll über Links vernetzt. Von der Hauptseite aus führen Verzweigungen zu Abschnitten, Unterabschnitten und wieder zurück. Zudem verlinken Querverweise (Unter-) Abschnitte, die in einer Beziehung zueinander stehen.

- Jede Einheit wird eine zu programmierende HTML-Datei.

- Die fertigen Dateien kann man anhand der Hypertext-Gliederung in Verzeichnisse einsortieren. Die Verzeichnisstrukturen müssen in den Pfadnamen der programmierten Links berücksichtigt sein.

21. *Stil* (Typ 2)

Es kommt jetzt eine lange Liste von HTML-Ausdrücken, fertigem Hypertext und Entwurfspraktiken. Urteile über jeden Punkt, ob die Programmierung, das Design bzw. das Vorgehen guter Stil oder schlechter Stil oder schlichtweg falsch ist.

- `<li>Eintrag` gut / schlecht / falsch

- Stanley Kubrick verfilmte Arthur C. Clarkes Roman „2001“ (Inhaltsangabe). gut / schlecht / falsch

- `<img src=“paris.gif”>` gut / schlecht / falsch

- Meta-Angabe des `iso-8895-4` oder Codierung der Umlaute, z. B. `J&uauml;rgen` gut / schlecht / falsch

- möglichst viele Spezialfeatures von Netscape 4.x nutzen gut / schlecht / falsch

- Reduzieren der Farbenzahl in Grafiken gut / schlecht / falsch

- Programmieren für die Auflösung 1152 × 864 gut / schlecht / falsch

- `<code>output:=input MOD 7 </code>` gut / schlecht / falsch

Jeder der Punkte thematisiert einen Aspekt des guten (bzw. schlechten) HTML-Stils:

- **Endtags**
- **Linktexte**

- **Größenangabe**
- **Zeichen**
- **Browser**
- **Grafikeinsatz**
- **Auflösung**
- **Zweckentfremden**

22. *Browser* (Typ 1)

Was ist guter HTML-Stil?

- sich gleich zu Beginn auf einen Browser festzulegen; man läuft sonst Gefahr, für die verschiedensten Plattformen durcheinander zu programmieren
- die wesentlichen Informationen auch mit älteren Browsern anzeigbar zu halten
- unterschiedliche Betriebssysteme zu bedienen, z. B. Schriftartnamen sowohl von UNIX als auch von Windows anzugeben

Es ist guter HTML-Stil, Code zu fabrizieren, der von vielen unterschiedlichen Browsern verarbeitet und damit von Menschen in unterschiedlichen Arbeitssituationen gelesen werden kann. Am besten testet man erstellte Seiten...

- ... in den beiden marktbeherrschenden Produkten von Netscape und Microsoft. Leider sind diese an so mancher Stelle inkompatibel.
- ... auch mit älteren Versionen der Browser. Zumindest wichtige Informationen sollten mit Netscape 3.x lesbar sein.
- ... auf mehreren Betriebssystemen. Es gibt den Browser von Netscape zwar unter Windows und unter UNIX, jedoch spielen in die Benutzung systemspezifische Eigenheiten wie etwa unterschiedliche Schriftartbezeichnungen hinein.

Praktisch alle Sprachelemente dieses Kurses laufen im "Netscape Navigator" und im "Internet Explorer" in der jeweils aktuellen Version.

23. *zweckentfremden* (Typ 1)

Welche Praktik ist guter HTML-Stil?

- kursive Schrift mit dem „Zitieren"-Tag zu umschreiben
- Seitenränder über eine schmale Tabelle zu realisieren
- große, fette Schrift mit einem einzigen Befehl, der Deklaration als Überschrift, zu erhalten

Falls es dir schon ein Begriff ist: Dieser Punkt bezieht sich vor allem auf **logische Textauszeichnungen**.
Es ist guter HTML-Stil, Steuerbefehle ausschließlich dafür zu benutzen, wozu sie gedacht sind. Ein zweckentfremdetes Kommando mag im eigenen Browser den gewünschten Effekt erzielen; ein anderer könnte es dagegen abweichend umsetzen. Beispiele:

- `<code>` markiert Programmiersprachenausdrücke. Es ist nicht dafür vorgesehen, um beliebige Texte in Schreibmaschinenschrift darzustellen.
- Das logische Tag mit dem Namen `<blockquote>` beschreibt Zitatblöcke, aber nicht generelle Einrückungen.
- Andererseits ist es völlig legitim, Farbflächen über eine Tabelle mit farbigen Hintergründen zu erzeugen. Eine Tabelle läßt keinen Interpretationsspielraum zu.

24. *Zeichen* (Typ 5)

Wenn du die Linkliste zu deiner neuen Homepage erstellst, darf natürlich deine finnische E-Mail-Freundin

Päivi Väinämöinen

nicht fehlen. Allerdings machst du dir berechtigte Sorgen, daß nicht jeder Browser auf der Welt ihren Namen vernünftig anzeigen wird. Entweder du baust explizit einen Zeichensatz mit Umlauten in deine Website ein (und läßt dann den Namen, wie er ist), oder du codierst die Umlaute.

Meta-Angabe des Zeichensatzes für Westeuropa: `<meta http-equiv= “content-type” content= “text/html;charset= iso-8859-1”>`

Päivis Name codiert: `Päivi Väinämöinen`

Zur Lösung der Aufgabe hilft Kenntnis von entweder **Defaultsprachen** und Zeichensätzen oder aber der Umschreibung von **Umlauten**.
Es ist guter HTML-Stil, dafür zu sorgen, daß Umlaute und andere Spezialzeichen jedem Browser richtig übermittelt werden.

25. *Endtags* (Typ 1)

Welche HTML-Ausdrücke sind guter Programmierstil?

- `<li>Listenelement</li>`
- `Zeile1<br>Zeile2`
- `<p><b>Textabsatz</b>`

Zum Lösen der Aufgabe solltest du dich mit **Tags** auskennen.
Es geht um den Sinn einleitender und abschließender Tags.

- Ungeachtet der Tatsache, daß einige Befehle (`<br>` oder `<hr>`) gar kein Endtag besitzen, kann auch bei normalen Tags (`<p>` oder `<li>`) der Abschluß in den Fällen weggelassen werden, wo die eindeutige Interpretierbarkeit gewährleistet bleibt. Guter HTML-Stil ist das jedoch nicht mehr – eher handelt es sich um schludrige Programmierung.
- Selbst wenn das Ergebnis mit wegelassenen Endtags vollkommen richtig aussieht – wer garantiert, daß nach der nächsten Änderung dies noch immer so ist, wenn sich u. U. Gültigkeitsbereiche verschoben haben?
- Selbst wenn das Ergebnis im eigenen Browser richtig aussieht – wer garantiert, daß es das in anderen Browsern, die abweichende Algorithmen zur Analyse verwenden, ebenfalls tut?

26. *Linktexte* (Typ 1)

Guter HTML-Stil: Wo ist Text für die Leser sinnvoll mit Links unterlegt worden?

- **News**: Hier kann man die letzten Neuigkeiten nachlesen.
- **News**: Hier kann man die letzten Neuigkeiten nachlesen.
- **News**: Hier kann man die letzten Neuigkeiten nachlesen.

Es ist guter Stil, Hypertext, besonders die Verweise zu anderen Seiten und Textstellen, angenehm lesbar zu gestalten.

- Links in Text treten nicht übertrieben häufig auf.
- Innerhalb eines Absatzes gibt es keine unnötigen Wiederholungen ein und desselben Links.
- Das unterlegte Textstück gibt klare Auskünfte darüber, was sich hinter dem Link verbirgt.

27. *Grafikeinsatz* (Typ 1)

Zum guten HTML-Stil: Auf welcher Seite sind Grafiken vernünftig eingesetzt worden?

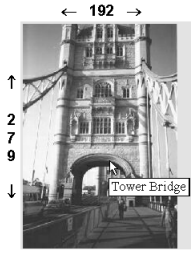
-
  Hallo!
  Willkommen auf
  meiner Homepage!
  Ich hoffe, es
- Bienvenu
  Välkommen
- Nouvelles
  Nyheterna
- Diskussionsforum
- Zeitungsartikel vom letzten Sonntag:

  Band's planning application comes fou
  Request t
  a little tir

Grafikeinsatz in Internet-Seiten will wohlüberlegt sein.

- Es ist einerseits angenehm für die Leser, wenn ein Text durch Bilder aufgelockert ist. Andererseits verursachen die Grafiken lange Ladezeiten während des Browsens einer Seite.
- Am komfortabelsten sind also kleine Grafikdateien. Grafiken brauchen ja nicht immer riesige Ausmaße anzunehmen oder Millionen von Farben zu enthalten. Effektiv und gleichzeitig effizient sind kleine Icons, grafische Trennlinien u. ä.
- Übermäßig große Bilder bringt man vorzugsweise auf eigenen Seiten unter.

28. *Groessenangabe* (Typ 5)



london.jpg

Binde die Grafikdatei in ein HTML-Dokument ein. Ergänze die Attribute für den alternativen Text, die Breite und die Höhe (in dieser Reihenfolge), damit das Laden der HTML-Datei für den Benutzer angenehmer wird.

```
<img src="london.jpg"
alt="Tower Bridge"
width=192 height=279>
```

Diese Aufgabe knüpft an Wissen zum Thema Grafiken an, nämlich den **Alternativtext** (das Attribut `alt`) sowie **Grafikbreite und -höhe** (Attribute `width`, `height`).
Es ist guter HTML-Stil, Grafiken die Angabe über ihre Größe sowie eine alternative Beschreibung hinzuzufügen.

- Die Größenangaben verkürzen die Wartezeit während des Seitenaufbauens erheblich, weil so der Browser zuerst den Text laden und anstelle der Grafiken ein freies Rechteck lassen kann. Die Kästchen werden nacheinander aufgefüllt, während der Besucher schon am Lesen ist.
- Der Alternativtext gibt schon Auskunft über das Bild, wenn es noch nicht geladen ist. Und auch danach kann man daran Zusatzinformationen als "Tooltip" ablesen.

29. *Aufloesung* (Typ 1)

Guter HTML-Stil: Was führt für User mit besonders hoher oder niedriger

Bildschirmauflösung nicht zu Problemen mit der Darstellung?

- eine Tabelle der Breite 75% des Bildschirmfensters
  - eine 1000 Pixel breite Hintergrundgrafik
  - drei horizontal aufgereihte Bilder der Größen 360 × 360

Es ist guter HTML-Stil, Hypertext-Seiten auflösungsunabhängig zu gestalten.

- Der verfügbare Platz auf dem Bildschirm eines Users ist von der eingestellten Auflösung abhängig.
- Man sollte also seine Seite nicht pixelgenau planen, sondern Höhen und Breiten nach Möglichkeit prozentual angeben:
  `width=80%` statt `width=820`
- Wenn es nicht anders geht, beispielsweise in Verbindung mit Grafiken, die immer eine Größe in Pixeln besitzen, dann sollte man testen, ob die Seite in niedrigen Standardauflösungen wie 640 × 480 oder zumindest 800 × 600 gut aussieht.

Dateiangaben

30. *Dateiangaben* (Typ 5)

Ein Webmaster hat zu einem besonders günstigen Preis den Domain-Namen `www.sibyllennetz.de` erstehen können. Leider erlaubt der Anbieter nur die Plazierung einer einzigen kurzen Seite auf dessen Server. Der Webdesigner beschließt, dort ein Dokument mit kurzer Einführung abzulegen, von dem die Besucher automatisch zur Original-Adresse der Seite weitergeleitet werden. Diese ist bereits bekannt und durch Suchmaschinen erfaßt, was auch mit der neuen Adresse geschehen soll. Eine Suchanfrage ergibt:

1. **Das Sibyllennetz**
Website über die Bücher von SF-Autorin Joan D. Vinge
URL: www.tu-bs.de/~y0003231/vinge/vinge.html - Last modified on: 23-Jun-2000

Füge in die Kopfsektion der neuen Datei folgende Angaben ein.

1. Dokumenttitel
2. Inhaltsbeschreibung
3. Erstelldatum des Originals
4. Verwendung des Zeichensatzes mit deutschen Umlauten
5. Weiterleitung in 6 Sekunden

```
<head>
  <title>Das Sibyllennetz
  </title>
  <meta name="description"
  content="Website über die
  Bücher von SF-Autorin Joan
  D. Vinge">
  <meta name="date"
  content="2000-06-23">
  <meta http-equiv=
  "content-type" content=
  "text/html;charset=
  iso-8859-1">
  <meta http-equiv="refresh"
  content="6;URL=
  http://www.tu-bs.de/
  ~y0003231/vinge/vinge.html"
  >
</head>
```

Die gesuchten Deklarationen sind zuerst der **Titel** `<title>...</title>` und darauf folgend vier **Meta-Angaben** `<meta ...>`

31. *Titel* (Typ 5)

Eine Hypertext-Seite soll den Titel "Project Bluebook" tragen. Trage die Titelangabe in HTML-Syntax an der richtigen Stelle ein.

```
<html>
<head>
  <title>Project Bluebook
```

```
  </title>
</head>
<body>
  ____________________
</body>
</html>
```

Zum Lösen der Aufgabe sollten dir die grundlegenden **Regeln** zur Syntax von Tags und zum Aufbau von HTML-Dateien bekannt sein.
Neu ist folgendes:

- Ein Hypertext-Dokument trägt einen Titel.
- Dieser wird u. a. in der Titelleiste des Browserfensters, Suchmaschinen-Ergebnissen und in der Lesezeichen-/Favoritenliste angezeigt.
- Man definiert den Titel in der Kopfsektion der HTML-Datei.
- Der Titelname steht zwischen einleitendem und abschließendem `<title>`-Tag.

32. *Metaangaben* (Typ 1)

Welche Meta-Definitionen sind korrekt?

- ```
  <head>
    <meta name="author"
    content="Diana Scaly">
    <meta name=
    "description" content=
    "Details on the
    Bluebook Project">
    <meta name="keywords"
    lang="en-us" content=
    "UFO, sighting, aliens"
    >
    <meta name="date"
    content="1998-08-30">
  </head>
  ```
- ```
  <body>
    <meta name="robots"
    content="nofollow">
  </body>
  ```
- ```
  <head>
    <meta http-equiv=
    "content-type" content=
    "charset=iso-8859-1">
  </head>
  ```
- ```
  <head>
    <meta http-equiv=
    "expires" content="0">
  </head>
  ```
- ```
  <head>
    <meta http-equiv=
    "refresh" content="3;
    URL=fbi.gov/bluebook/
    welcome.html">
  </head>
  ```

In jedem Frage wird geprüft, ob du die entsprechende Meta-Angabe beherrschst:

- **Beschreibung**
  mit **Stichwortsprache**
- **Auslesen**
- **Defaultsprachen**
- **Originaladresse**
- **Weiterleitung**

33. *Beschreibung* (Typ 5)

UFOs über Offenbach! In der Nacht zum 3.12.2000 hat Claudia Schmidt bei einem nächtlichen Stadtrundgang eine fliegende Untertasse beobachtet. Zu Hause angekommen stellt sie sofort eine detaillierte Beschreibung der Vorgänge ins Internet – natürlich mit genauen Meta-Angaben für Suchmaschinen, damit die Neuigkeit bald in aller Munde ist:

1. Name der Autorin
2. Inhaltsangabe: „Protokoll einer UFO-Sichtung“
3. Stichworte: „Aliens, UFO, Offenbach“
4. Erstelldatum
  ```

Wie sehen die Meta-Angaben aus?

```
<head>
  <meta name="author" content=
  "Claudia Schmidt">
  <meta name="description"
  content="Protokoll einer
  UFO-Sichtung">
  <meta name="keywords"
  content="Aliens, UFO,
  Offenbach">
  <meta name="date" content=
  "2000-12-03">
</head>
```

Zum Lösen der Aufgabe sollten dir die grundlegenden **Regeln** zur Syntax von Tags und zum Aufbau von HTML-Dateien bekannt sein.
Das Meta-Format ähnelt dem beim Erlauben und Verbieten des **Auslesens**.
Neu ist folgendes:

- In der Kopfsektion kann man Informationen über das Dokument – sogenannte Meta-Informationen – vermerken. Sie werden beim Browsen selbst nicht angezeigt.
- Angaben zu Autor und Inhalt sowie Schlüsselbegriffe und das Erstellungsdatum geben Suchmaschinen im WWW wertvolle Anhaltspunkte zur Archivierung und zum leichteren Auffinden der Seite.
- Das Format eines `<meta>`-Tags lautet

  `<meta name="`*Kategorie*`"`
  `content="`*Inhalt*`">`

Kategorie	*Inhalt*
`author` (Autor)	`Vor Nach`
`description` (Inhalt)	`Text`
`keywords` (Begriffe)	`a, b, c`
`date` (Datum)	`JJJJ-MM-TT`

34. *Auslesen* (Typ 5)

Die Bundeswehr führt bei Offenbach geheime Experimente mit neuartigen Militärflugzeugen durch. Da bereits Zeugen der Testflüge Berichte über UFO-Sichtungen ins Netz gestellt haben, bemüht man sich im selben Medium um eine Gegendarstellung. Zur Vermeidung weiterer Gerüchte wird in die Seite jedoch ein interner Vermerk eingefügt, der deren Erfassung durch Suchmaschinen unterbindet.
Wie sieht die für das Verbot benötigte Meta-Angabe aus?

```
<head>
  <meta name="robots" content=
  "noindex">
</head>
```

(alternativ: `content="none"`*)*

Zum Lösen der Aufgabe sollten dir die grundlegenden **Regeln** zur Syntax von Tags und zum Aufbau von HTML-Dateien bekannt sein.
Das Meta-Format ist ähnlich dem der **Beschreibung** des Dateiinhaltes.
Neu ist folgendes:

- In der Kopfsektion kann man Informationen über das Dokument – sogenannte Meta-Informationen – vermerken. Sie werden beim Browsen selbst nicht angezeigt.
- Meta-Angaben zu Suchmaschinen können diesen das Auslesen und Archivieren einer Seite verbieten oder explizit erlauben. Zusätzlich kann die Verarbeitung der verlinkten Unterseiten verboten oder eben erlaubt werden.
- Das Format dieser Meta-Information lautet

  `<meta name="robots"`
  `content="`*Inhalt*`">`

  ("robots" = „Suchmaschinen")

Inhalt	Bedeutung
`noindex`	Seite auslesen verbieten
oder `none`	
`index`	Seite auslesen erlauben
`nofollow`	Linkverfolgen verbieten
`follow`	Linkverfolgen erlauben

35. *Stichwortsprache* (Typ 5)

Notiere Meta-Angaben mit Stichworten auf

1. englisch:
   - Psychology
   - Perception
   - Attention
2. deutsch:
   - Psychologie
   - Wahrnehmung
   - Aufmerksamkeit

```
<head>
  <meta name="keywords" lang=
  "en" content="Psychology,
  Perception, Attention">
  <meta name="keywords" lang=
  "de" content="Psychologie,
  Wahrnehmung,
  Aufmerksamkeit">
</head>
```

Die Aufgabe erweitert die **Beschreibung** von HTML-Dateien mit dem `<meta>`-Tag.
Neu ist folgendes:

- Man kann Meta-Informationen gleich mehrfach in verschiedenen Sprachen notieren.
- Damit auch alle Zeilen berücksichtigt werden, ist jeweils das Attribut
  `lang="`*Sprache*`"`
  zu ergänzen.
- Erlaubte Sprachkürzel sind `de`, `en` (bzw. `en-us`), `fr` usw.

36. *Defaultsprachen* (Typ 5)

Notiere die Meta-Angabe für Sprachen und Zeichensätze, welche sicherstellt, daß Umlaute und das „ß“ richtig dargestellt werden.

```
<head>
  <meta http-equiv=
  "content-type" content=
  "text/html;charset=
  iso-8859-1">
</head>
```

Zum Lösen der Aufgabe sollten dir die grundlegenden **Regeln** zur Syntax von Tags und zum Aufbau von HTML-Dateien bekannt sein.
Das Meta-Format sieht so ähnlich aus wie die Anweisung zum Lesen von der **Originaladresse** oder wie die **Weiterleitung**.
Neu ist folgendes:

- In der Kopfsektion kann man Informationen über das Dokument – sogenannte Meta-Informationen – vermerken. Sie werden beim Browsen selbst nicht angezeigt.
- Meta-Angaben zu Sprachen ermöglichen u. a. die Festlegung auf einen Zeichensatz. Mittels einer einzigen Zeile kann man sicherstellen, daß deutsche Umlaute und das „ß“ überall auf der Welt richtig angezeigt werden.
- Das Format dieser Meta-Information lautet
  `<meta http-equiv=`
  `"content-type" content=`
  `"text/html;charset=`
  Zeichensatz`">`
  ("content" = „Inhalt“)

Zeichensatz	Sprachen
iso-8859-1	Westeuropa
iso-8859-2	slavische
iso-8859-3	Esperanto
iso-8859-4	baltische
iso-8859-9	Türkisch
iso-8859-10	Inuit, Sami

37. *Originaladresse* (Typ 5)

Angenommen, du betreust eine Börsen-Website, welche den Verlauf gewisser Aktienkurse überwacht. Es ist für deine Leser von Bedeutung, daß sie beim Surfen alle 30 Minuten die auf dem Server aktualisierten Daten geliefert bekommen.
Welche Meta-Angabe sorgt dafür?

```
<head>
  <meta http-equiv="expires"
  content="1800">
</head>
```

Zum Lösen der Aufgabe sollten dir die grundlegenden **Regeln** zur Syntax von Tags und zum Aufbau von HTML-Dateien bekannt sein.
Das Meta-Format sieht so ähnlich aus wie die Definition von **Defaultsprachen** oder wie die **Weiterleitung**.
Neu ist folgendes:

- In der Kopfsektion kann man Informationen über das Dokument – sogenannte Meta-Informationen – vermerken. Sie werden beim Browsen selbst nicht angezeigt.
- Die Meta-Angabe für das Laden von der Originaladresse garantiert, daß die Hypertext-Seite (nach Ablauf einer Frist) neu aus dem Internet geladen wird, und nicht eine eventuell veraltete Version aus einem Cache-Speicher. Die Leser bleiben so stets auf dem laufenden, müssen jedoch längere Ladezeiten in Kauf nehmen.
- Das Format dieser Meta-Information lautet

  ```
  <head>
    <meta http-equiv=
    "expires" content=
    "Inhalt">
  </head>
  ```

  („expires" = „läuft ab")
- *Inhalt* ist die Zeit in Sekunden, die ohne ein neuerliches Laden von der ursprünglichen Adresse verstreichen darf.

38. *Weiterleitung* (Typ 5)

Deine Homepage ist umgezogen! Du hast dir eine eigene Domain mit Namen `www.zeitvertreib.de` geleistet und alle Dateien hinüberkopiert. Auf der alten Adresse läßt du ein HTML-File zurück, das Besuchern die neue URL anzeigt und sie nach 10 Sekunden automatisch dorthin weiterleitet.
Welche Meta-Angabe implementiert den automatischen Weiterleitmechanismus?

```
<head>
  <meta http-equiv="refresh"
  content="10;URL=
  http://www.zeitvertreib.de">
</head>
```

Zum Lösen der Aufgabe sollten dir die grundlegenden **Regeln** zur Syntax von Tags und zum Aufbau von HTML-Dateien bekannt sein.
Das Meta-Format sieht so ähnlich aus wie die Anweisung zum Lesen von der **Originaladresse** oder wie die Definition von **Defaultsprachen**.
Neu ist folgendes:

- In der Kopfsektion kann man Informationen über das Dokument – sogenannte Meta-Informationen – vermerken. Sie werden beim Browsen selbst nicht angezeigt.
- Die Meta-Angabe zum Weiterleiten bewirkt, daß der Browser

nach einer Übergangszeit zu einer neuen Adresse verzweigt.

- Das Format dieser Meta-Information lautet
  `<head>`
  `  <meta http-equiv=`
  `  “refresh” content=`
  `  “`*Sekundenzahl*`;`
  `  URL=`*WWW-Adresse*`”>`
  `</head>`
  (“refresh” = „auffrischen“)

Globaldefinitionen

39. *dateiweit* (Typ 6)

Mal angenommen, du möchtest deinen Sportverein ins Internet bringen und bist gerade in der Phase, wo du globale Designfestlegungen triffst.

Zuerst einmal: Wie soll die Seite aussehen?

„Ich nehme das Design...“

`sterne.jpg`
gelb (255, 255, 0)
hellblau (0, 255, 255)
hellgrün (0, 255, 0)
schwarz (0, 0, 0)
`netz.gif`
weiß (255, 255, 255)
gelb (255, 255, 0)
hellblau (0, 255, 255)
pink (255, 0, 255)

Und dann soll im Hintergrund noch eine passende Musik abgespielt werden – welche?
„Ich nehme das Lied...“

`gowest.mid`
`finalcountdown.mid`

Okay, und jetzt implementiere deinen Entwurf, indem du das HTML-Fragment vervollständigst! Nimm die Farbe Grau (128, 128, 128) als Ersatzhintergrund für die Grafik dazu und binde die Musikdatei als unendliche Schleife in Netscape- und Microsoft-Syntax ein.

```
<head>

 <bgsound src="_____.mid"
 loop=infinite>

</head>

<body background="____"
bgcolor=#808080 text=__
link=__ vlink=__ alink=__>

 <embed src="_____.mid"
 autostart=true loop=true
 hidden=true height=0 width=0>

</body>
```

Hier geht es darum, die gloablen Festlegungen über die Gestalt einer Hypertext-Seite zu treffen. Dabei werden Gestaltungselemente aus mehreren Einzelaufgaben kombiniert: zum einen dateiweit gültige **Farben** für Hintergrund, Text und Links (`<body>`-Attribute `bgcolor`, `text`, `link`, `vlink`, `alink`), **Hintergrundbilder** (`background`) sowie Hintergrundmusik (`<bgsound ...>`/`<embed ...>` – beide ohne Endtag).

40. *Farben* (Typ 6)

Für die Wahl von dateiweit gültigen Vorder- und Hintergrundfarben sollte man neben der Ästhetik auch auf einen guten Helligkeitskontrast zwischen beidem achten.

Suche als erstes eine Hintergrundfarbe aus.
„Ich nehme als Hintergrund...“

Dunkelblau (0, 0, 128)
Silber/Hellgrau (192, 192, 192)

Wähle als nächstes passende Farben für die Schrift aus – Text, Link, benutzter Link und aktiver Link.
„Ich nehme die Textfarben. . . “

Türkis (0, 128, 128)
Pink (255, 0, 255)
Violett (128, 0, 128)
Schwarz (0, 0, 0)
Weiß (255, 255, 255)
Rot (255, 0, 0)
Grau (128, 128, 128)
Gelb (255, 255, 0)

Und nun implementiere deinen Farben-Entwurf, indem du das benötigte einleitende Tag angibst (Farbattribute ohne Anführungszeichen).

```
<body bgcolor=__ text=__
link=__ vlink=__ alink=__>
```

Du brauchst einfach nur das einleitende `<body>`-Tag um das Attribut für die **Hintergrundfarbe** `bgcolor` und die vier **Textfarben**-Attribute `text`, `link`, `vlink` und `alink` mit den entsprechenden Werten zu ergänzen.

42. *Textfarbe* (Typ 6)

Standardmäßig geben Browser dem Hypertext eine schwarze Schrift mit blauen Links sowie einen weißen oder hellgrauen Hintergrund. Es ist aber auch möglich, selbst dateiweit gültige Farben, z. B. für den Text, vorzugeben.

Wähle als erstes eine Farbe für den normalen Text.
„Ich nehme als Textfarbe. . . “

Grün (0, 128, 0)
Dunkelrot (128, 0, 0)

Wähle als nächstes passende Farben für die Links – unbenutzte, benutzte und angeklickte.
„Ich nehme die Linkfarben. . . “

Oliv (128, 128, 0)
Türkis (0, 128, 128)
Hellgrün (0, 255, 0)
Rot (255, 0, 0)
Schwarz (0, 0, 0)
Gelb (255, 255, 0)

Und nun implementiere deinen Farben-Entwurf, indem du das benötigte einleitende Tag angibst (Farbattribute ohne Anführungszeichen).

```
<body text=__ link=__
vlink=__ alink=__>
```

Als Voraussetzungen dürftest du bereits die grundsätzlichen **Regeln** und die **Farbdefinition** beherrschen.
Hier geht es nun darum, Farben einzusetzen.
Die globalen Textfarben gelten erstmal für das ganze Dokument, können aber für Textstellen durch **Schriftfarben** überlagert werden.

- Die globalen Textfarben kommen als Attribute in das einleitende `<body>`-Tag.
- `text=`*Farbe* steht für den normalen Lesetext.
- `link=`*Farbe* steht für mit einem Link unterlegten Text.
- `vlink=`*Farbe* steht für die Farbe, die ein besuchter Link annimmt (“visited”).
- `alink=`*Farbe* steht für die Farbe, die ein aktiver, d. h. gerade angeklickter, Link annimt.
- Der Farbwert kann im Hexadezimalcode oder als Name angegeben werden.

44. *Hintergrundmusik* (Typ 6)

Suche eine Hintergrundmusik für die Website einer Videothek aus.
„Ich nehme die Musikdatei. . . “

```
starwars.mid
ghostbusters.mid
flintstones.mid
```

Definiere deine Hintergrundmusik als Endlosschleife in Netscape-Syntax.

```
<body>
 <embed src="_____.mid"
 autostart=true loop=true
 hidden=true height=0 width=0>
</body>
```

Und gleich dazu in der Schreibweise für Microsoft, ebenfalls unendlich oft abgespielt.

```
<head>
 <bgsound src="_____.mid"
 loop=infinite>
</head>
```

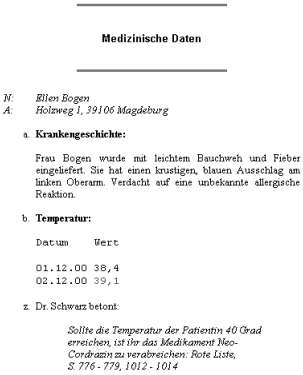
Du brauchtst erstmal nur die allgemeinen **Regeln** von HTML zu kennen. Weitere Erklärungen folgen.
Mit den Browsern von Netscape und Microsoft kann man während des Anzeigens einer Hypertext-Seite zusätzlich ein Musikstück im Hintergrund abspielen! Und das geht so:

- Es wird eine Musikdatei, entweder `.mid`, `.wav` oder `.au`, in das Dokument eingebunden.
- "Netscape Navigator" und "Internet Explorer" interpretieren jeder für sich eine eigene Notation. Man sollte beide kennen und beide in den Quellcode aufnehmen.
- Netscape – im HTML-Rumpf:
  `<embed src="`*Musikdatei (ggf. mit Pfad)*`" autostart= true loop=true hidden= true height=0 width=0>`
  („Einbetten der Quelle *Musikdatei* mit automatischem Start, als Schleife und ohne Anzeige eines Players")
- Microsoft – im HTML-Kopf:
  `<bgsound src="`*Musikdatei (ggf. mit Pfad)*`" loop= infinite>`
  („Hintergrundgeräusch aus der Quelle *Musikdatei*, undendlich oft wiederholt")

Textgestaltung

45. *Textgestaltung* (Typ 5)

Es ist deine Aufgabe, den abgebildeten Hypertext-Entwurf zu implementieren.



Medizinische Daten

N: *Ellen Bogen*
A: *Holzweg 1, 39106 Magdeburg*

a. **Krankengeschichte:**

Frau Bogen wurde mit leichtem Bauchweh und Fieber eingeliefert. Sie hat einen krustigen, blauen Ausschlag am linken Oberarm. Verdacht auf eine unbekannte allergische Reaktion.

b. **Temperatur:**

Datum	Wert
01.12.00	38,4
02.12.00	39,1

z. Dr. Schwarz betont:

Sollte die Temperatur der Patientin 40 Grad erreichen, ist ihr das Medikament Neo-Cordrazin zu verabreichen: Rote Liste, S. 776 - 779, 1012 - 1014

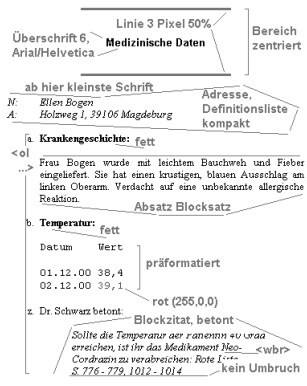
Keine Grund zur Sorge: Der größte Teil des Quellcode-Ausschnitts steht schon da. Vervollständige nur noch die Lücken, wo Tags, Attribute und Attributwerte fehlen.
Vorsicht: Einige wenige Felder bleiben unausgefüllt. Sie sind für unterschiedliche Lösungswege gedacht.

```
<div align=center>
```
(alternativ mit `<center>`*)*

```
<hr noshade size=3 width=50%>
<h6>
<font face="Arial,Helvetivca">
Medizinische Daten
</font>
</h6>
<hr noshade size=3 width=50%>
</div>
<basefont size=1>
```
(alternativ mit `font`*)*

```
<address>
<dl compact>
<dt>N:</dt>
<dd>Ellen Bogen</dd>
<dt>A:</dt>
<dd>Holzweg 1,
39106 Magdeburg</dd>
</dl>
</address>
<ol type=a>
<li>
<b>Krankengeschichte:</b>
<p align=justify>
Frau Bogen wurde mit leichtem
Bauchweh und Fieber
eingeliefert. Sie hat einen
krustigen, blauen Ausschlag
am linken Oberarm. Verdacht
auf eine unbekannte
allergische Reaktion.
</p>
</li>
<li>
<b>Temperatur:</b>
<pre>
Datum    Wert
01.12.00 38,4
02.12.00 <font color=red>39,1
</font>
(alternativ color=#FF0000)
</pre>
</li>
<li value=26>
Dr. Schwarz betont:
<blockquote>
<em>
Sollte die Temperatur der
Patientin 40 Grad erreichen,
ist ihr das Medikament
Neo-<wbr>Cordrazin zu
verabreichen: Rote Liste,
<nobr>S. 776 - 
779, 1012 -
1014</nobr>
(eins von beiden genügt)
</em>
</blockquote>
</li>
</ol>
(</font>)
```

Dies ist die umfangreichste Aufgabe im Programm. In ihr sind nochmal alle Aspekte der Textgestaltung zusammengefaßt: **Überschriften**, **Absätze**, **Umbrüche**, verschiedene **Listen** sowie **Zitate und Adressen**. **Präformatieren**, das **logische** oder **physische** Textauszeichnen, die drei **Schriftoperationen**, **Bereiche** und **Trennlinien**.
Welcher Art die einzelnen Formatierungen sind, kannst du in der folgenden Grafik ablesen.



46. *Ueberschriften* (Typ 3)

Wähle eine Überschriftengröße.

```
1     6
| | | | | |
```

Nun mache aus folgendem Text eine zentriert ausgerichtete Überschrift der gewählten Ordnung.

```
<h  align=center>
Genmanipulation: ja oder nein?
```

```
</h  >
```

Man kann die **Überschriftdefinition** um das Attribut zur **Ausrichtung** von Text (`align`) ergänzen. Die Überschrift kann links- (Normalfall) oder rechtsbündig (`right`), zentriert (`center`) oder als Block (`justify`) über den Text gesetzt werden.

47. *Ueberschriftdefinition* (Typ 3)

Wähle eine Überschriftengröße.

```
1       6
| | | | | |
```

Erhebe nun den folgenden Text zu einer Überschrift der gewählten Ordnung.

```
<h  >Die Proteinbiosynthese
</h  >
```

Die **Regeln** zu Tags und dem Umgang mit normalem Text dürften dir ja bereits bekannt sein. In dieser Aufgabe bearbeitest du Text mithilfe eines Tags.

- Man kann aus einem Textstück eine Überschrift machen. Dieses wird dann i. a. größer und fett gedruckt dargestellt. Darüber hinaus setzt es der Browser nach oben und nach unten vom übrigen Text ab.
- Es gibt Überschriften in sechs Größenabstufungen: `<h1>` steht für sehr groß und `<h6>` für sehr klein.
- Der Überschrifttext wird vom einleitenden und vom abschließenden Überschriften-Tag einer der sechs Größen umschlossen.

49. *Absaetze* (Typ 5)

Mache aus folgendem Text einen Absatz im Blocksatz.

> Im schummrigen Fackellicht konnte der Reisende nur Umrisse einer gedrungenen Gestalt ausmachen. All seinen Mut zusammennehmend trat er dem Fremden entgegen.

Vervollständige dazu das angegebene HTML-Fragment. Benutze Anfangs- und Endtags.

```
<p align=justify>Im
schummrigen Fackellicht
konnte der Reisende nur
Umrisse einer gedrungenen
Gestalt ausmachen. All seinen
Mut zusammennehmend trat er
dem Fremden entgegen.</p>
```

Für Text, der nicht wie gewöhnlich linksbündig ausgerichtet sein soll, erweitere einfach die **Absatzdefinition** (`<p>`) um das Attribut zum **Ausrichten** (`align` mit Werten `right`, `center` oder `justify`).

50. *Absatzdefinition* (Typ 5)

Implementiere den abgebildeten Hypertext-Entwurf mit zwei Absätzen.

> Vorsichtig schritt der Reisende durch das Tor.
>
> 'Halt! Wer da?' erklang eine tiefe Stimme.

Vervollständige dazu das angegebene HTML-Fragment. Benutze Anfangs- und Endtags.

```
<p>Vorsichtig schritt der
Reisende durch das Tor.</p>
<p>'Halt! Wer da?' erklang
eine tiefe Stimme.</p>
```

Für die Aufgabe mußt du erstmal nur die allgemeinen **Regeln** kennen. Neu ist dann folgendes:

- Man kann und sollte Text in Absätzen zusammenfassen.

- Der Text eines Absatzes wird von `<p>` und `</p>` umschlossen ("paragraph" = „Absatz").

51. *Umbrueche* (Typ 5)

Setze den folgenden Ausschnitt aus einem Literaturverzeichnis in HTML um. Berücksichtige drei Vorgaben.

- Der Text besteht aus drei Zeilen.
- Der Doppelname darf im Browser nach dem Bindestrich getrennt werden.
- Die Seitenangabe darf nicht getrennt werden.

[1]
Jung, Carl-Gustav: Die Funktionen des Unbewussten
In Traum und Traumdeutung, S. 18 – 25

Vervollständige nun das angegebene HTML-Fragment.

```
[1]<br>
Jung, Carl-<wbr>Gustav: Die
Funktionen des Unbewussten<br>
In Traum und Traumdeutung,
<nbr>S. 18 - 25</nbr>
(S. 18 - 25)
```

Die Lösung der drei Teilaufgaben besteht darin, Tags zur Umbruchsteuerung anzuwenden.

- **Erzwingen eines Umbruchs** (`<br>`)
- **Erlauben eines Umbruchs** (`<wbr>`)
- und **Verhindern eines Umbruchs** (`<nobr>`) oder **Leerzeichen** (` `)

52. *ErzwingenUmbruch* (Typ 5)

Implementiere den abgebildeten Hypertext-Entwurf mit Umbrüchen zwischen den Zeilen.

100g Mehl
20g Butter
2 Eier
1 TL Salz
eine halbe Knoblauchzehe

Vervollständige dazu das angegebene HTML-Fragment.

```
100g Mehl<br>
20g Butter<br>
2 Eier<br>
1 TL Salz<br>
eine halbe Knoblauchzehe
```

Die Syntax der erzwungenen Umbrüche gleicht der des **Erlaubens eines Umbruchs**.
Für die Aufgabe mußt du erstmal nur die allgemeinen **Regeln** kennen.
Neu ist dann folgendes:

- Da ja normalerweise Zeilen automatisch so getrennt werden, daß möglichst viele Wörter ins aktuelle Browserfenster passen, bietet HTML die Möglichkeit dort selbst einzugreifen und eigene Umbrüche zu setzen.
- Mit dem alleinstehenden `<br>`-Tag ("break" für „Umbruch") beendet man eine Zeile (bzw. beginnt eine neue).

53. *VerhindernUmbruch* (Typ 5)

Schreibe die abgebildete Formel (mit Leerzeichen) so in HTML, daß keine Gefahr besteht, daß ein Browser sie auf mehrere Zeilen verteilt. Benutze das `<nobr>`-Tag.

MR = (MU + KL + ST) / 3 - 2 * AG

Trage deine Lösung in die Felder ein.

```
<nobr>MR = (MU + KL + ST) / 3
 - 2 * AG</nobr>
```

Für die Aufgabe mußt du erstmal nur die allgemeinen **Regeln** kennen.
Neu ist dann folgendes:

- Da normalerweise Zeilen automatisch mit Erreichen des Fensterrahmens an einem Leerzeichen getrennt werden, gibt es in HTML auch die Möglichkeit, diesen Automatismus zu unterbinden. Überlange Zeilen werden entweder über den anzeigbaren Bereich hinaus weitergeführt oder in der nächsten Zeile fortgesetzt.
- Ein Kommando zum Ausweisen von zusammenhängendem Text ist `<nobr>` ("no break" = „kein Umbruch"). Der Gültigkeitsbereich wird mit `</nobr>` abgeschlossen.

54. *Leerzeichen* (Typ 5)

Im abgebildeten Textstück soll verhindert werden, daß die Seitenangabe durch automatische Zeilenumbrüche zerstückelt wird. Wie kann man dies mit drei geschützten Leerzeichen erreichen?

(mehr dazu im Lehrbuch auf S. 8 - 12)

Trage deine Lösung in die Felder ein.

```
mehr dazu im Lehrbuch auf
S. 8 - 12
```

Die Syntax dieser speziellen Leerzeichen gleicht der von **Sonderzeichen**.
Um die Aufgabe lösen zu können, ist erstmal nur die Kenntnis der allgemeinen **Regeln** erforderlich.
Neu ist dann folgendes:

- Da normalerweise Zeilen automatisch mit Erreichen des Fensterrahmens an einem Leerzeichen getrennt werden, gibt es in HTML auch die Möglichkeit, diesen Automatismus zu unterbinden. Eine Lösung dafür ist, Wörter mit geschützten Leerstellen „zusammenzukleben".
- Ersetze einfach alle „zu schützenden" Spaces durch den Ausdruck ` ` (steht für "nonbreaking space" = „nichtumzubrechende Leerstelle").

55. *ErlaubenUmbruch* (Typ 5)

Füge in das lange „Wort"
www.bs-net.de/kultur/kino/welcome.html
hinter jedem Schrägstrich die Erlaubnis für einen Umbruch ein.

Hier ist Platz für deine Lösung.

```
www.bs-net.de/<wbr>kultur/
<wbr>kino/<wbr>welcome.html
```

Die Syntax der erlaubten Umbrüche gleicht der des **Erzwingens eines Umbruchs**.
Um die Aufgabe lösen zu können, ist erstmal nur die Kenntnis der allgemeinen **Regeln** erforderlich.
Neu ist dann folgendes:

- An sich trennen Browser Zeilen nur an Leerstellen automatisch. Es gibt aber ein Tag zum expliziten Einfügen von erlaubten Trennstellen in Wörter.
- Das Tag heißt `<wbr>` (für "word break = „Wortumbruch") und wird einfach mitten ins Wort geschrieben, z. B.

  ```
  Hans-<wbr>Georg
  ```

- Der Umbruch erfolgt allein am rechten Fensterrand und ohne einen zusätzlichen Trennungsstrich. Woanders passiert überhaupt nichts; es wird auch kein Leerzeichen eingefügt.

56. *Nummerlisten* (Typ 5)

Implementiere eine numerierte Liste.

B. I-Amt
C. Uni-Klinik
F. Sporthalle
G. Bibliothek
H. Mensa

Ergänze dazu den Lückentext.

```
<ol type=A start=2>
  <li>I-Amt</li>
  <li>Uni-Klinik</li>
  <li value=6>Sporthalle</li>
  <li>Bibliothek</li>
  <li>Mensa</li>
</ol>
```

Hier sollen die Attribute für numerierte Listen (`<ol>`) kombiniert werden; das sind **Nummertyp** (`type`) und **Startnummer** (`start`, `value`).

57. *Nummerlistendefinition* (5)

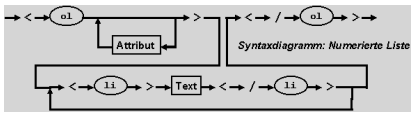
Implementiere eine numerierte Liste.

1. Abwaschen
2. Einkaufen
3. Essen kochen

Ergänze dazu den Lückentext.

```
<ol>
  <li>Abwaschen</li>
  <li>Einkaufen</li>
  <li>Essen kochen</li>
</ol>
```

Numerierte Listen funktionieren ähnlich wie **Aufzähllisten** und **Definitionslisten**.
Die **Regeln** für Tags und die Anzeige von normalem Text kennst du ja bereits.
Hier wird nun ein Tag vorgestellt, mit welchem man Listen erzeugen kann, deren Einträge in ihrer Reihenfolge automatisch numeriert werden.



- Den Rahmen einer numerierten Liste bilden die Tags `<ol>` und `</ol>` (“ordered list”, „geordnete Liste“).
- Jeder einzelne Texteintrag wird von `<li>` und `</li>` umschlossen.
- Die Nummern gibt man nicht selbst an. Sie werden automatisch generiert.
- Im einleitenden `<ol>`-Tag können spezielle Attribute untergebracht werden – aber dazu an anderer Stelle mehr.

58. *Nummertyp* (Typ 5)

Implementiere eine numerierte Liste.

I. Aufbau
II. Beobachtung
III. Deutung

Ergänze dazu den Lückentext.

```
<ol type=I>
  <li>Aufbau</li>
  <li>Beobachtung</li>
  <li>Deutung</li>
</ol>
```

Jetzt wird für die **Nummerlistendefinition** `<ol>` ein Attribut benötigt.

- Das Attribut `type=`*Format* bewirkt eine von den gewöhnlichen Ziffern abweichende Numerierung – entweder Buchstaben oder römische Zahlen.
-

Format	Numerierung
`a`	a., b., c., ...
`A`	A., B., C., ...
`i`	i., ii., iii., ...
`I`	I., II., III., ...

59. *Startnummer* (Typ 5)

Implementiere eine numerierte Liste.

101. Meier
102. Schultz
103. Krause
201. Vasquez
202. Wilke

Ergänze dazu den Lückentext.

```
<ol start=101>
  <li>Meier</li>
  <li>Schultz</li>
  <li>Krause</li>
  <li value=201>Vasquez</li>
  <li>Wilke</li>
</ol>
```

Die **Nummerlistendefinition** `<ol>` wird um Attribute erweitert.

- Die numerierte Liste muß nicht bei 1 (bzw. a, A, i oder I) beginnen. Mit dem Attribut
  `start=`*Nummer*
  im Listen-Tag kann man eine Startnummer über 1 (also nicht 0!) festlegen (auch bei Buchstaben- oder römischer Numerierung die *Nummer* als Ziffer angeben!).
- Genausowenig muß die Numerierung über die gesamte Liste hinweg gleichmäßig ansteigen. Mit dem Attribut
  `value=`*Nummer*
  in einem Listeneintrag-Tag bewirkt man, daß der Nummernzähler auf einen neuen Wert gesetzt wird.

60. *Aufzaehllisten* (Typ 5)

Implementiere eine Aufzählungsliste mit Kreisen als Markierungen.

- Bremen
- Hamburg
- Berlin

Ergänze dazu den Lückentext.

```
<ul type=circle>
  <li>Bremen</li>
  <li>Hamburg</li>
  <li>Berlin</li>
</ul>
```

Jetzt wird für die **Aufzählllistendefinition** `<ul>` ein Attribut benötigt.

- Durch das Attribut `type=`*Format* erhält man unterschiedliche Markierungen der Listenelemente. Das Aussehen ist im Einzelfall browser- und schriftartabhängig.
-

Format	Markierung
`circle`	Kreis
`square`	Quadrat
`disc`	„Diskette“

61. *Aufzaehllistendefinition* (5)

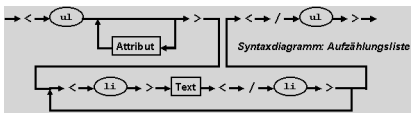
Implementiere eine Aufzählungsliste.

- ca. 1,85m
- dunkle Haare
- Narbe im Gesicht

Ergänze dazu den Lückentext.

```
<ul>
  <li>ca. 1,85m</li>
  <li>dunkle Haare</li>
  <li>Narbe im Gesicht</li>
</ul>
```

Aufzählungslisten funktionieren ähnlich wie **Nummerlisten** und **Definitionslisten**.
Die **Regeln** für Tags und die Anzeige von normalem Text kennst du ja bereits.
Hier wird ein Tag vorgestellt, durch welches man Listen mit einem Punkt vor jedem Eintrag erzeugt.



- Den Rahmen der Aufzählungsliste bilden die Tags `<ul>` und `</ul>` („unsortierte Liste“).
- Jeder einzelne Texteintrag wird von `<li>` und `</li>` umschlossen.
- Im einleitenden `<ul>`-Tag können spezielle Attribute untergebracht werden – aber dazu an anderer Stelle mehr.

62. *Definitionslisten* (Typ 5)

Implementiere eine kurze „kompakte“ Definitionsliste.

BS Braunschweig
WOB
    Wolfsburg

Ergänze dazu den Lückentext.

```
<dl compact>
  <dt>BS</dt>
  <dd>Braunschweig</dd>
  <dt>WOB</dt>
  <dd>Wolfsburg</dd>
</dl>
```

Die Aufgabe enthält eine kleine Erweiterung der **Definitionslistendefinition** (`<dl>`, `<dt>`, `<dd>`).

- Durch das Attribut `compact` im Listen-Tag erhält die Definitionsliste ein „kompakteres“ Erscheinungsbild.
- Begriff und Erklärung werden dann vom Browser in dieselbe Zeile geschrieben, wenn genügend Platz ist.

63. *Definitionslistendefinition* (5)

Implementiere eine kurze Definitionsliste.

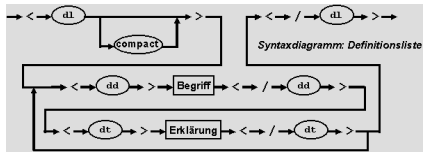
SNP
    Scottish National Party
PC
    Plaid Cymru

Ergänze dazu den Lückentext.

```
<dl>
  <dt>SNP</dt>
  <dd>Scottish National
  Party</dd>
  <dt>PC</dt>
  <dd>Plaid Cymru</dd>
</dl>
```

Definitionslisten funktionieren in etwa so wie **Nummerlisten** und **Aufzählungslisten**.

Die **Regeln** für Tags und die Anzeige von normalem Text kennst du ja bereits.
Hier wird ein Tag vorgestellt, durch welches man formatierte Listen von Begriffen mit ihren Erklärungen erzeugen kann.



- Den Rahmen der Definitionsliste bilden die Tags `<dl>` und `</dl>`.
- Jeder Eintrag besteht aus dem Paar
  - „Begriff“, eingerahmt von `<dt>` und `</dt>` (“term”),
  - sowie seiner „Erklärung“, eingerahmt von `<dd>` und `</dd>` (“definition”).
- Das Attribut `compact` spielt in dieser Aufgabe noch keine Rolle.

64. *ZitateAdressen* (Typ 5)

Implementiere ein Textstück mit Zitat und Adresse.

Das Umweltamt verwies auf:

    Das Stadtreinigungsamt
    Leipziger Str. 57

Meldet Euch dort!

Ergänze den Lückentext.

```
Das Umweltamt verwies auf:
<blockquote>
Das Stadtreinigungsamt
<address>Leipziger Str. 57
</address></blockquote>
Meldet Euch dort!
```

Die Lösung ist die Verschachtelung einer **Adresse** (`<address>`) in einem **Zitat** (`<blockquote>`).

65. *Zitat* (Typ 5)

Implementiere ein Textstück mit Zitat.

> Viele behaupten:
>
> Der Weg ist das Ziel.
>
> Stimmt das?

Ergänze einfach den Lückentext.

```
Viele behaupten:
<blockquote>Der Weg ist das
Ziel.</blockquote>
Stimmt das?
```

Du kennst ja bereits die allgemeinen **Regeln** zur Textbearbeitung: Textabschnitte werden durch die Anwendung von Tags manipuliert.
Bei dem hier eingeführten Tag handelt es sich genaugenommen um eine **logische Textauszeichnung**.

- Wenn die Tags `<blockquote>` und `</blockquote>` einen Text umschließen, ist dieser ein Zitat.
- Üblicherweise wird ein solches Zitat vom übrigen Text abgesetzt und von links eingerückt dargestellt.

66. *Adresse* (Typ 5)

Implementiere einen Text mit Adresse (ohne Link).

> Melden Sie Ihren Fernseher unter
> *registrierung@gez.de*
> an.

Ergänze einfach den Lückentext.

```
Melden Sie Ihren Fernseher
unter
<address>registrierung@gez.de
</address>
an.
```

Du kennst ja bereits die allgemeinen **Regeln** zur Textbearbeitung: Textabschnitte werden durch die Anwendung von Tags manipuliert.
Bei dem hier eingeführten Tag handelt es sich genaugenommen um eine **logische Textauszeichnung**.

- Wenn die Tags `<address>` und `</address>` einen Text umrahmen, ist dieser eine Adresse.
- Üblicherweise werden Adressen in eine eigene Zeile gesetzt und kursiv dargestellt.

67. *praeformatiert* (5)

Sorge mit einem Anfangs- und einem Endtag dafür, daß das folgende Textstück beim Anzeigen genau so formatiert, jedoch höchstens 640 Pixel breit, dargestellt wird.

```
be      was     been
        were
begin   began   begun
blow    blew    blown
build   built   built
```

Trage Tags (und Attribute) ein.

```
<pre width=640>

be      was     been
        were
begin   began   begun
blow    blew    blown
build   built   built

</pre>
```

Schreibe in die **Formatierungsdefinition** (`<pre>`) das allgemeine Attribut für die **Breite** (`width`).
Achtung: Breitenangaben in präformatiertem Text werden in den Programmen von Netscape und Microsoft (noch) nicht berücksichtigt!

68. *Formatierungsdefinition* (5)

Sorge mit einem Anfangs- und einem Endtag dafür, daß das folgende Programmstück im Browser genau so formatiert dargestellt wird.

```
BEGIN
    WRITELN ('Hallo Welt');
    WRITELN ('----- ----')
END
```

Ergänze einfach die Tags.
<pre>

```
BEGIN
    WRITELN ('Hallo Welt');
    WRITELN ('----- ----')
END
```

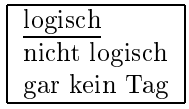
</pre>

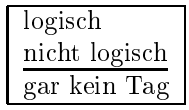
Die grundlegenden **Regeln** besagen, daß der Hypertext anders formatiert wird, als es der Text in der HTML-Datei ist.

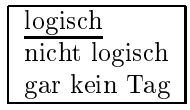
- Mit den zwei Tags `<pre>` und `</pre>` („präformatiert“) kann man genau diese Regel in einem Abschnitt aufheben.
- Der Text im Gültigkeitsbereich wird dann in „Schreibmaschinenschrift“ mit all seinen Leerzeilen, Leerzeichen, Einrückungen usw. angezeigt.
- HTML-Tags behalten im präformatierten Bereich ihre Wirkung!
- Anwendungen sind die Darstellung von Quellcode, kleine Tabellen u. ä.

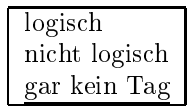
70. *logisch* (Typ 2)

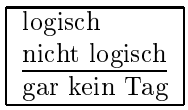
Gib für jeden Steuerbefehl an, ob es sich um eine logische Textauszeichnung handelt.

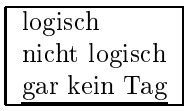
- `<strong>` 

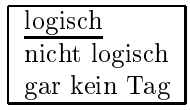
- `<b>` 

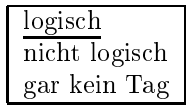
- `<code>` 

- `"` 

- `<sub>` 

- `\emph` 

- `<samp>` 

- `<em>` 

Hier werden die **Regeln** für das Manipulieren von Text mittels Tags konkretisiert.
Das logische Textauszeichnen ist das Gegenstück zur **physischen Textauszeichnung**.

- Es gibt eine Menge Steuerbefehle für einfache Schriftmodifikationen, mit denen die Bedeutung der Textformatierung spezifiziert werden kann.
- `<em>` beispielsweise kennzeichnet „betonten“ Text. Der wird in der Regel von Browsern kursiv angezeigt; Vorschrift ist dies jedoch nicht.

Tag	Auswirkung
`<em>`	betont ("emphasized")
`<strong>`	stark
`<code>`	Quellcode
`<cite>`	kurzes Zitat
`<dfn>`	Definition
`<samp`	Beispiel

  (Logische Tags bestehen immer aus einleitendem und abschließendem Tag.)

71. *physisch* (Typ 5)

    Implementiere ein Textstück mit verschiedenen physischen Textformatierungen.

    Su. Nachmieter: **1-Zi.-App.**, 14 m^2, Kochni., Du, WC, ***254***,– w, *frei ab* 15/01/01

    Ergänze den Lückentext.

    ```
    <u>Su. Nachmieter:</u> <b>
    1-Zi.-App.</b>, 14 m<sup>2
    </sup>, Kochni., Du, WC, <b>
    <i>254,-</i></b> w, <i>frei ab
    </i> <tt>15/01/01</tt>
    ```

    Jetzt werden die **Regeln** für das Manipulieren von Text mittels Tags in die Tat umgesetzt!
    Physische Textauszeichnung ist das Gegenstück zur **logischen Textauszeichnung**.

    - Es gibt eine Menge Steuerbefehle für einfache Schriftmodifikationen, über die man direkt das Erscheinungsbild von Textstücken bestimmen kann.
    - Alle physischen Tags bestehen aus einleitendem und abschließendem Tag.

Tag	Auswirkung
`<b>`	fett ("bold")
`<i>`	kursiv ("italic")
`<u>`	unterstrichen
`<tt>`	Schreibmaschine ("teletyper")
`<sup>`	hoch ("super")
`<sub>`	tief ("sub")

72. *Schriftoperationen* (Typ 6)

    Wähle als erstes ein Themengebiet, zu welchem du eine Website aufbauen möchtest.
    „Ich nehme das Thema. . . "

Umweltschutz
eine Popgruppe

    Wähle als nächstes ein angemessenes Schriftbild für die Seiten.
    „Ich nehme die Schrift. . . "

Arial–Helvetica 4/+1 grün
Jester–Comic Sans MS 5/+2 rot

    Teste nun deinen Entwurf, indem du ein einleitendes Schriften-Tag, erweitert um Attribute für Art und Farbe, aufstellst. Es kann dann auf Textabschnitte angewendet werden.

    ```
    <font face="__,__" color=__>
    ```

    Jetzt gib noch ein einzelnes oder einleitendes Tag an, das ein Textstück auf deine vorgegebene Größe bringt.

    ```
    <font size=__>
    ```
    oder
    ```
    <basefont size=__>
    ```

    Die Aufgabe soll `<font>`-Definitionen einüben helfen. Für den ersten Teil benötigt man Kenntnis der Attribute für **Schriftart** (`face`) und **Schriftfarbe** (`color`). Für den zweiten Teil braucht man entweder die Kenntnis des **Schriftgröße**-Attributes (`size`) oder das **Normalschriftgröße**-Tag (`basefont`).

73. *Schriftgroesse* (Typ 3)

    Wähle zuerst eine Schriftgröße.

    2 6
    | | | | |

Stelle jetzt das Wort AUTOMATENTHEORIE in deiner Schriftgröße dar. Wie? Das wäre zu einfach?? Na gut: Abweichend davon soll das Teilwort TOMATEN durch eine noch größere Schrift hervorgehoben sein, und zwar vier Stufen über der Größe normaler HTML-Schrift. Verwende ausschließlich `<font>`-Tags.

```
<font size=  >AU
<font size=+4>TOMATEN</font>
THEORIE</font>
```

(alternativ zu `+4` *geht auch* `7`*)*

Die Aufgabe dürfte ziemlich einfach sein, wenn du bereits die **Schriftfarbe**, die **Schriftart** oder die **Normalschriftgröße** definieren kannst.
Wenn dir die allgemeinen **Regeln** bekannt sind, dann weißt du auch, wie man Tags und Attribute benutzt.

- Zum Verändern des Schriftbildes gibt es das Tag `<font>` mit `</font>`.
- Um die Größe von Schrift festzulegen, erweitert man das Tag um ein Attribut `size=...`
- Die Werte des Attributes können absolut,
  `1` (klein) bis `7` (groß),
  oder relativ zur Normalschriftgröße (voreingestellt `3`),
  `-2` (zwei Stufen kleiner)
  bis `+4` (vier Stufen größer),
  notiert werden.
- Schriften-Tags kann man beliebig schachteln. Bei einer Verwendung innerhalb von Überschriften hängt die dargestellte Größe auch von der Überschriftgröße ab. Bei mehreren Schriftgrößentags gilt dagegen jeder Wert für sich.
- Größe, Farbe und Art können in einem Schriften-Tag zusammen definiert werden.

74. *Schriftfarbe* (Typ 4)

Wähle einen Farbton.

green 0,128,0	lime 0,255,0
olive 128,128,0	teal 0,128,128

Und jetzt sorge dafür, daß in dem Text GRUENDERZEIT das Teilwort GRUEN deinen Farbton bekommt.

```
<font color=       >GRUEN
</font>DERZEIT
```

(Attributwerte ohne Anführungszeichen)

Die Aufgabe dürfte leichter verständlich sein, wenn du schon die **Schriftgröße** oder die **Schriftart** definieren kannst.
Ansonsten solltest du die allgemeinen **Regeln** kennen und wissen, wie eine **Farbdefinition** funktioniert.
Neu ist folgendes:

- Zum Verändern des Schriftbildes gibt es das Tag `<font>` mit `</font>`.
- Um die Farbe von Schrift festzulegen, erweitert man das Tag um ein Attribut `color=...`
- Der Wert des Attributes ist eine Farbe – egal, ob hexadezimal codiert oder als Name.
- Größe, Farbe und Art können in einem Schriften-Tag zusammen definiert werden.

75. *Schriftart* (Typ 5)

Stelle in dem Wort AUTOMATENTHEORIE das Teilwort TOMATEN in der Schriftart “Arial” (Windows) dar. Gib für UNIX-Benutzer auch die Schrift “Helvetica” mit an.

AUTOMATENTHEORIE

Trage deine Antwort in den HTML-Lückentext ein.

```
AU<font face="Arial,
Helvetica">TOMATEN</font>
THEORIE
```

Die Aufgabe dürfte am einfachsten sein, wenn du bereits die **Schriftfarbe** oder die **Schriftgröße** definieren kannst.
Wenn dir immerhin die allgemeinen **Regeln** bekannt sind, dann weißt du ja, wie man Tags und Attribute benutzt.

- Zum Verändern des Schriftbildes gibt es das Tag `<font>` mit `</font>`.
- Um eine Schriftart zu spezifizieren, erweitert man das Tag um ein Attribut `face=...`
- Wert des Attributes ist die exakte Bezeichnung der verwendeten Schriftart, z. B. "Comic Sans MS", in Anführungsstriche geschrieben.
- Aber Vorsicht: Kein Benutzer hat genau dieselben Schriftarten wie du installiert. Es verwendet nicht einmal jedes Betriebssystem dieselben Namen für ähnliche Fonts. Deshalb ist es immer von Vorteil, mehrere Schriftartnamen anzugeben. Man schreibt sie durch Kommas getrennt hintereinander in die Anführungszeichen, die bevorzugten Schriften zuerst.
- Größe, Farbe und Art können in einem Schriften-Tag zusammen definiert werden.

76. *Normalschriftgroesse* (Typ 3)

Wähle eine Normalschriftgröße.

```
4    7
| | | |
```

Und jetzt sorge dafür, daß in dem Text GRUENDERZEIT für das Teilwort GRUEN deine eingestellte Normalschriftgröße gilt. Danach schalte wieder wieder auf den voreingestellten Wert um.

```
<basefont size= >GRUEN<basefont size=3>DERZEIT
```

Die Aufgabe dürfte leichter verständlich sein, wenn du schon eine lokale **Schriftgröße** definieren kannst.
Wenn dir die allgemeinen **Regeln** bekannt sind, dann weißt du, wie man Tags und Attribute benutzt.
Neu ist folgendes:

- Abgesehen von Stellen, wo die Größe der Schrift per `<font>`- oder Überschriftentag geändert wird, gilt überall die Normalschriftgröße 3.
- Mit dem Tag `<basefont>` (ohne Abschluß) kann dieser global gültige Wert ab einer Stelle im Dokument geändert werden.
- Um die Größe festzulegen, ergänzt man `<basefont>` um das Attribut `size=...`
- Soll die meiste Schrift kleiner sein als voreingestellt, wird der Attributwert 2 oder 1 gewählt. Für eine größere Normalschrift stehen die Werte 4 bis 7 zur Verfügung.
- Zur Information: Größenänderungen für Textabschnitte mit dem Tag `<font size=+/-...>` finden dann abhängig von der neuen Normalschriftgröße statt – jedoch insgesamt nie unter 1 oder über 7 hinaus.

77. *Bereiche* (Typ 5)

Eine Grafik (`<img>`) und eine Überschrift (`<h3>`) sollen gemeinsam zentriert ausgerichtet werden.

Homepages an der TU Braunschweig

Vervollständige dazu das angegebene HTML-Fragment.

```
<div align=center>
<img src="y-files.gif"
```

```
height=44 width=240>
<h3>Homepages an der TU
Braunschweig</h3>
</div>
```

(Alternative: `<center>`*)*

Zum Lösen der Aufgabe benötigst du das **Ausrichten**-Attribut (`align`). Neu ist folgendes:

- Man kann einen ganzen Bereich bzw. die Objekte darin `left`, `right`, `center` oder als `justify` ausrichten.
- Die Objekte müssen zwischen `<div>` und `</div>` stehen ("division", „Bereich"), wobei das einleitende Tag um das Attribut erweitert wird.

78. *Trennlinien* (Typ 5)

Definiere eine Trennlinie

1. ohne Schattierung
2. der Dicke 3 Pixel
3. der Breite 75%
4. rechtsbündig ausgerichtet

Vervollständige dazu folgendes Tag.

```
<hr noshade size=3 width=75%
align=right>
```

Die Aufgabe prüft alle Attribute ab, die man ins `<hr>`-Tag stecken kann: **Nichtschattierung** (`noshade`), **Linienhöhe** (`size`), **Breite** (kurze Linie mit `width`) sowie das **Ausrichten** (`align` – nur `left` oder `right`, denn „zentriert" ist voreingestellt).

79. *Liniendefinition* (Typ 5)

Definiere doch mal eine ganz normale Trennlinie.

```
<hr>
```

Mehr als die HTML-**Regeln** mußt du hierfür nicht kennen.
Das ist neu:

- Man kann Abschnitte auf einer Hypertext-Seite durch horizontale Linien trennen.
- Das alleinstehende Linien-Tag heißt `<hr>` (für "horizontal rule" = „waagerechte Linie").

80. *Nichtschattierung* (Typ 5)

Definiere eine Trennlinie ohne Schatteneffekt.

```
<hr noshade>
```

Die Aufgabe erweitert die **Liniendefinition** (`<hr>`).
Das ist neu:

- Normalerweise werden Trennlinien etwas hervorgehoben dargestellt. Es entsteht ein räumlicher Effekt.
- Möchte man aber einfach nur einen harmlosen Strich, so ergänzt man das Linien-Tag um ein Attribut namens `noshade`.

81. *Linienhoehe* (Typ 5)

Wähle zuerst eine Linienhöhe.

```
1       5
| | | | |
```

Nun definiere eine entsprechend dicke Trennlinie.

```
<hr size=  >
```

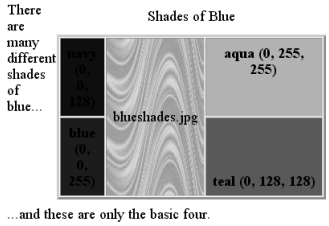
Die Aufgabe erweitert die **Liniendefinition** (`<hr>`).
Das ist neu:

- Normalerweise werden Trennlinien in der Dicke 2 Pixel dargestellt. Höhere Werte machen sie dicker, kleinere dünner.
- Das benötigte Attribut zum Linien-Tag lautet `size=`*Pixelhöhe*

Tabellen

82. *Tabellen* (Typ 5)

Es ist deine Aufgabe, den abgebildeten Hypertext-Entwurf zu implementieren.



Ergänze dazu in folgendem Tabellengerüst noch die Attributnamen bzw. für die Hintergrundfarben die Attributwerte (entweder alle in Hexadezimalschreibweise oder alle als Farbnamen).

```
<table border=3 cellpadding=8
cellspacing=1 width=300
height=200 align=right>

<caption align=top>
Shades of Blue</caption>

<colgroup span=3>

</colgroup>

<tr>

<th valign=top
bgcolor=#000080>
navy (0, 0, 128)</th>
(oder navy)

<td rowspan=2
background="blueshades.jpg"
align=center>blueshades.jpg
</td>

<th valign=top
bgcolor=#00FFFF>
aqua (0, 255, 255)</th>
(oder aqua)

</tr>

<tr>

<th valign=bottom
bgcolor=#0000FF>
blue (0, 0, 255)</th>
(oder blue)

<th valign=bottom
bgcolor=#008080 nowrap>
teal (0, 128, 128)</th>
(oder teal)

</tr>

</table>

</div>

There are many different
shades of blue...

<br clear=all>

...and these are only the
basic four.
```

Zugegeben, die Aufgabe ist ziemlich umfangreich. Wenn du sie bestehst, kannst du aber mit Fug und Recht behaupten, ein Meister des Tabellendefinierens zu sein. Das wird wiederholt: **Tabellenaufbau**, dann **Rahmen/Gitter/Rand** (`border`, `cellspacing` und `cellpadding`), die **Höhe und Breite** (`height` und `width`, `nowrap`), das **Zellenausrichten** mit `align` und `valign`, Zeilen und Spalten **verbinden** (`rowspan` bzw. `colspan`), **Farbtabellen** (`bgcolor` und `background`) sowie das **Beschriften und Ausrichten** mit `caption`, `<br ...>` und `align`.

83. *Tabellenaufbau* (Typ 5)

Gib dem Browser zu folgender Tabelle Spalten der Breiten „einfach“ und „fünffach“ vor – ausgehend von den zu erwartenden Einträgen.

```
<table border>
  <tr>
    <td>1</td>
    <td>VfB</td>
    <td>29</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Eintracht/td>
```

```
    <td>28</td>
  </tr>
</table>
```

Vervollständige das HTML-Stück.

```
<colgroup>
  <col width=1*>
  <col width=5*>
  <col width=1*>
</colgroup>
<table border> ...
```

Es wird nochmal die Deklaration von Tabellen, insbesondere die **Spaltenvordefinition** mit `<colgroup>`, aufgegriffen.
Wie gibt man dem Browser Spalten vor, die unterschiedlich breit werden sollen?

- Für verschieden breite Spalten notiert man keine *Attribute* im Spaltengruppen-Tag. Stattdessen schreibt man in den Gültigkeitsbereich pro Spalte ein alleinstehendes Tag
  `<col width=`*Breite*`>`
- Spaltengrößen können als `x` Pixel, `y%` der Gesamtbreite oder in der Form `z*` („*z*-fach“) geschrieben werden.

84. *Tabellendefinition* (Typ 1)

Welche der Aussagen über Tabellen sind zutreffend?

- Tabellen werden zwischen den Tags `<table>` sowie `</table>` beschrieben.
- `<table border>` deklariert eine Tabelle mit Linien.
  - `<table blind>` leitet eine blinde Tabelle ein.
  - Gitternetzlinien werden mittels `<vline>` und `<hline>` gesetzt.
- Eine Tabellenzelle ist i. a. einer Zeile und einer Spalte zugeordnet.
  - In den Zellen gelten die Textmodifikationen aus der Tabellenumgebung.

Als Einstieg reichen Kenntnisse der grundlegenden **Regeln**.
Ein ganz neues Konzept: Man kann Text, Grafik und alle anderen Gestaltungselemente übersichtlich in Tabellen anordnen. Einerseits kann man so strukturierte Daten darstellen, andererseits der Seite ein geordnetes Erscheinungsbild verleihen.

- Alles, was zu einer Tabelle gehört, steht zwischen den Tags `<table>` und `</table>`.
- Eine Tabelle besteht aus Zeilen und Spalten.
- Der Rahmen der Tabelle und die Zellenbegrenzungen können sichtbar oder unsichtbar sein.
- Sollen die Linien gezeigt werden, ergänzt man im einleitenden Tag das Attribut `border`.
- Tabellen ohne Begrenzungslinien heißen auch „blinde“ Tabellen. Da sie für die Leser nicht unbedingt als solche zu erkennen sind, stellen sie ein nützliches Werkzeug dar, um gestalterische Effekte erzielen.
- Bei allen Vorteilen: Große Tabellen verzögern den Ladevorgang der Seite. Für die HTML-Programmierer ist es ein wenig mühsam, daß sich die Tabelle umschließende Tags meist nicht auf die Zellen auswirken, sondern dort neu definiert werden müssen – z. B. Schriftoperationen und Textauszeichnungen.

85. *ZeilenSpalten* (Typ 5)

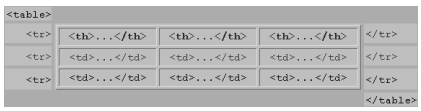
Es ist deine Aufgabe, die abgebildete Tabelle zu implementieren.

Platz	Verein	Punkte
1	VfB	26
2	Eintracht	24

Vervollständige dazu das angegebene HTML-Fragment.

```
<table border>
  <tr>
    <th>Platz</th>
    <th>Verein</th>
    <th>Punkte</th>
  </tr>
  <tr>
    <td>1</td>
    <td>VfB</td>
    <td>26</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Eintracht</td>
    <td>24</td>
  </tr>
</table>
```

Erinnere dich an die **Tabellendefinition** per `<table border>`
Folgendes kommt neu hinzu:



- Die Tabelleninhalte werden zeilenweise definiert.
- Eine Zeile steht zwischen `<tr>` und `</tr>` ("table row", „Tabellenzeile").
- Eine Zeile enthält pro Spalte, die die Tabelle haben soll, ein Feld (Zelle).
- Ein Feld (zumeist jene der ersten Zeile) kann eine (Spalten-) Überschrift enthalten. Eine solche Zelle wird in `<th>...</th>` ("table header" = eine „Tabellenüberschrift") gesetzt. So erwirkt man eine besondere Formatierung des Inhaltes.
- Ein ganz normales Feld wird in `<td>...</td>` ("table data" = „Tabellendaten") geschrieben.
- In Tabellenfeldern kann praktisch jeder Code stehen, der in der Rumpfsektion vorkommen darf.

86. *Spaltenvordefinition* (Typ 5)

Gib dem Browser zu folgender Tabellendefinition die Spaltenzahl vor. Alle Spalten sollen 150 Bildpunkte breit werden.

```
<table border>
  <tr>
    <td>1</td>
    <td>Eintracht/td>
    <td>27</td>
  </tr>
  <tr>
    <td>2</td>
    <td>VfB</td>
    <td>26</td>
  </tr>
</table>
```

Vervollständige das HTML-Stück.

```
<colgroup width=150 span=3>
</colgroup>
<table border> ...
```

Die Aufgabe ist eine Ergänzung zur Definition von **Zeilen und Spalten**. Folgendes ist neu:

- Eine große Tabelle aufzubauen dauert mitunter ziemlich lange. Man kann dem Browser aber eine Information geben, gemäß derer er schon vorab Umrisse und Teile der Tabelle erzeugen kann.
- Zu diesem Zweck definiert man vor der eigentlichen Tabelle ihr grobes Schema: Wie breit soll sie werden? Wie viele Spalten soll sie besitzen? Hierfür gibt es das einleitende und abschließende Tag

`<colgroup`*Attribute*`>`
(„Spaltengruppe“).

- Für den Fall, daß alle Spalten gleich breit werden sollen, ergänzt man als *Attribute*
  `width=`*Breite*
  und
  `span=`*Spaltenzahl*
- Verschieden breite Spalten werden später behandelt.
- Spaltengrößen können als `x` Pixel, `y%` der Gesamtbreite oder in der Form `z*` („*z*-fach“) geschrieben werden.

87. *RahmenGitterRand* (Typ 3)

Spezifiziere die Maße eines Tabellenlayouts nach deinem Geschmack.

1. Wie breit soll der Rahmen der Tabelle sein?
   1 6
   | | | | | |
2. Wieviel Abstand zwischen den Zellen?
   5 10
   | | | | | |
3. Wieviel Abstand innerhalb der Zellen?
   5 10
   | | | | | |

Nun gib die entsprechenden Attribute der einleitenden Tabellendefinition mit den von dir eingestellten Werten an.

`<table border=__ cellspacing=__ cellpadding=__ >`

Kombiniere im Anfangstag der Tabellendefinition die Attribute für

1. **Rahmen** (`border`)
2. **Gitter** (`cellspacing`)
3. **Rand** (`cellpadding`)

89. *Gitter* (Typ 5)

Definiere eine Tabelle mit 8 Pixeln Abstand zwischen den Feldern.

`<table cellspacing=8>`
Tabellenzeilen `</table>`

Die Aufgabe ist eine Erweiterung der **Tabellendefiniton**.
Sie funktioniert ähnlich wie das Definieren von **Rand** innerhalb der Zellen.
Ein neues Attribut für `<table>`:

- Man kann dem Browser vorgeben, wieviel Platz er zwischen den Zellen einer Tabelle lassen soll. Für Tabellen mit Gitter ist dies die Breite der Gitternetzlinien.
- Das Attribut wird als
  `cellspacing=`*Abstand*
  notiert, mit dem *Abstand* als Anzahl der Bildpunkte.

90. *Rand* (Typ 5)

Definiere eine Tabelle mit 5 Pixeln Abstand um die Zellentexte herum.

`<table cellpadding=5>`
Tabellenzeilen `</table>`

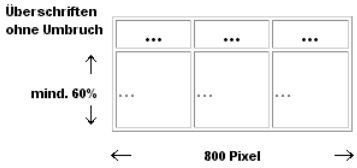
Die Aufgabe ist eine Erweiterung der **Tabellendefiniton**.
Sie funktioniert ähnlich wie das Definieren des Abstandes im **Gitter**.
Ein neues Attribut für `<table>`:

- Man kann dem Browser vorgeben, wieviel Rand er innerhalb der Zellen einer Tabelle um deren Inhalt lassen soll.
- Das Attribut wird als
  `cellpadding=`*Abstand*
  notiert, mit dem *Abstand* als Anzahl der Bildpunkte.

91. *TabellenHoeheBreite* (Typ 5)

Es ist deine Aufgabe, eine ausgesprochen große Tabelle zu implementieren. Finde heraus, welche drei Attri-

bute nötig sind, um die angegebenen Maße zu erhalten.



Vervollständige das Quellcode-Fragment.

```
<table border width=800>
  <tr>
    <th nowrap>...</th>
    <th>...</th>
    <th>...</th>
  </tr>
  <tr>
    <td height=50%>...</td>
    <td>...</td>
    <td>...</td>
  </tr>
</table>
```

Man kann die allgemeinen Attribute für erzwungene **Höhe** (`height`) und **Breite** (`width`) auf die Tabelle selbst oder auf Tabellenfelder anwenden. Es handelt sich dann um Mindestwerte, denn durch große Zelleninhalte, etwa langen Text, der keinen **Zellenumbruch** erfahren soll (`nowrap`), kann sich z. B. die Breite weiter ausdehnen.

93. *Zellenumbruch* (Typ 5)

Definiere ein Tabellenfeld ohne Überschriftformatierung, welches in jedem Fall so breit wird, daß sein Inhalt in eine Zeile paßt.

```
<tr>
  <td nowrap>
  Prof. Dr. rer. nat.
  Hieronymus Schlauch
  </td>
</tr>
```

Gesucht ist ein Attribut zur Erweiterung der Definition von Tabellenfeldern (`<th>`, `<td>`) in **Zeilen und Spalten**.

- Man kann dafür sorgen, daß der Inhalt einer Tabellenzelle nicht umgebrochen, sondern die Zelle so weit verbreitert wird, daß er hineinpaßt.
- Dies geschieht durch ein einfaches Attribut, welches in beiden Tabellenfeld-Tags vorkommen darf: `nowrap`

94. *Zellenausrichtung* (Typ 5)

Implementiere folgende Tabelle. Finde heraus, welche Ausrichtungsattribute ergänzt werden müssen.

Star Trek I	Ein geheimnisvolles Flugobjekt bedroht die Erde.
Star Trek II	Ein alter Widersacher Kirks bedroht die Enterprise.

Vervollständige dazu das angegebene HTML-Fragment.

```
<table border>
 <tr>
  <th align=left valign=top>
  Star Trek I</td>
  <td align=right>Ein
  geheimnisvolles Flugobjekt
  bedroht die Erde.</td>
 </tr>
 <tr>
  <th align=left valign=top>
  Star Trek II</td>
  <td align=right>Ein alter
  Widersacher Kirks bedroht die
  Enterprise.</td>
 </tr>
</table>
```

Text in Tabellenzellen wird normalerweise in der Vertikalen mittig und in der Horizontalen linksbündig ausgerichtet; bei “header”-Feldern horizontal i. a. zentriert. Um die Voreinstellungen zu umgehen, notiert man Attribute: zum **Vertikalausrichten** `valign` und zum horizontalen **Ausrichten** das gewöhnliche `align`, wel-

ches man in so vielen Tags benutzen kann.

95. *Vertikalausrichten* (Typ 5)

Implementiere eine 2×2-Tabelle. Alle Zellen der ersten Zeile sollen am oberen Rand, alle Zellen der zweiten am unteren Rand ausgerichtet sein.

oben	oben wie eben
unten	und wieder unten

Vervollständige dazu das angegebene HTML-Fragment. Es gibt zwei verschiedene Lösungswege, bei denen jeweils Lücken freibleiben.

```
<table border>
  <tr valign=top>
    <td valign=top>oben</td>
    <td valign=top>oben<br>
    wie eben</td>
  </tr>
  <tr valign=bottom>
    <td valign=bottom>unten
    </td>
    <td valign=bottom>und
    wieder<br>unten</td>
  </tr>
</table>
```

(`<br>` = Zeilenumbruch)

(Die Attribute können in die Zeilen, in jedes Feld oder gemischt eingetragen werden.)

Den Umgang mit **Zeilen und Spalten** bist du ja schon gewohnt.
Jetzt werden in Tabellenzeilen und in Tabellenfelder Attribute zum vertikalen Ausrichten eingetragen, was ähnlich wie das gewöhnliche horizontale Ausrichten (`align`) funktioniert.

- Wenn ein Feld größer als sein Inhalt ist, dann wird dieser für gewöhnlich direkt in die Mitte gesetzt.
- Mit dem Attribut `valign=`*Position* ("vertical alignment" = „vertikale Ausrichtung") kann man Text und Grafiken auch am oberen Rand beginnen oder am unteren Rand enden lassen.
- Als Werte der *Position* sind erlaubt: `top` (oben), `bottom` (unten), `middle` (Defaultwert Mitte) oder `baseline` (Ausrichten an einer Grundlinie).
- Die vertikale Ausrichtung kann für ein Feld (`<th>`, `<td>`) oder gleich für eine komplette Zeile (`<tr>`) geltend gemacht werden.

96. *Verbinden* (Typ 5)

Definiere folgende Tabelle mit einem übergroßen Feld, das über zwei Zeilen und drei Spalten reicht.

Ergänze im HTML-Fragment an den richtigen Stellen die zwei fehlenden Tabellenfelddefinitionen.

```
<table border>
  <tr>
    <td>...</td>
    <td>...</td>
    <td>...</td>
    <td>...</td>
  </tr>
  <tr>
    <td>...</td>
    <td rowspan=2 colspan=3>
    ...</td>
  </tr>
  <tr>
    <td>...</td>

    ____________
  </tr>
</table>
```

(alternativ: `<td colspan=3`

`rowspan=2>...</td>`)

Kombiniere die Attribute für **Zeilenverbinden** (`rowspan`) und **Spaltenverbinden** (`colspan`) in einer einzigen `<td>/</td>`-Definition und setzte den Ausdruck an die richtige Stelle. Zähle dann ab, wo noch eine einfache Tabellenfeldefinition (ohne Attribute) fehlt.

97. *Zeilenverbinden* (Typ 5)

Definiere eine Tabelle mit einer Spalte, die zwei Zeilen überspannt.

<table>
<tr><td rowspan="2">Athlon</td><td>833 MHz</td></tr>
<tr><td>866 MHz</td></tr>
</table>

Vervollständige dazu das angegebene HTML-Fragment.

```
<table border>
  <tr>
    <td rowspan=2>Athlon
    </td>
    <td>833 MHz</td>
  </tr>
  <tr>
    <td>866 MHz</td>
  </tr>
</table>
```

Hier kommt eine Erweiterung für das **Zeilen und Spalten**-Konstrukt ins Spiel.
Das Zeilenverbinden verläuft ähnlich wie das **Spaltenverbinden**.
Folgendes ist neu:

- Das Gerüst einer Tabelle muß kein starres, gleichmäßiges Gitter sein. Man kann Felder definieren, die über mehrere Zeilen gehen.
- Um dies zu erreichen, läßt man einfach die `<th>/<td>`-Tags der überdeckten Felder weg.
- Stattdessen notiert man als Attribut im Tag des überdeckenden Feldes, wie viele Zeilen es umfaßt: `rowspan=`*Zeilenzahl*

98. *Spaltenverbinden* (Typ 5)

Definiere eine Tabelle mit einer Zeile, die zwei Spalten überspannt.

<table>
<tr><td colspan="2">Klasse 10</td></tr>
<tr><td>Klasse 11a</td><td>Klasse 11b</td></tr>
</table>

Vervollständige dazu das angegebene HTML-Fragment.

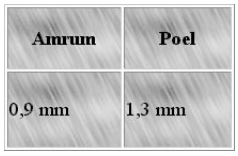
```
<table border>
  <tr>
    <td colspan=2>Klasse 10
    </td>
  </tr>
  <tr>
    <td>Klasse 11a</td>
    <td>Klasse 11b</td>
  </tr>
</table>
```

Hier kommt eine Erweiterung für das **Zeilen und Spalten**-Konstrukt ins Spiel.
Das Spaltenverbinden geschieht ähnlich dem **Zeilenverbinden**.
Folgendes ist neu:

- Das Gerüst einer Tabelle muß kein starres, gleichmäßiges Gitter sein. Man kann Felder definieren, die über mehrere Spalten gehen.
- Um dies zu erreichen, läßt man einfach die `<th>/<td>`-Tags der überdeckten Felder weg.
- Stattdessen notiert man als Attribut im Tag des überdeckenden Feldes, wie viele Spalten es umfaßt: `colspan=`*Spaltenzahl*

99. *Farbtabellen* (Typ 5)

Eine Tabelle soll als Hintergrundbild die Grafik `regen.gif` aufweisen. Leser, die im Browser die Bilder abgeschaltet haben, sollen immerhin einen hellblauen (0, 255, 255) Hintergrund zu sehen bekommen.



Gib das einleitende Tabellen-Tag mit Attributen für Hintergrundgrafik und -farbe (ohne Tabellengröße) an.

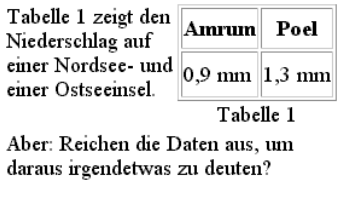
```
<table border background=
"regen.gif" bgcolor=#00FFFF>
```

(alternativ: `bgcolor=aqua`*)*

Erinnere dich an die **Tabellendefinition** und ergänze **Hintergrundbild** (`background`) sowie Hellblau als **Hintergrundfarbe** (`bgcolor`). Die Farbdefinition kann hexadezimal codiert oder als Farbname geschehen.
Neben einem tabellenweiten Hintergrund kann man Farben und Bilder auch für einzelne Zeilen und Felder angeben.

100. *BeschriftungAusrichtung* (5)

Es ist deine Aufgabe, wissenschaftliche Untersuchungen eines meteorologischen Institutes im Internet publik zu machen. Man gibt dir folgenden Entwurf in die Hand und trägt dir auf, besonders die Ausrichtungen und Beschriftungen von Tabellen einzuhalten.



Implementiere die Tabelle, indem du den angegebenen Lückentext vervollständigst.

```
<table border height=80
align=right>
 <caption align=bottom>
 Tabelle 1</caption>
 <tr>
  <th>Amrum</th>
  <th>Poel</th>
 </tr>
 <tr>
  <td>0,9 mm</td>
  <td>1,3 mm</td>
 </tr>
</table>
Tabelle 1 zeigt den
Niederschlag auf einer
Nordsee- und einer
Ostseeinsel.
<br clear=all>
Aber: Reichen die Daten aus,
um daraus irgendetwas zu
deuten?
```

(`<table ...>` definiert das Tabellenobjekt.)

Beschriften und Ausrichten von Tabellen ähnelt dem **Beschriften und Umfließen** von Grafiken.
Sind dir viele Tags im Sourcecode unbekannt? Das macht nichts. Konzentriere dich auf die Lücken und wende einfach die **Tabellenüberschrift** (`<caption>`), die **Textfortsetzung** (`<br ...>`) unter Tabellen sowie das **Ausrichten** von Objekten an.
Das an sich in seiner Wirkung recht harmlose Ausrichten gewinnt im Zusammenhang mit Tabellen (und Grafiken) stark an Effekt, weil man damit objektumfließenden Text realisieren kann.

- Notiere die Ausrichtung im einleitenden `<table>`-Tag.
- Das Attribut `align=left` platziert eine Tabelle links und läßt es rechts von dem Text, der im Code auf das Objekt folgt, umfließen.
- Das Attribut `align=right` platziert ein Tabellenobjekt rechts und läßt es links von Text umfließen.

101. *Tabellenueberschrift* (Typ 6)

Es ist immer wieder ein kontrovers diskutiertes Thema, welcher Stil der

bessere ist: Schreibt man Bezeichnungen von Tabellen darüber oder darunter? Zu welcher Fraktion zählst du dich??
„Ich verwende. . . “

Tabellenüberschriften
Tabellenunterschriften

Na schön, und wie macht man das in HTML? Beschrifte eine fiktive Tabelle 2.1 gemäß deiner Vorlieben.

```
<table border>
  <caption align=____>
  Tabelle 2.1</caption>
</table>
```

Tabellenbeschriftungen sind ziemlich einfach. Du mußt nur die grundlegenden **Regeln** für Tags kennen.

- Eine Tabellenbeschriftung muß im HTML-Quellcode irgendwo zwischen einleitendem und abschließendem `<table>`-Tag stehen. Woanders ergibt sich kein Effekt.
- Die Syntax der Beschriftungen lautet:
  `<caption` *Attribut*`>`*Text*
  `</caption>`
- Der *Text* darf auch selbst Tags enthalten. Ansonsten wird bloß die einfache Bezeichnung ausgeschrieben. Eine automatische Nummerngenerierung gibt es in HTML nicht.
- Das *Attribut* `align=top` sorgt für Tabellenüberschriften bzw. `align=bottom` für Tabellenunterschriften.

103. *blindeTabellen* (Typ 5)

Implementiere folgenden Entwurf als blinde Tabelle mit

- zwei Spalten
- der Grafik `map.jpg`
- 10% Rand auf beiden Seiten
- und Hellblau als Hintergrundfarbe (0, 255, 255)

Ergänze dazu den Lückentext.

```
<div align=center>
 <table width=80%>
 <colgroup span=2>
 </colgroup>
 <tr bgcolor=#00FFFF>
  <td><img src="map.jpg"></td>
  <td>Coventry...</td>
 </tr>
 </table>
</div>
```

(Alternativen: `<center>`*; Farbname* `aqua`*)*

Der Seitenausschnitt enthält zwar einige kompliziert erscheinende Effekte, es ist jedoch alles ganz einfach zu erreichen. Als Anwendungen blinder Tabellen kommen vor: **Zweispaltigkeit**, **Text und Grafik** (`<img>`) nebeneinander, **Seitenränder** von zusammen 20% der Fenstergröße sowie zwei **Farbflächen** in einer Tabellenzeile mit gemeinsamer Hintergrundfarbe (`bgcolor`).

104. *Mehrspaltigkeit* (Typ 1)

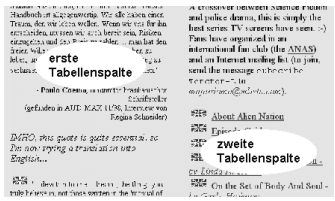
Welche Aussagen über das Definieren von mehrspaltigem Text stimmen?

- Man kann mehrere Textspalten durch das Ausnutzen „blinder“ Tabellen bekommen, die man mittels `<table>` ohne `border`-Attribut deklariert.
  - Ein Nachteil bei der Verwendung von Layout-Tabellen besteht leider darin, daß zusätzliche Angaben über Breite und

Ausrichtung keine (vorhersagbaren) Auswirkungen haben.

- Man erhält eine Seite mit dreispaltigem Text durch folgende Tabellendefinition:
```
<table>
 <tr>
  <td>Spaltentext links</td>
  <td>Spaltentext Mitte</td>
  <td>Spaltentext rechts</td>
 </tr>
</table>
```
  - Angenommen, die obige Definition wäre richtig, dann kann (und sollte) man dem Benutzer z. B. durch diese Vorabangabe unangenehmes Warten zur Zeit des Spaltenaufbauens ersparen:
```
<colgroup>
 <col width=3*>
</colgroup>
```

Zur Lösung der Aufgabe solltest du schon ein einige Kenntnisse im Umgang mit Tabellen erworben haben. Dazu gehört auch die Vorabangabe über den zu erwartenden **Tabellenaufbau** mittels `<colgroup>`.
Wie definiert man mehrspaltigen Hypertext?



- Man benutzt dazu „blinde“ Tabellen; also Tabellen, bei denen der Benutzer die Linien nicht sieht. Solche Tabellen werden häufig zum Erwirken eines Layoutes und nicht zur Darstellung von Daten verwendet.
- Für die Abbildung wurde eine seitengroße blinde Tabelle mit zwei Spalten zu je einem Feld definiert. In beide Zellen wurde dann Text eingetragen.
- Um die Ladezeit einer Tabelle, die sich über die ganze Seite erstreckt, zu minimieren, ist es nötig, dem Browser Vorabinformationen über die Anzahl der Spalten und ihre erwarteten Größen mitzuteilen.
- Nützliche Tips: Wenn du dafür die Attribute kennst, dann gib in der Deklaration der blinden Tabelle bzw. ihrer Felder eine erwünschte Breite an und richte die Spalten vertikal am oberen Rand aus.

105. *TextGrafik* (Typ 1)

Welche Aussagen über das Anordnen von Text und Grafiken stimmen?

- Man kann beides ordentlich nebeneinander darstellen, indem man Text und Grafik in eine „blinde“ Tabellen steckt, welche man mittels `<table>` ohne `border`-Attribut deklariert.
  - Durch folgende Tabellendefinition erhält man eine Grafik und einen Text übereinander:
```
<table>
 <tr>
  <td><img href=
  “foto.jpg”></td>
 </tr>
 <tr>
  <td>Text</td>
 </tr>
</table>
```
- Durch folgende Tabellendefinition erhält man einen Text und eine Grafik nebeneinander:
```
<table>
 <tr>
  <td>Text</td>
  <td><img src=
  “foto.jpg”></td>
```

```
 </tr>
</table>
```

- Angenommen, die obere Definition wäre richtig, dann kann man den Lesern z. B. durch diese Vorabangabe unangenehmes Warten beim Spaltenaufbauen ersparen:
```
<colgroup>
 <col width=1*>
</colgroup>
```

Zur Lösung der Aufgabe solltest du Grafiken mit dem `<img>`-Tag **referenzieren** können sowie über Kenntnisse im Umgang mit Tabellen verfügen. Dazu gehört auch die Vorabangabe über den zu erwartenden **Tabellenaufbau** mittels `<colgroup>`.
Wie kann man Text ordentlich neben Grafiken setzen?

- Man kann dazu „blinde“ Tabellen benutzen; also Tabellen, bei denen der Benutzer die Linien nicht sieht. Derartige Tabellen werden oft zum Erwirken eines Layoutes anstatt zur Darstellung von Daten verwendet.
- Für die Abbildung wurde eine blinde Tabelle mit zwei Spalten zu je einem Feld definiert. Ins erste Feld wurde der Text geschrieben, ins zweite die Grafik eingefügt.
- Man kann natürlich auch Text genau über bzw. unter Grafiken setzen. Dann benötigt man zwei Zeilen und eine Spalte.
- Um die Ladezeit einer Tabelle, die große Grafiken enthält, zu minimieren, ist es u. a. nötig, dem Browser Vorabinformationen über die Anzahl der Spalten und ihre erwarteten Größen mitzuteilen.

106. *Seitenraender* (Typ 3)

Du möchtest deinen Seitentext mit Rändern links und rechts versehen. Wieviel Prozent des Gesamtfensters soll jeder Rand breit sein?

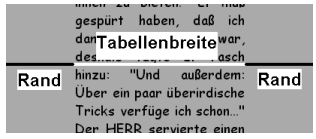
10 30

Ergänze nun das angegebene HTML-Fragment. Erzeuge deine Randbreite, indem du den Text in eine seitlich begrenzte und als Bereich zentrierte blinde Tabelle setzt. Vergiß nicht eine Vorabangabe über die Spaltenzahl.

```
<div align=center>
 <table width=____>
  <colgroup span=1>
  </colgroup>
  <tr>
   <td>
    Seitentext
   </td>
  </tr>
 </table>
</div>
```

(Alternative zu `<div>`*:* `<center>`*)*

Mit Tabellen bist du ja bereits vertraut. Du solltest die **Breite** (`width`) in Prozent erzwingen können sowie die Vorabangabe des zu erwartenden **Tabellenaufbaus** (`<colgroup>`) beherrschen. Die Tabelle wird als **Bereich** (`<div>`) ausgerichtet.
Aber alles der Reihe nach:



- Man kann Ränder rechts und links erzeugen, indem man den Seitentext in eine „blinde“ Tabelle packt; also eine Tabelle,

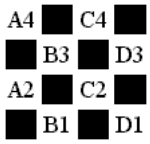
bei welcher der Benutzer die Linien nicht sieht. Solche Tabellen werden oft zum Erwirken eines Layoutes anstatt zur Darstellung von Daten verwendet.

1. Definiere also eine blinde Tabelle mit einem Feld.
2. Beschränke ihre Breite auf soviel Prozent, wie sie im Fenster einnehmen soll: $Breite = 100\% - 2 \times Rand$
3. Damit auf beiden Seiten gleich viel Rand entsteht, setze die Tabelle in einen zentrierten Bereich.

- Um die Ladezeit einer Tabelle, die sich horizontal über die ganze Seite erstreckt, zu minimieren, ist es nötig, dem Browser mitzuteilen: „Es kommt nur eine Spalte.“

107. *Farbflaechen* (Typ 5)

Wie erzeugt man ein Schachbrett der Größe 2×2? (das linke untere Viertel der Abbildung)



Ergänze im Lückentext schwarze und weiße Hintergrundfarben für die Felder einer blinden Tabelle. Trage die Koordinaten der weißen Felder ein. Gib dem Browser die Vorabinformation über Anzahl und relative Größe der Spalten.

```
<table>
 <colgroup width=50% span=2>
 </colgroup>
 <tr>
  <td bgcolor=#000000>A2</td>
  <td bgcolor=#FFFFFF> 
  </td>
 <tr>
  <td bgcolor=#FFFFFF> 
  </td>
  <td bgcolor=#000000>B1</td>
 </tr>
</table>
```

(Alternative: Farbnamen `black` *und* `white`*)*

Tabellen mit Zeilen und Spalten zu definieren dürfte für dich ja kein Problem mehr sein. Offenbar kennst du auch schon die Vorabangabe des zu erwartenden **Tabellenaufbaus** über `<colgroup>`.
Hast du dir schonmal überlegt, was man mit Tabellen so für Effekte erzielen kann? Hier geht es darum, wie man Hintergrundmuster erzeugt.

- Man kann den Seitentext mit wechselnden Farbflächen unterlegen, indem man ihn auf die Zellen einer „blinden“ Tabelle verteilt, deren Hintergrund jeweils eingefärbt ist. Der Benutzer sieht die Hintergrundlinien nicht. Ihm oder ihr fällt eventuell nicht mal auf, daß das Hintergrundlayout in Wirklichkeit eine Tabelle ist.
- Hintergrundfarben können für die gesamte Tabelle, Zeilen davon oder für einzelne Felder definiert werden.
- Ein Feldhintergrund wird u. U. nur dann angezeigt, wenn sich etwas in der Zelle befindet. Will man leere Flächen, muß man zumindest Leerzeichen ` ` notieren.
- Vorinformationen über Anzahl und relative Größe der zu erwartenden Spalten helfen dem Browser, große Tabellen stückweise am Bildschirm aufzubauen.

108. *nichtvisuell* (Typ 5)

Ergänze in der Tabellendefinition zur Abbildung die rotgeschriebenen Zusatzinformationen für Sprach-Browser.

Zusammenfassung: "Ergebnisse der Klassenarbeiten"

Name gilt für die Spalte	Vokabeltest gilt für die Spalte Kategorie "Note"	Klausur gilt für die Spalte Kategorie "Note"
Agnes	2 Beschreibung: "im Vokabeltest"	4 Beschreibung: "in der Klausur"
Lars	3 Beschreibung: "im Vokabeltest"	2 Beschreibung: "in der Klausur"

```
<table border summary=
“Ergebnisse der
Klassenarbeiten”>
 <tr>
  <th scope=col>Name</th>
  <th scope=col axis=“Note”>
  Vokabeltest</th>
  <th scope=col axis=“Note”>
  Klausur</th>
 </tr>
 <tr>
  <td>Agnes</td>
  <td abbr=“im Vokabeltest”>2
  </td>
  <td abbr=“in der Klausur”>4
  </td>
 </tr>
 <tr>
  <td>Lars</td>
  <td abbr=“im Vokabeltest”>3
  </td>
  <td abbr=“in der Klausur”>2
  </td>
 </tr>
</table>
```

In der Aufgabe sind nochmal alle Angaben für nichtvisuelle Browser vereint: die **Zellenbescheibung** mittels `abbr`, der Scope bzw. die **Kopfzellenreferenz**, das Schaffen von **Kategorien** per `axis` und die **Zusammenfassung** `summary`

109. *Zellenbeschreibung* (Typ 5)

Ergänze die folgende Tabellendefinition um Kurzbeschreibungen in den unteren Zellen, so daß ein Sprachausgabe-Browser vorlesen könnte:

⟨Sprachsynthese:
„Insel.
Amrum.
Poel.
Niederschlag.
Auf Amrum 0,9 mm.
Auf Poel 1,3 mm.“⟩

```
<table border>
 <tr>
  <th>Insel</th>
  <td>Amrum</td>
  <td>Poel</td>
 </tr>
 <tr>
  <th>Niederschlag</th>
  <td abbr=“auf Amrum”>0,9 mm
  </td>
  <td abbr=“auf Poel”>1,3 mm
  </td>
 </tr>
</table>
```

Die Aufgabe erweitert die Definition von **Zeilen und Spalten**.
Wie bei **Kopfzellenreferenz**, **Kategorien** sowie der **Zusammenfassung** kommen Zusätze für nichtvisuelle Browser zum Einsatz. Die Zellenbeschreibung ist mit dem **Alternativtext** von Grafiken vergleichbar.
Folgendes ist neu:

- Sehbehinderte surfen mit speziellen Sprachausgabe-Browsern, die ihnen Texte auf Internet-Seiten vorlesen.
- Diese können die meisten einfachen Hypertext-Element gut interpretieren – bis auf Tabellen, denn die enthalten viel Information in ihrer Struktur, die intuitiv erkannt werden muß.
- Das Attribut
  `abbr=“`*Beschreibungstext*`”`
  in einer Tabellenzellendefinition liefert nichtvisuellen Browsern eine kurze (“abbreviated”) Beschreibung über den Feldin-

halt, die in gewöhnlichen Programmen nicht angezeigt wird.

110. *Kopfzellenreferenz* (Typ 5)

Ergänze die Tabelle um "Scopes" der Überschriftzeilen, so daß ein Sprachausgabe-Browser vorlesen könnte:

⟨Sprachsynthese:
„Insel.
Insel Amrum.
Insel Poel.
Niederschlag.
Niederschlag 0,9 mm.
Niederschlag 1,3 mm.“⟩

```
<table border>
 <tr>
  <th scope=row>Insel</th>
  <td>Amrum</td>
  <td>Poel</td>
 </tr>
 <tr>
  <th scope=row>Niederschlag
  </th>
  <td>0,9 mm</td>
  <td>1,3 mm</td>
 </tr>
</table>
```

Die Aufgabe erweitert die Definition von **Zeilen und Spalten**.
Wie bei **Zellenbeschreibung**, **Kategorien** sowie der **Zusammenfassung** kommen Zusätze für nichtvisuelle Browser zum Einsatz.
Folgendes ist neu:

- Sehbehinderte surfen mit speziellen Sprachausgabe-Browsern, die ihnen Texte auf Internet-Seiten vorlesen.
- Diese können die meisten einfachen Hypertext-Element gut interpretieren – bis auf Tabellen, denn die enthalten viel Information in ihrer Struktur, die intuitiv erkannt werden muß.
- Das Attribut
  `scope=col/row`
  in einer Header-Definition zeigt dem nichtvisuellen Browser an, daß die Überschrift der Spalte darunter/der Zeile daneben gilt ("scope" = „Gültigkeitsbereich“).
- Weiterführend gibt es noch andere Werte für `scope` sowie eine komplexere Variante der Referenz mit den Attributen `id` und `headers`.

111. *Kategorien* (Typ 5)

Für Sprachausgabe-Browser: Erweitere an den richtigen Stellen die Header-Zeilen um eine Kategorie/Achse „Wetterdaten“.

```
<table border>
 <tr>
  <th ______>Insel</th>
  <td>Amrum</td>
  <td>Poel</td>
 </tr>
 <tr>
  <th axis="Wetterdaten">
  Niederschlag</th>
  <td>0,9 mm</td>
  <td>1,3 mm</td>
 </tr>
 <tr>
  <th axis="Wetterdaten">
  Temperatur</th>
  <td>16,0 Grad</td>
  <td>16,3 Grad</td>
 </tr>
</table>
```

Die Aufgabe erweitert die Definition von **Zeilen und Spalten**.
Wie auch bei **Zellenbeschreibung**, **Kopzellenreferenz** sowie **Zusammenfassung** kommen Angaben für nichtvisuelle Browser zum Einsatz.
Folgendes ist neu:

- Sehbehinderte surfen mit speziellen Sprachausgabe-Browsern,

die ihnen Texte auf Internet-Seiten vorlesen.

- Diese können die meisten einfachen Hypertext-Element gut interpretieren – bis auf Tabellen, denn die enthalten viel Information in ihrer Struktur, die intuitiv erkannt werden muß.
- Das Attribut
  `axis="`*Kategoriename*`"`,
  vorzugsweise in Header-Definitionen, schafft logische Verbindungen zwischen Inhalten, die nicht eindeutig aus der Tabellenstruktur hervorgehen. Gleiche Namen stehen für Tabelleneinheiten einer Kategorie.

112. *Zusammenfassung* (Typ 5)

Gib für Sprachausgabe-Browser folgende Zusammenfassung einer Tabelle an.

> ⟨Sprachsynthese: „Wetterbedingungen auf den Nord- und Ostseeinseln“⟩

```
<table border summary=
"Wetterbedingungen auf den
Nord- und Ostseeinseln">
```

Die Aufgabe erweitert die Definition von **Tabellendefinition**.
Wie auch bei **Zellenbeschreibung**, **Kopzellenreferenz** und **Kategorien** kommen Angaben für nichtvisuelle Browser zum Einsatz.
Folgendes ist neu:

- Sehbehinderte surfen mit speziellen Sprachausgabe-Browsern, die ihnen Texte auf Internet-Seiten vorlesen.
- Diese können die meisten einfachen Hypertext-Element gut interpretieren – bis auf Tabellen, denn die enthalten viel Information in ihrer Struktur, die intuitiv erkannt werden muß.
- Das Attribut
  `summary="`*Text*`"`,
  im `<table>`-Anfangstag hinterläßt Sprach-Browsern eine Zusammenfassung des Inhalts, die von den herkömmlichen Browsern nicht angezeigt wird. Man sollte den ungefähren Sinn einer Tabelle dort ja mit einem Blick erkennen können.

Links

113. *Links* (Typ 2)

Beurteile die folgenden acht Link-Definitionen: Sind sie richtig, enthalten sie einen Schreibfehler (z. B. fehlender Schrägstrich), oder sind sie inhaltlich falsch (z. B. MIME-Typ verwechselt)?

- ```
  <a href="file:///d:/
  briefe/muster.doc" type=
  "application/msword">
  Musterbrief</a>
  ```
  richtig
  Schreibfehler
  inhaltlicher Fehler
- ```
  <a href="gopher:
  gopher.liu.se:70/11"
  LiU Gopher</a>
  ```
  richtig
  Schreibfehler
  inhaltlicher Fehler
- ```
  <a href="ftp://
  ftp.uni-tuebingen.de/
  pub/berichte/r81_90.zip"
  type="application/
  x-tar">
  Archiv 1981-90</a>
  ```
  richtig
  Schreibfehler
  inhaltlicher Fehler
- ```
  <a href=
  "index.html#filme">
  Zur Filmliste</a>
  ```

richtig
Schreibfehler
inhaltlicher Fehler

- `<a href="telnet://www.geocities.com">Login Geocities</a>`

  richtig
  Schreibfehler
  inhaltlicher Fehler

- `<a href="http://www.bigbrother.de.finale.avi" type="video/x-msvideo">Video-Clip Finale</a>`

  richtig
  Schreibfehler
  inhaltlicher Fehler

- `<a href="news:news.pets.fish.tropical?cc=news.pets.fish.goldfish">Sende Nachricht an Fischfreunde</a>`

  richtig
  Schreibfehler
  inhaltlicher Fehler

- `<a href="mailto:d.maciuszek@tu-bs.de?subject=Kritik&body=Doofes Lernprogramm">Negatives Feedback</a>`

  richtig
  Schreibfehler
  inhaltlicher Fehler

Dies hier ist die große Abschlußaufgabe zu Links. Prüfe die HTML-Stücke auf den korrekten Umgang mit dem **Linkaufbau** (`<a>`), **Ankern** (`#`*Ankername*), **Projektlinks** und **anderen Adressen**, **E-Mail**-Links über `mailto`, **Download**-Links (mit MIME-Typ) und schließlich Links zu **eigenen Dateien** (`file`).

114. *Linkaufbau* (Typ 1)

Welche Aussagen über Links sind zutreffend?

- ○ Ein Link ist z. B.
  `<a href="http://www.fischzucht.de"></a>`
- Ein Link ist z. B.
  `<a href="filme/f.html">Filme mit F</a>`
- Die Referenz und die Bezeichnung am Zielort müssen identisch sein, insbesondere auch in der Groß- und Kleinschreibung übereinstimmen.
- Durch Links verknüpft man die Informationen in einer Website und verbindet diese mit anderen Projekten im Internet.
- ○ HTML ist speziell auf Hypertext im World Wide Web ausgerichtet. Zur Realisierung von Hypermedia-Projekten, die von CD-ROM laufen, ist die Sprache weniger bis gar nicht geeignet.

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☺	☺
Dokument		☺	☺	☺	☺
Datei		☺	☺	☺	☺
Verzeichnis			☺	☺	☺
E-Mail				☺	

Dies hier ist eine grundlegende Aufgabe. Du brauchst erstmal nur die einfachen **Regeln** zu kennen.
Jetzt machen wir aus Text Hypertext!

- Im Hypertext sind Informationen miteinander „verlinkt“. Eine verlinkte Information kann aufgerufen werden, indem der entsprechende Link angeklickt wird.
- Die obige Tabelle zeigt, auf welche Objekte man Links setzen darf und an was für Orten sich diese befinden können.
- Die HTML-Syntax eines Links lautet:
  `<a href="`*Zielort*`">`
  Anzeige`</a>`

(“anchor” = „Anker“, “hyper reference” = „Hyper-Verweis“)

- Die *Zielorte* werden über Bezeichner, Datei- oder Pfadnamen angesprochen – auf Groß- und Kleinschreibung achten!
- Die *Anzeige* ist der Text oder die Grafik, die mit dem Link unterlegt und damit anklickbar gemacht wird. Sie wird zumeist unterstrichen und andersfarbig dargestellt.

115. *Anker* (Typ 5)

Definiere an der Textstelle „Literaturverzeichnis D“ einen Bezugspunkt mit dem Bezeichner `literatur_d`
`<a name="literatur_d">`
`Literaturverzeichnis D</a>`

Erzeuge im zweiten Schritt einen seiteninternen Link „Zum Literaturverzeichnis D“ dorthin.
`<a href="#literatur_d">`
`Zum Literaturverzeichnis D</a>`

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☺	☺
Dokument		☺	☺	☺	☺
Datei		☺	☺	☺	☺
Verzeichnis			☺	☺	☺
E-Mail				☺	

Die Aufgabe behandelt einen Typ des **Linkaufbaus**.

- Man kann einen Link zu einer anderen Textstelle (oder Grafik) auf derselben Seite definieren, so daß durch einen einfachen Klick dorthin gesprungen wird.
- Dazu setzt man zunachst mal einen „Anker“ an die Stelle und gibt diesem eine eindeutige Bezeichnung:
  `<a name="`*Ankername*`">`
  Textstelle`</a>`
- Ein Link zu dem Verankerungspunkt sieht dann so aus:
  `<a href="#`*Ankername*`">`
  Anzeige`</a>`
  Das #-Zeichen vor dem Namen weist den Bezeichner als Ankerpunkt aus – nicht vergessen!

116. *Projektlinks* (Typ 1)

Welche lokalen Linkdefinitionen sind korrekt?

- ○ `<a href="texte.html/#the_river">The River</a>`
- ○ `<a href=suchliste.html>Suchliste</a>`
- • `<a href="privat/gedichte.html#fernweh">Fernweh</a>`
- • `<a href="#Anfang">Zum Seitenanfang</a>`
- • `<a href="archiv/privat">Privatarchiv</a>`
- ○ `<a href="faelle\betrug29.htm">Schneider gegen Schulze</a>`

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☺	☺
Dokument		☺	☺	☺	☺
Datei		☺	☺	☺	☺
Verzeichnis			☺	☺	☺
E-Mail				☺	

Die Aufgabe dreht sich ganz um verzeichnisübergreifende Links, bis hin zu **anderen Ankern**.
Was noch nicht erwähnt wurde:

- Gibt man in einer Linkdefinition nur den Pfad ohne einen Dateinamen an, so zeigt der Browser das Directory am Ende des Pfades mit allen freigegebenen Dateien an (vorausgesetzt, daß keine „Defaultdatei“ existiert).
- Ein Link zu einem Verzeichnis sieht also folgendermaßen aus:
  `<a href="`*Pfad*`">`*Anzeige*
  `</a>`

117. *gleichesVerzeichnis* (Typ 5)

Unterlege „Bericht 1999“ mit einem Link zum Dokument `r99.html`, das sich im selben Verzeichnis wie die aktuelle Datei befindet.

```
<a href="r99.html">
Bericht 1999</a>
```

Die Aufgabe behandelt eine Ausprägung des **Linkaufbaus**.

- Ein Verweis auf ein Hypertext-Dokument, das (auch auf dem Server) im gleichen Ordner wie das verweisende liegt, sieht relativ einfach aus, da kein Pfad angegeben werden muß.
- Zwischen den Anführungsstrichen steht dann der Name der HTML-Datei:
  `<a href="`*Dateiname*`">`
  Anzeige`</a>`
- Klickt der Leser auf den Linktext *Anzeige*, wird das referenzierte Dokument in den Browser geladen.

118. *anderesVerzeichnis* (Typ 5)

Unterlege „Filmliste“ mit einem Link zum Dokument `liste.html`, das sich im Unterverzeichnis `filme` befindet.

```
<a href="filme/liste.html">
Filmliste</a>
```

Das Verweisen auf Dokumente in anderen Verzeichnissen funktioniert wenig anders als ein Link auf eine Datei im **gleichen Verzeichnis**.

- Es muß bloß zusätzlich ein Pfad vom aktuellen Verzeichnis aus angegeben werden:
  `<a href="`*Pfad/Dateiname*`">`
  Anzeige`</a>`
- Achtung: Die Notation ist die von UNIX, also mit Vorwärts-Slashes zwischen Directories!

119. *andererAnker* (Typ 5)

Wir befinden uns in der Datei `welcome.html` des Hauptverzeichnisses. In einem Unterverzeichnis mit dem Pfad `stories/fruehe` liegt das Dokument `der_alptraum.html` In diesem ist ein Anker deklariert:

```
<a name="epilog">Epilog</a>
```

Unterlege den Satzanfang „Nach dem Aufwachen“ mit einem Link zum definierten Ankerpunkt.

```
<a href="stories/fruehe/
der_apltraum.html#epilog">
Nach dem Aufwachen</a>
```

Natürlich kann man auch zu einem **Anker** in einer anderen Datei und sogar in einem **anderen Verzeichnis** springen.

- Ein Link zu dem Verankerungspunkt sieht dann so aus:
  `<a href=`
  `"`*Pfad/Datei*`#`*Ankername*`">`
  Anzeige`</a>`
- Das #-Zeichen mit dem Ankernamen steht also direkt hinter der HTML-Datei, in welcher er definiert ist.

120. *andereAdressen* (Typ 5)

Implementiere einen Text mit Internet-Links.

Sie erreichen den Verein unter:
www.fischzucht.de
gopher.fischzucht.de:70/7
ftp.fischzucht.de/pub/doc/
Rechnerzugang:
guppy.fischzucht.de
Lesen Sie auch:
news.pets.fish.tropical

Ergänze einfach den Lückentext.

```
Sie erreichen den Verein
unter:<br>

<a href="http://
www.fischzucht.de">
www.fischzucht.de</a><br>

<a href="gopher://
gopher.fischzucht.de:70/7">
gopher.fischzucht.de:70/7</a>
<br>

<a href="ftp://
ftp.fischzucht.de/pub/doc/">
ftp.fischzucht.de/pub/doc/</a>
<br>

Rechnerzugang:<br>

<a href="telnet://
guppy.fischzucht.de">
guppy.fischzucht.de</a><br>

Lesen Sie auch:<br>

<a href="news:
news.pets.fish.tropical">
news.pets.fish.tropical</a>
```

(
 = Zeilenumbruch)

Die Aufgabe übt das Definieren von Links zu fremden Rechnern im Internet. Erinnere dich, wie man Server für **WWW**, **FTP**, **Gopher**, **Telnet** und **Newsgroups** adressiert. Welche Protokollnamen müssen vor den Zielorten stehen?

121. *WWW* (Typ 5)

Mache aus dem Wort „Internet-Seminar“ einen Link zum Hypertext-Dokument

```
www.tu-bs.de/~wedelman/
internet.htm
```

Fülle dazu die Lücken aus.

```
<a href="http://
www.tu-bs.de/~wedelman/
internet.htm">
Internet-Seminar</a>
```

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☹	☺
Dokument		☺	☺	☹	☺
Datei		☺	☺	☹	☺
Verzeichnis			☺	☹	☺
E-Mail				☺	

WWW-Links sehen ähnlich aus wie **FTP**- oder **Gopher**-Links.
Der *Zielort* im **Linkaufbau** kann eine WWW-Adresse sein.
Eine solche “URL” darf ein **anderes Verzeichnis** und einen **Anker** enthalten.

- Das World Wide Web ist der am meisten benutzte Teil des Internet. Hier liegen die Hypertext-Dokumente. Deine externen Links werden zum größten Teil ins WWW zeigen.
- Nenne als Ziel die exakte URL mit allen Großbuchstaben und Sonderzeichen sowie einem vorangestellten `http://` (“hypertext transfer protocol”).
- Ein WWW-Link mit allen erlaubten Elementen sieht dann so aus:
  `<a href=`
  “`http://`*Domain*`/`*Pfad*`/`
  Datei`#`*Ankername*”`>`*Anzeige*
  `</a>`
  Es kann aber auch einfach nur die Domain bzw. ein Verzeichnis verlinkt werden. Anker sind eher selten.

122. *FTP* (Typ 5)

Mache aus dem Textstück „Alien Nation FTP“ einen Link zum FTP-Verzeichnis

`ftp.demon.co.uk/pub/doc/`
`alien_nation/`

Fülle dazu die Lücken aus.
`<a href="ftp://`
`ftp.demon.co.uk/pub/doc/`
`alien_nation/">`
`Alien Nation FTP</a>`

Der *Zielort* im **Linkaufbau** kann ein FTP-Verzeichnis oder eine von dort downloadbare Datei sein.
FTP-Links sind ähnlich zu **WWW**- und **Gopher**-Links.

- Auf FTP-Servern werden Benutzern Dateien zur Übertragung auf den eigenen Rechner bereitgestellt.
- Nenne als Ziel die exakte FTP-Anschrift unter Beachtung der Groß-/Kleinschreibung und aller Schrägstriche, mit vorangestelltem `ftp://` ("file transfer protocol").
- Ein kompletter FTP-Link sieht dann so aus:
  `<a href=`
  "`ftp://`*Domain/Pfad/Datei*"`>`*Anzeige*`</a>`
  Die Datei kann wegfallen, dann werden alle Files im Directory angezeigt.

123. *Gopher* (Typ 5)

Mache aus dem Text „University of Delaware“ einen Link zum Gopher-Server `gopher.udel.edu:70/1`

Fülle dazu die Lücken aus.
`<a href="gopher://`
`gopher.udel.edu:70/1">`
`University of Delaware</a>`

Der *Zielort* im **Linkaufbau** kann ein Gopher-Verzeichnis bzw. eine Datei dort sein.
Gopher-Links sehen ähnlich aus wie **WWW**- oder **FTP**-Links.

- Gopher ist ein früher Vorgänger des WWW. Noch existieren einige wenige Gopher-Verzeichnisse, wo man stöbern und Dokumente lesen kann.
- Notiere als Ziel die exakte Bezeichnung des Servers mit vorangestelltem `gopher://`
- Ein Gopher-Link sieht dann so aus:
  `<a href=`
  "`gopher://`*Gopheradressse*"`>`
  Anzeige`</a>`

124. *Telnet* (Typ 5)

Mache aus dem Wort „Wunderland-MUD“ einen Telnet-Verweis auf den Rechner `wl.mud.de:4711`

Fülle dazu die Lücken aus.
`<a href="telnet://`
`wl.mud.de:4711">`
`Wunderland-MUD</a>`

Der *Zielort* im **Linkaufbau** kann ein Telnet-Rechner sein.
Eine Telnet-Adresse wird ähnlich wie eine WWW-Domain (**anderes Verzeichnis**) gelinkt.

- Mit Telnet-Programmen steuert man die Adressen von Rechnern an, um sich dort einzuloggen und Programme zu starten.

Für gewöhnlich benötigt man als User eine Zugangsberechtigung.

- Notiere als Ziel der Telnet-Verknüpfung die exakte Rechneranschrift mit `telnet://` davor.
- Das sieht dann so aus:
  `<a href=“telnet://`*Rechner*`”>`
  Anzeige`</a>`

125. *Newsgroups* (Typ 5)

Mache aus dem Wort „Tauschangebote“ einen Link auf die Newsgroup `alt.video.tape-trading`

Fülle dazu die Lücken aus.

```
<a href=“news:
alt.video.tape-trading”>
Tauschangebote</a>
```

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☺	☺
Dokument		☺	☺	☺	☺
Datei		☺	☺	☺	☺
Verzeichnis			☺	☺	☺
E-Mail				☺	

Der *Zielort* im **Linkaufbau** kann eine Newsgroup sein.
Newsgroup-Links sehen ähnlich aus wie die zu **Mailadressen**.

- In einer Newsgroup kann man aktuelle Beiträge von Internet-Usern zu einem bestimten Thema lesen. Newsbeiträge sind E-Mails sehr ähnlich.
- Notiere als Ziel der Verknüpfung die exakte Newsgroup-Bezeichnung mit vorangestelltem `news:`
- Das sieht dann so aus:
  `<a href=“news:`*Newsgroup*`”>`
  Anzeige`</a>`

126. *EMail* (Typ 6)

Mal angenommen, du wärst unzufrieden mit der Regierung und würdest auf einer eigenen Beschwerde-Website `www.beschwerde.org` deinem Ärger Luft machen. Den Lesern bietest du den Service eines automatischen Beschwerdegenerators in Form eines E-Mail-Links an.

Überlege dir zuerst, an wen die Beschwerden gerichtet werden sollen.
„Die Mails gehen an...“

```
info@spd.de
forum@gruene.de
```

Welches Subject und welchen Inhalt sollen die automatischen Beschwerde-Mails haben?
„Ich dachte da an...“

```
Steuern – Die Steuern sind zu hoch.
Arbeit – Ich will Arbeit!
```

Gib nun einen Link „Beschweren“ an, der Mails erzeugt, in denen die obigen Informationen bereits voreingestellt sind. Eine BCC-Kopie der gesendeten Mails soll jeweils
`webmaster@beschwerde.org`
erhalten. Vervollständige das HTML-Fragment.

```
<a href=“mailto:_____
?bcc=webmaster@beschwerde.org
&subject=_____
&body=_____”>
Beschweren</a>
```

Kombiniere die E-Mail-Vorgaben für **Empfänger** (`cc`/`bcc`), **Subject** und **Body**.

- Man kann alle drei Angaben in einem Link unterbringen.
- Sie werden einfach hintereinandergeschreiben und durch Und-Zeichen `&` getrennt.

127. *Mailadresse* (Typ 5)

Unterlege die Wörter „Meine Brieffreundin“ mit der Mailadresse
`p.vainamoinen@gmx.fi`

Fülle dazu die Lücken aus.

```
<a href="mailto:
p.vainamoinen@gmx.fi">
Meine Brieffreundin</a>
```

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☺	☺
Dokument		☺	☺	☺	☺
Datei		☺	☺	☺	☺
Verzeichnis			☺	☺	☺
E-Mail				☺	

Die Aufgabe behandelt eine Ausprägung des **Linkaufbaus**.
Der Link sieht ähnlich aus wie der für **Newsgroups**, dient aber zum Verschicken von Mails, nicht zum Einlesen von Nachrichten.

- Mit einem Klick auf den Mail-Link wird eine E-Mail versandt bzw. das Mailprogramm mit einer neuen Nachricht an die verlinkte Adresse geöffnet.
- Notiere als Ziel die exakte E-Mail-Anschrift des Empfängers (in der Regel deine eigene) mit vorangestelltem `mailto:`
- Das sieht dann so aus:
  `<a href="mailto:user@domain.land">`*Anzeige*`</a>`
- Es macht Sinn, als *Anzeige* den Empfängernamen oder die E-Mail-Adresse selbst zu wählen.

128. *Empfaenger* (Typ 5)

Unterlege das Wort „Klaus“ mit folgenden Mailadressen.

An: `klaus@gmx.net`

CC: `klaus@web.de`

CC: `klaus@excite.com`

Fülle dazu die Lücken aus.

```
<a href="mailto:
klaus@gmx.net?cc=klaus@web.de;
klaus@excite.com">
Klaus</a>
```

Die Aufgabe erweitert die Möglichkeiten des Referenzierens von **Mailadressen**.
Das Auflisten von mehreren Empfängern funktioniert ähnlich wie die Angabe von **Subject** und **Body**.

- Man kann über einen Link E-Mails an mehrere Adressen verschicken lassen.
- Dazu notiert man direkt hinter der ersten Adresse ein Fragezeichen `?` mit angehängten CC- oder BCC-Adressen (das sind „Durchschläge“ und „versteckte Durchschläge“).
- CC-Adressen beginnen mit `cc=` und werden durch Semikolons `;` getrennt; BCC-Adressen analog mit `bcc=`
- Das sieht dann so aus:
  `mailto:user@domain.land?cc=user@domain.land; user@domain.land`

129. *Subject* (Typ 5)

Unterlege das Wort „Klaus“ mit der E-Mailadresse `klaus@gmx.net` sowie der vorgegebenen Betreffzeile „Frage zum Artikel“.

Fülle dazu die Lücken aus.

```
<a href="mailto:
klaus@gmx.net?subject=
Frage zum Artikel">Klaus</a>
```

Die Aufgabe erweitert die Möglichkeiten des Referenzierens von **Mailadressen**.
Das Vorgeben eines Betreffs funktioniert ähnlich wie das Angeben mehrerer **Empfänger** und eines **Body**.

- Man kann für einen Mail-Link nicht nur die Adresse, sondern auch schon die Betreffzeile vorgeben!

- Dazu notiert man direkt hinter der Adresse ein Fragezeichen `?` mit angehängtem Betrefftext.
- Der Text beginnt mit `subject=` und darf Leerstellen enthalten.
- Das sieht dann so aus:
  `mailto:user@domain.land?`
  `subject=Text Text Text`

130. *Body* (Typ 5)

Unterlege das Wort „Klaus“ mit der E-Mailadresse `klaus@gmx.net` sowie der vorgegebenen Nachricht

> „Habe deinen Artikel gelesen“

Fülle dazu die Lücken aus.

```
<a href="mailto:
klaus@gmx.net?body=
Habe deinen Artikel gelesen">
Klaus</a>
```

Die Aufgabe erweitert die Möglichkeiten des Referenzierens von **Mailadressen**.
Das Vorgeben des Nachrichteninhaltes funktioniert ähnlich wie das Angeben mehrerer **Empfänger** und eines **Subject**.

- Man kann für einen Mail-Link nicht nur die Adresse, sondern auch schon den Inhalt vorgeben!
- Dazu notiert man direkt hinter der Adresse ein Fragezeichen `?` mit dem angehängten Text.
- Er beginnt mit `body=` und darf Leerstellen enthalten.
- Das sieht dann so aus:
  `mailto:user@domain.land?`
  `body=Text Text Text`

131. *Download* (Typ 5)

Versuche, die folgenden drei Definitionen mit Download-Links zu vervollständigen. Was für Dateiendungen kannst du aus den MIME-Typen schließen? Welche Übertragungsprotokolle werden benutzt?

Fall 1:

```
<a href="formulare/
bestellung.doc"
type="application/msword">
Bestellformular</a>
```

Fall 2:

```
<a href="ftp://ftp.pcshop.de/
pub/support/drivers.tar"
type="application/x-tar">
Treiber-Archiv</a>
```

Fall 3:

```
<a href="http://
www.pcshop.de/multimedia/
cebit.avi"
type="video/x-msvideo">
CEBIT-Video</a>
```

Dies hier knüpft an die Aufgabe über **Dateien** an.

- Es gibt furchtbar viele Sorten von Dateien, die der Browser nicht unmittelbar anzeigt. Sie werden beim Klicken auf den Link heruntergeladen oder an eine andere Anwendung weitergereicht. Um Dateien zu kategorisieren, wurde der “MIME”-Standard (= “Multipurpose Internet Mail Extensions”) eingeführt.
- Der MIME-Typ zeigt Browsern an, mit was für Dateien sie es bei einem Link zu tun bekommen. An der Dateiendung ist das nicht immer eindeutig festzumachen. Der Schluß von einem MIME-Typ auf die Dateisorte – und damit die Dateiendung oder zumindest eine feste Anzahl – ist dagegen klar.
- MIME-Typen der Aufgabe:
  - `application/x-tar`: TAR-Archiv `.tar`

  - `application/msword`: MS Word `.doc`
  - `video/x-msvideo`: Windows-Film `.avi`
- Der Typ kommt als ein Attribut `type=`*MIME-Typ* ins einleitende `<a>`-Tag.

132. *Dateien* (Typ 2)

Welche Links führen im Browser zum Laden eines neuen Dokuments, welche starten in der Regel einen Download-Dialog, und welche sind irgendwie falsch?

- `<a href="www.wwf.org/german/tiere_a.html">Bedrohte Tiere A</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href="referate/pandabaer.doc">Panda: Vom Aussterben bedroht</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href="ftp://ftp.wwf.org/public/README.txt">README</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href="http://www.greenpeace.org/index.htm">Home</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href=ftp://ftp.muzix.net/Ärzte/Ich_ess_Blumen.mp3>Ich ess' Blumen</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href="http://www.wwf.org/german/bilder/panda.jpg">panda.jpg (5K)</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href="logos/wwf.gif">WWF-Panda</a>`

Dokument-Link
Download-Link
keins von beidem

- `<a href="ftp://ftp.wwf.org/public/animals_all.zip">animals archive</a>`

Dokument-Link
Download-Link
keins von beidem

Achte zunächst darauf, ob Links in **andere Verzeichnisse**, das **WWW** oder auf **FTP**-Server wirklich korrekt definiert sind.
Wenn ja, dann sieh dir die Endungen der Dateien genauer an. Nicht alle Dateitypen können vom Browser als Dokumente dargestellt werden. Der Surfer wird dann gefragt, ob er/sie die Datei downloaden möchte.

- Anzeigbar sind auf jeden Fall Hypertexte (`.html`), der reine ASCII-Text (`.txt`) sowie Bilder in Internet-typischen Formaten (`.gif` und `.jpg`).
- Viele andere Dateien gehören nicht in den Zuständigkeitsbereich eines Web-Browsers. Das sind vor allem solche für spezielle Anwendungen wie "Microsoft Word" (`.doc`) oder gepackte Archive, die gerade für die Weitergabe (per Download) geschaffen wurden (z. B. `.zip`).
- Um die Menge der anzeigbaren Dokumente zu erweitern, kann man sogenannte "Plug-Ins" im Browser installieren.

133. *eigeneDateien* (Typ 5)

Implementiere einen Link zur Datei `start.htm` auf `a:`

Legen Sie die Diskette ins Laufwerk a: ein. Klicken Sie dann auf START!

Ergänze den Lückentext.

```
Legen Sie die Diskette ins
Laufwerk a: ein. Klicken Sie
dann auf
<a href="file://localhost/a:/
start.htm">START</a>!
```

(alternativ `file:///a:/`)

Links	Seite	gleiches Verzeichnis	anderes Verzeichnis	Internet	Datenträger
Stelle	☺	☺	☺	☺	☺
Dokument		☺	☺	☺	☺
Datei		☺	☺	☺	☺
Verzeichnis			☺	☺	☺
E-Mail				☺	

Der *Zielort* im **Linkaufbau** kann eine Datei auf einem Datenträger sein.

- Man sollte Laufwerke nur mit einem Link ansprechen, wenn es unbedingt sein muß. Selbst Benutzer mit gleichen Betriebssystemen haben unterschiedliche Laufwerksbuchstaben. Lokale Links innerhalb einer Partition haben keine Laufwerksbezeichnung nötig.
- Man spricht ein Laufwerk über `file:///`*Laufwerk* oder `file://localhost/`*Laufwerk* an. Darauf folgt der Pfad bis hin zur Datei, geschrieben mit Vorwärts-Schrägstrichen.
- Das sieht dann so aus:
  `<a href="file:///`*Laufwerk*`/`*Pfad*`/`*Datei*`">`*Anzeige*`</a>`
  Die Datei kann wegfallen, dann werden alle Files im Directory angezeigt.

Grafik

134. *Grafik* (Typ 2)

Untersuche die folgenden Grafik-Referenzierungen: Sind sie richtig, oder liegen Fehler in Tags, in Attributnamen oder in Attributwerten vor?

- ```
  <map name="Webring">
   <area shape=square
   coords="0,0,40,40"
   href="list.html">
  </map>
  <img src="ringlogo.gif"
  usemap="Webring">
  ```
richtig
Tag-Fehler
Attributname-Fehler
Attributwert-Fehler
- ```
  <img src="rates.gif"
  valign=middle>Tarife
  ```
richtig
Tag-Fehler
Attributname-Fehler
Attributwert-Fehler
- ```
  <table><td><tr>
  <img src="sleep.jpg">
  </tr>
  <tr>Sleep Cycles
  </tr></td></table>
  ```
richtig
Tag-Fehler
Attributname-Fehler
Attributwert-Fehler
- ```
  <p><img src="english/
  welcome.gif" width=320
  height=240></p>
  ```
richtig
Tag-Fehler
Attributname-Fehler
Attributwert-Fehler
- ```
  <a href="brocken.html"
  noborder>
   <img src="brocken.jpg"
   align=right>
  </a>
  Den Gipfel erreicht<br
  break=all>Das war am
  13.8.94.
  ```
  ```

richtig
Tag-Fehler
Attributname-Fehler
Attributwert-Fehler

- `<img src="aufbau.jpg" alt="Versuchsaufbau" border=1 widht=171 height=204 hspace=5 vspace=5>`

richtig
Tag-Fehler
Attributname-Fehler
Attributwert-Fehler

(Vom zweiten und dritten Punkt muß nur einer richtig beantwortet sein.)

Dies hier ist die große Abschlußaufgabe zu Grafiken. Prüfe die HTML-Stücke auf den richtigen Umgang mit dem **Einbinden** (`<img>`), der **Breite und Höhe** (`width`, `height`), **Beschriften und Umfließen** von Grafiken, dem **Abstand** durch `hspace` und `vspace`, **Grafiklinks** und, last but here first, **Maps**.

135. *Einbinden* (Typ 5)

Betrachte das Foto `teich.jpg` von einer WWW-Seite. Mit welchem Tag wurde es eingebunden, und was für Attribute wurden ergänzt?

Fülle die Lücken.

`<img src="teich.jpg" border=1 alt="Teich mit Trauerweide">`

Der Autor hat dem Foto einen **Rahmen** (`border`) der Breite 1 Pixel sowie einen **Alternativtext** (`alt`) verpaßt.

136. *Referenzieren* (Typ 5)

Baue das animierte GIF-Bild `blue_ribbon.gif` aus dem Unterverzeichnis `graphix` in ein Hypertext-Dokument ein.

Fülle die entsprechenden Lücken aus.

`<img src="graphix/blue_ribbon.gif">`

Diese Aufgabe bildet die Grundlage für alles, was mit Grafiken in HTML zu tun hat. Es reicht, wenn du vorher die **Regeln** für Tags und Attribute kennst.

Ob Icons, Fotos, Zeichnungen, Diagramme – was wäre das das WWW ohne Grafik? Es ist auch gar nicht schwer, Bilddateien in ein Hypertext-Dokument einzufügen.

- Eine Grafik fürs Internet muß im Format JPEG (`.jpg`) oder GIF (`.gif`) vorliegen. Animierte GIFs oder welche mit durchsichtigem Hintergrund werden im Browser auch genau so dargestellt.
- Liegt die Grafikdatei im selben Verzeichnis wie die aufrufende Seite, dann wird das Bild mit folgendem alleinstehenden Tag referenziert.
  `<img src="`*Grafikdatei*`">`
  ("image source", „Bildquelle")
- Für Bilder, die in einem anderen Ordner untergebracht sind, muß zusätzlich der Pfad angegeben werden. Directories werden (wie in UNIX üblich) mit Vorwärts-Slashes getrennt.
  `<img src="`*Pfad*`/`*Grafikdatei*`">`
- Was man besser nicht tut, ist das Einbinden von Bilddateien, die irgendwo im Internet stehen – das verursacht unnötige Wartezeit und Netzbelastung!
  `<img src="http://`*Domain*`/`*Pfad*`/`*Grafikdatei*`">`

137. *Alternativtext* (Typ 5)

Baue das transparente GIF-Bild `gedichte.gif` aus dem Unterverzeichnis `schriften` in ein Hypertext-Dokument ein und statte es mit dem alternativen Text „Eingang hier“ aus.

Fülle die entsprechenden Lücken.

```
<img src="schriften/
gedichte.gif"
alt="Eingang hier">
```

Die Aufgabe behandelt eine Erweiterung des **Referenzieres** von Grafiken.

- Während die Grafik noch geladen wird, kann man an der Stelle bereits einen kurzen Text anzeigen lassen. Die Beschreibung wird außerdem als “Tooltip” sichtbar, wenn der Mauszeiger über dem fertigen Bild steht.
- Die Angabe eines alternativen Textes erfolgt über ein zusätzliches Attribut im `<img>`-Tag:
  `alt="`*Text*`"`

138. *GrafikBreiteHoehe* (Typ 5)

Binde das Foto `hamburg.jpg` in eine WWW-Seite ein. Ergänze Attribute für die Breite und Höhe des Bildes in Pixeln (Originalgröße) und beschleunige damit den Aufbau der Hypertext-Seite.

Fülle die Lücken.

```
<img src="hamburg.jpg"
width=279 height=190>
```

Man sollte die **Referenzierung** einer Grafik (`<img>`) um die allgemeinen Attribute für die **Breite** (`width`) und **Höhe** (`height`) erweitern, um so die Größe des Bildes in Pixeln festzulegen.
Das hat zwei Vorteile:

- Man kann bei Bedarf die Ausmaße der Grafik ändern, ohne die Datei selbst anzutasten.
- Der Browser erfährt frühzeitig, wieviel Platz das Bild einnehmen wird. So erzeugt er erstmal ein Rechteck der benötigten Größe und baut die Seite zuende auf, bevor er die (eventuell ziemlich goße) Grafikdatei lesen muß.

139. *BeschriftenUmfliessen* (Typ 2)

Gib für jedes HTML-Stück an, welches beschriftete Bild es implementiert.

- ```
  Wingdings ist ein
  <img src="test.gif">
  lustiger Font
  ```
  Tropfen
  Fahne
  Telefon
  weiß nicht
- ```
  <img src="test.gif"
  align=left>
  Wingdings ist ein<br
  clear=all>lustiger Font
  ```
  Tropfen
  Fahne
  Telefon
  weiß nicht
- ```
  <img src="test.gif"
  align=middle>
  Wingdings ist ein
  lustiger Font
  ```
  Tropfen
  Fahne
  Telefon
  weiß nicht
  ```

- `<table><tr><td>`
  `<img src="test.gif"></td>`
  `<td>Wingdings ist ein`
  `</td></tr></table>`
  `lustiger Font`

Tropfen
Fahne
Telefon
weiß nicht

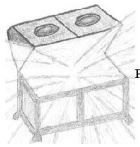
(Von den letzten beiden Punkten darf der Lerner einen mit „weiß nicht“ beantworten.)

Das Beschriften und Umfließen von Bildern funktioniert ähnlich der **Beschriftung und Ausrichtung** bei Tabellen.
Es gibt viele verschiedene Möglichkeiten zur Komposition von Texten und Grafiken.

- Man setzt **Text und Grafik** in einer blinden Tabelle mit zwei Spalten ordentlich nebeneinander.
- Für einen kurzen Kommentar (eine Zeile) erreicht man ähnliches **Grafikbeschriften** über das `align`-Attribut und seine Werte `top` etc.
- Man kann das Bild vom Text (beliebig viele Zeilen) per horizontalem **Ausrichten** der Grafik umfließen lassen. `align` ist dann z. B. `right`, und der Text fließt links um das Bild. Das Fließen wird mittels **Textfortsetzung** (`<br ...>`) unter der Grafik abgebrochen.
- Möchte man Text nur vor und nach der Grafik, dann schreibt man sie in einen `<p>`-**Absatz**.

140. *Grafikbeschriften* (Typ 5)

Betrachte das beschriftete Bild. Welches Attribut fand Verwendung?



Targe den Attributnamen und alles, was sonst noch fehlt, ein.

`<img src="box_open.jpg"`
`align=middle>`Pandora's Box

Gesucht sind hier, neben einem Attribut zur Grafikbeschriftung, die Begrenzungszeichen eines alleinstehenden Tags, wie in den HTML-**Regeln** erläutert.
Dahinter steht ein Text ohne irgendwelche weiteren Tags.

- Selbst ausgefüllt sieht das ganze noch merkwürdig aus. Bewirkt wird aber, daß eine Grafik (`<img>`) und ein kurzer Beschriftungstext (eine Zeile) nebeneinanderstehen.
- Verantwortlich ist das Attribut `align=`*Position* im Grafik-Tag. Es akzeptiert als *Position* `top` (Beschriftung oben), weiterhin `middle` (Mitte), sowie `bottom` (unten).
- ... und noch so einige Variationen: Man hat z. B. für mittige Beschriftungen auch schon die Werte `center`, `absmiddle` und `abscenter` gesehen.

141. *Abstand* (Typ 5)

Binde das Foto `bus.jpg` in eine Web-Seite ein. Ergänze Abstände um das Bild herum.

Fülle die entsprechenden Lücken.

`<img src="bus.jpg"`
`hspace=5 vspace=10>`

Man kann die **Referenzierung** einer Grafik (`<img>`) um Attribute erweitern, die einen gewissen Abstand zwischen Bildrand und umgebenden Elementen garantieren.
Notiere...

- `hspace`=*Pixelzahl*
  ("horizontal space")
  für Abstände rechts und links
- `vspace`=*Pixelzahl*
  ("vertical space")
  für Abstände oben und unten

142. *Grafiklinks* (Typ 5)

Binde das Icon `english.gif` in dein Hypertext-Dokument ein. Wenn man es anklickt, soll `english.html`, die englische Version deiner Homepage, geladen werden. Sorge dafür, daß im Browser kein künstlicher Rahmen um den Link gezeichnet wird.

Fülle die entsprechenden Lücken.

```
<a href="english.gif">
<img src="english.gif"
border=0></a>
```

Verfahre wie sonst beim **Einbinden eines Grafiklinks**, aber füge das Attribut `border=0` für „keinen Rand" hinzu.

143. *EinbindenGrafiklink* (Typ 5)

Binde die Grafik `gedichte.gif` aus dem Unterverzeichnis `schriften` in ein Hypertext-Dokument ein. Wenn man sie anklickt, soll `gedichte.html` geladen werden.

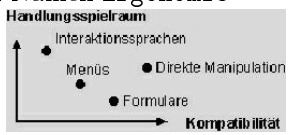
Fülle die entsprechenden Lücken.

```
<a href="gedichte.gif">
<img src="schriften/
gedichte.gif"></a>
```

Schreibe einfach die Grafik-**Referenzierung** (`<img>`) als eine Link-*Anzeige* zwischen die Tags im **Linkaufbau** (`<a>`). Dann ist das Bild mit einem Verweis unterlegt, was im Browser an einem Bilderrahmen in der dateiweiten Linkfarbe erkennbar ist.

144. *Maps* (Typ 5)

Was immer uns dieses Diagramm sagen möchte, es hat in jedem Fall vier wunderschöne Punkte, hinter denen sich bestimmt detailliertere Informationen verstecken lassen. Machen wir doch aus der Grafik eine Map mit dem Namen `Ergonomie`



Fülle die Lücken aus, so daß eine korrekte Map-Definition entsteht.

1. Map-Anker:

```
<map name="Ergonomie">
 <area shape=circle
 coords="37,37,4"
 href="sprachen.html">
 <area shape=circle
 coords="70,70,4"
 href="menues.html">
 <area shape=circle
 coords="100,85,4"
 href="formulare.html">
 <area shape=circle
 coords="130,55,4"
 href="manipulation.html">
</map>
```

2. Map-Referenz:

```
<img src="ergo.jpg"
usemap="#Ergonomie">
```

Erinnere dich daran, wie man Grafiken **referenziert** (`<img>`) und wie die zweiteiligen **Anker**-Definitionen (`<a name=...>`, `<a href=...>`) aussahen.
Es folgt eine Schritt-für-Schritt-Anleitung für das Programmieren von

Maps. Anders als beim **Einbinden von Grafiklinks** sind hier nur Bildausschnitte Links.
Der Benutzer kann also einen Bereich auf einer Grafik anklicken und damit zu einem anderen Ort verzweigen.

- Zuerst werden die aktiven Koordinaten der Map als eine Art Anker deklariert. Nur heißt das Tag hier `<map>` und nicht `<a>`
  `<map name="`*Mapname*`">`
  `<area shape=`*Form*
  `coords="`*Koordinaten*`"`
  `href="`*Zielort*`">`
  ...
  `</map>`
- Jeder Bereich (`<area>`) hat eine *Form* (es gibt `circle`, `rect` und `polygon`), Bild-*Koordinaten* (beim `circle` x, y, r – Mittelpunkt und Radius) plus den gelinkten *Zielort*.
- Im zweiten Schritt fügt man die Map-Grafik ein, auf der die Bereiche liegen.
  `<img src="`*Grafikdatei*`"`
  `usemap="#`*Mapname*`">`
- Der *Mapname* ist der im ersten Schritt definierte – und nicht das `#` vergessen, dann dürfte es funktionieren!

Allgemeines

41. *Hintergrundfarbe** (Typ 5)

Wie sieht das Attribut für dunkelroten (128, 0, 0) Hintergrund aus? (ohne Anführungszeichen)
`bgcolor=#800000`

(Alternative: `maroon`*)*

Hilfreich ist hier die Kenntnise der **Farbdefinition** und der **Regeln** für Attribute.

- Man kann für eine HTML-Seite (`<body>`), Tabellen und Tabellenkomponenten die Hintergrundfarbe ändern.
- Das geht mit dem Attribut
  `bgcolor=`*Farbe*
- *Farbe* darf hexadezimal codiert oder ein Farbname sein.

43. *Hintergrundbild** (Typ 1)

Welche Attribute führen wirklich zu einem Hintergrundbild?

- ○ `background=clouds.gif`
- ○ `background=forest.bmp`
- `background="marble.jpg"`

Die Aufgabe setzt nur die **Regeln** voraus.

- Man kann in Hypertext-Seiten (`<body>`), Tabellen-Tags sowie Tabellenkomponenten-Tags jeweils ein Hintergrundbild angeben. Die Grafik wird dann innerhalb des verfügbaren Platzes endlos oft hintereinanader gehängt.
- Das geht mit dem Attribut
  `background="`*Grafikdatei*`"`
- Die *Grafikdatei* muß ein `.gif`- oder `.jpg`-File sein. Sie kann in einem anderen Verzeichnis stehen; dann ist eine Pfadangabe erforderlich.

48. *Ausrichten** (Typ 1)

Welche Attribute bewirken eine Ausrichtung?

- `align=justify`
- ○ `valign=left`
- ○ `align=centre`

Als Vorkenntnisse reichen die grundlegenden **Regeln**
Es gibt ein paar sehr ähnliche Attribute, z. B. das **Vertikalausrichten** (`valign`) in Tabellen.

- Man kann in sehr viele HTML-Tags, z. B. `<div>`, ein Attribut schreiben, welches das jeweilige Element horizontal positioniert und „bündig“ darstellt.
- Das Ausrichten-Attribut heißt `align=`*Bündigkeit*
- Als Bündigkeit dürften in den meisten Fällen
  - `left` (links),
  - `right` (rechts),
  - `center` (zentriert)
  - und `justify` (Blocksatz)

  ihre Wirkung entfalten.
- Im Zusammenhang mit Tabellen und Grafiken haben `left` und `right` noch den besonderen Effekt, daß der umgebende Text das ausgerichtete Objekt umfließt.

69. *Breite** (Typ 1)

Was sind gültige Breiten?

- `width=55%`
- `width=200`
- ○ `width=height*1.5`

Voraussetzung sind nur die allgemeinen Regeln für Attribute.
Die Angabe für die **Höhe** funktioniert praktisch genauso.

- Man kann viele Objekte dahingehend beeinflussen, wie breit sie in der Anzeige (mindestens) sein sollen.
- Das Attribut lautet `width=`*Größenangabe*
- Als *Größenangabe* ist stets die absolute Pixelgröße erlaubt. In den meisten Fällen geht auch eine Prozentangabe relativ zur derzeitigen Fenstergröße – bei Tabellenspalten relativ zur Tabelle.

88. *Rahmen** (Typ 5)

Definiere einen Tabellenrahmen von 5 Bildpunkten Dicke
`<table border=5>`

Zur Lösung der Aufgabe solltest du die **Tabellendefinition** kennen.

- Tabellen haben in der Regel einen Rahmen; Bilder können einen bekommen. Die Breite der Rahmenlinien kann man festlegen.
- Das zugehörige Attribut ist `border=`*Pixelgröße*
- Die *Pixelgröße* kann bei Grafiken 0 sein, um Linkrahmen zu unterdrücken. Ansonsten wird das Attribut einfach weglassen.

92. *Hoehe** (Typ 1)

Was sind gültige Höhen?

- `height=67%`
- ○ `height=2*`
- `height=50`

Voraussetzung sind nur die allgemeinen Regeln für Attribute.
Die Angabe für die **Breite** funktioniert praktisch genauso.

- Man kann viele Objekte dahingehend beeinflussen, wie hoch sie in der Anzeige (mindestens) sein sollen.
- Das Attribut lautet `height=`*Größenangabe*

- Als *Größenangabe* ist stets die absolute Pixelgröße erlaubt. In den meisten Fällen geht auch eine Prozentangabe relativ zur derzeitigen Fenstergröße – bei Tabellenfeldern relativ zur Tabelle.

102. *Textfortsetzung** (Typ 1)

Welche Ausdrücke unterbrechen umfließenden Text?

- ○ `<br>`
- • `<br clear=all>`
- ○ `<br align=center>`

Voraussetzung sind nur die allgemeinen Regeln für Attribute.
Das Tag ist das zum **Erzwingen eines Umbruchs**, aber mit zusätzlichem Attribut.

- Texte, die Grafiken oder Tabellen umfließen (durch ein `align`-Attribut), können durch einen Befehl mitten im Text unterbrochen werden.
- Es geht dann direkt unter dem Objekt weiter.
- Das Tag lautet
  `<br clear=all>`
  ("break" = „Umbruch")

B.2 Kursdateien

Voraussetzungen

- Java JRE oder JDK Version 1.3
- AdAsTra Version 2.0.7 oder höher

Systemdateien

`domains.dat`
`users.dat`

Basen

`html.ksb`
`grundreg.ksb`
`hdemo.ksb`

Aufgaben

`item001.000`
`...`
`graf144.000`

Kursgrafiken

`allgemei.gif`
`dateiang.gif`
`globalde.gif`
`grafik.gif`
`grundreg.gif`
`links.gif`
`stern.gif`
`tabellen.gif`
`textgest.gif`

Itemgrafiken [1]

`graf006.gif`
`...`
`graf144.gif`

Sound [2] [3]

`ton39a.mid`
`ton39b.mid`
`ton44a.mid`
`ton44b.mid`
`ton44c.mid`
`ton109.wav`
`ton110.wav`
`ton112.wav`

[1] Hintergrundbilder geladen von "AAA-Backgrounds"
[2] MIDI-Musik von "MIDI World"
[3] Sprachausgaben generiert mit `http://www.bell-labs.com/project/tts/german.html`