

Webseiten, HTTP und HTML

Eine Webseite ist eine Textdatei, die ein Webserver an einen Browser schickt. Diese Textdatei enthält Zusatzinformation über Gliederung, Formatierung und nicht-textuelle Elemente der Webseite wie zum Beispiel Bilder. Sie enthält also „mehr als Text“, deswegen spricht man von der Hypertext¹-Markup-Language → HTML. Die HTML-Befehle heißen „Tags“ und unterscheiden sich vom normalen Text dadurch, dass sie in spitze Klammern gesetzt werden und (meistens) den Text umfassen, den sie verändern sollen:

```
<!DOCTYPE html>
<!-- Ein Kommentar -->
<html>
  <head>
    <title>Eine Insel</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Eine Webseite</h1>
    <button>Klick mich und es passiert nichts</button>
  </body>
</html>
```

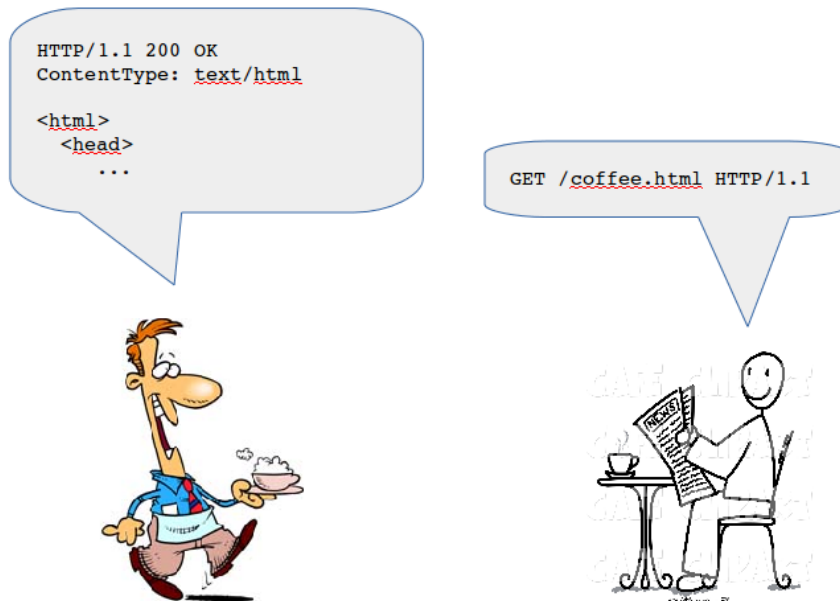
Aus dem Text und den Zusatzinformationen berechnet („rendert“) der Browser die tatsächliche Seite. Tippe die Seite oben einmal ab, speichere sie als beispiel.html und sieh sie Dir im Browser an (Doppelklick).

Bevor wir uns ansehen, was die Teile eine HTML-Webseite bedeuten gehen wir noch einen Schritt tiefer und sehen uns zunächst an, wie sich der Browser und der Webserver unterhalten.

HTTP – Hypertext Transfer Protocol

Die Sprache, die der Client (=Webbrowser) und der Server (=Webserver) miteinander reden nennt man „Hypertext Transfer Protocol“ (HTTP), also „Sprache zur Übertragung von 'Mehr-als-Text'“.

1 von griechisch „hyper“ - „über...hinaus“



Dieses Protocol besteht grundsätzlich auch aus Text und sie wird benötigt, damit der Browser dem Webserver mitteilen kann, welche Seite er überhaupt abrufen möchte. Betrachten wir mal eine typische URL (unified resource locator):

<http://www.coffee-online.com/coffee.html>

Diese besteht aus drei Teilen:

- mit http: teilen wir dem Browser mit, dass wir per http mit dem Server
- www.coffee-online.com auf Port 80 kommunizieren wollen² und von diesem gerne
- die Datei coffee.html geliefert bekämen

Das führt dazu, dass der Browser dem Server www.coffee-online.com folgende Nachricht schickt³:

```
GET /coffee.html HTTP/1.1
```

Beachte die Leerzeile – sie bedeutet „Ende der HTTP-Nachricht“

Darauf antwortet der Webserver, indem er einen HTTP-Header schickt. Danach kommt das eigentliche HTML-Dokument. Dabei erfolgt die Trennung durch eine Leerzeile (=Ende der HTTP-Nachricht). Die (kleinstmögliche) HTTP-Antwort des Server ist dabei

```
HTTP/1.1 200 OK
Content-Type: text/html
```

Der Server sagt damit, dass er die Anfrage bearbeiten kann und dass das

² Port 80 ist die Voreinstellung. Wollen wir einen anderen Port, müssen wir das hinter die Serveradresse schreiben: <http://www.coffee-online:8080/coffee.html>

³ Eigentlich schickt er noch viel mehr Befehle und Informationen. Aber der GET Befehl ist das, wofür es ankommt.

Ergebnisse ein HTML-formatierter Text ist.

Eigene HTTP Abfragen

Das schauen wir uns jetzt einmal in der Praxis an. Man kann nämlich auch zu Fuss HTTP-Abfragen absenden. Dazu verwenden wir das Programm „telnet“. Mit telnet kann man TCP-Verbindungen zu Rechnern herstellen (sofern diese Verbindungen akzeptieren) und dann textbasierte Nachrichten schicken. Wenn wir uns dabei an die Vorgaben des HTTP-Protokolls halten, sollten wir auch eine Antwort bekommen...

Öffne die Verbindung zum Schulserver capo.wara.de auf Port 80:

```
telnet capo.wara.de 80
```

Setze jetzt folgende HTTP Befehle ab:

```
GET /index.html HTTP/1.1  
host:capo.wara.de
```

Beachte die zusätzliche Leerzeile – erst danach reagiert der Server, weil Du ihm damit signalisierst, dass Du fertig bist. Außerdem musst Du Dich mit dem Tippen etwas beeilen – unser Server wartet nicht ewig⁴. Der HTTP-Befehl „host“ muss hier zusätzlich angegeben werden, da die meisten Webserverprogramme auf einer Maschine mehrere Websites mit unterschiedlichen Serveradressen anbieten können (so genannte „Virtual Hosts“). Deswegen müssen wir noch angeben, welchen davon wir haben wollen.

Aufgabe

Der Kollege Meinicke hat unter anderem die Datei

<http://capo.wara.de/~mne/E1IT4T/Beispiel.csv>

hinterlegt. Rufe diese per telnet ab und sieh Dir den Header an.

Ein eigener Server

Jetzt wollen wir einmal sehen, was den ein Browser so an den Webserver schickt. Dazu schreiben wir uns einfach unseren eigenen kleinen Server. Das ist in Java nicht besonders schwer, zumal das HTTP-Protokoll einen recht einfachen Ablauf definiert:

1. der Server wartet
2. der Browser verbindet sich
3. der Browser schickt eine Anfrage (HTTP Request)
4. der Server antwortet
5. der Server trennt die Verbindung
6. der Server wartet auf die nächste Anfrage
7. ...

⁴ Du kannst die Befehle auch in einem Editor vorbereiten und dann in die Konsole reinkopieren. In Linux kopiert man, indem man etwas markiert, das Fenster wechselt und die mittlere Maustaste drückt.

Aufgabe

Das kriegen wir hin ! Zum Anfang lesen wir einfach nur, was reinkommt, schreiben es auf die Konsole und geben eine kurze Antwort:

```

20 public class Minimalserver {
21
22     public static void main(String[] args) throws IOException {
23         ServerSocket serversocket = new ServerSocket(9000); // Punkt 0 - einen Server erstellen der...
24
25         while (true) { // ... für uns wartet (Punkt 1 und 6)
26             Socket connection = serversocket.accept(); // Punkt 2: ein Browser will verbinden, wir akzeptieren
27
28             // Zum Lesen der geschickten Daten (Punkt 3) benutzen wir den guten alten Scanner
29             Scanner vomBrowser = new Scanner(new InputStreamReader(connection.getInputStream()));
30
31             String zeile = vomBrowser.nextLine();
32             while (!zeile.equals("")) { // das Ende der HTTP-Nachricht wird durch eine leere Zeile markiert
33                 System.out.println(zeile);
34                 zeile = vomBrowser.nextLine();
35             }
36
37             // Punkt 4: wir antworten
38             OutputStreamWriter ausgang = new OutputStreamWriter(connection.getOutputStream());
39             ausgang.append("HTTP/1.1 200 OK\n"); // der HTTP-Header...
40             ausgang.append("Content-Type:text/html\n");
41             ausgang.append("\n"); // ...wird mit einer leeren Zeile beendet
42             // Jetzt kommt unsere eigentliche Antwort:
43             ausgang.append("<html><body><h1>Tach auch</h1></body></html>");
44             ausgang.flush(); // Wichtig: mit flush die Antwort abschicken
45             connection.close(); // Punkt 5 - der Server schließt Verbindung zu
46         } // while
47     } // main
48 }
49

```

Schreibe das ab (viel Strg-Space benutzen und die Kommentare nicht mit abschreiben !) und starte es. Verbinde vom Browser auf Deinen Server (<http://localhost:9000>) und schau Dir in der Netbeans-Ausgabe an, was der Browser so alles von sich gibt.

Stoppe den Server und entferne die Zeilen 39 bis 41 durch auskommentieren, so dass die Antwort ohne HTTP-Header gesendet wird – welchen Unterschied macht das im Browser?

Das ist es schon fast ! Es fehlt nur noch, dass das GET-Kommando des Browsers ausgewertet und die angeforderte Datei tatsächlich geliefert wird. Das machen wir jetzt auch . Dabei berücksichtigen wir noch den Fall, dass es den Dateinamen nicht gibt.

Baue Deinen Code so um, dass die Ausgabe aussieht wie im Screenshot unten. Beachte Zeile 32, hier greifen wir die erste Zeile des Requests zur späteren Verwendung ab.

Die Dateien, die Du sehen willst, musst Du ins Projektverzeichnis legen, dorthin, wo auch „manifest.ml“ liegt. Kopiere die Datei beispiel.html vom Anfang dorthin und rufe sie per Browser ab (<http://localhost:9000/beispiel.html>).

```
31 String zeile = vomBrowser.nextLine();
32 String httpcmd = zeile; // die erste Zeile des Browsers MUSS der HTTP-Befehl sein
33 while (!zeile.equals("")) { // das Ende der HTTP-Nachricht wird durch eine leere Zeile markiert
34     System.out.println(zeile);
35     zeile = vomBrowser.nextLine();
36 }
37
38 // Punkt 4: wir antworten
39 OutputStreamWriter ausgang = new OutputStreamWriter(connection.getOutputStream());
40 // Prüfen: hat der Browser ein korrektes HTTP-Kommando der Form
41 // GET dateiname HTTP/1.1 geschickt
42
43 String dateiname = httpcmd.split(" ")[1].substring(1);
44 if (dateiname.isEmpty()) {
45     dateiname = "index.html";
46 }
47 File f = new File(dateiname);
48 if (f.exists()) {
49     ausgang.append("HTTP/1.1 200 OK\n"); // der HTTP-Header...
50     ausgang.append("Content-Type:text/html\n");
51     ausgang.append("\n"); // ...wird mit einer leeren Zeile beendet
52     // Jetzt kommt die Datei:
53     Scanner datei = new Scanner(f);
54     while (datei.hasNext()) {
55         ausgang.append(datei.nextLine());
56     }
57     datei.close();
58 } else { // Datei nicht vorhanden
59     ausgang.append("HTTP/1.1 404 NOT FOUND\nContent-Type:text/html\n"); // der HTTP-Header mal in kurz
60     ausgang.append("<html><body><h1>" + dateiname + " nicht vorhanden</h1></body></html>");
61
62 } // f.exists
63
64 ausgang.flush(); // Wichtig: mit flush die Antwort abschicken
65 connection.close(); // Punkt 5 - der Server schließt Verbindung zu
66 } // while
67 } // main
```

HTML – Hypertext Markup Language

Kommen wir nun zur eigentlichen „Sprache des Internets“, der Hypertext-Auszeichnungssprache. Im folgenden ein etwas längeres Beispiel mit mehr verschiedenen Tags:

```
<!DOCTYPE html>
<!-- Ein Kommentar -->
<html>
  <head>
    <title>Eine Insel</title>
    <meta charset="utf-8">
  </head>
  <body>
    <h1>Eine Webseite</h1>
    <p>Viel BlaBla in einem Absatz.</p>
    <p>Schickes Bild:<br />
    <img src='./bilder/toll.jpg' /><br />
    <a href="toll.html">Hier klicken</a>
  </p>
</body>
</html>
```

Beachte, dass dabei zwischen der öffnenden Klammer und dem Befehl kein Leerzeichen stehen darf. HTML ist ansonsten sehr entspannt im Umgang mit Leerzeichen, aber nicht an dieser Stelle.

Die meisten HTML-Befehle beziehen sich auf einen Text und bestehen aus einem Anfangs- und einem Endtag, die den Text umschließen, auf den sich der Befehl bezieht. Die schließenden Tags erkennt unterscheiden sich von den öffnenden Tags durch einen Slash:

```
<title>Eine Insel</title>
```

kennzeichnet also „Eine Insel“ als den Titel der Webseite (der in der Fensterüberschrift des Browsers angezeigt wird). Manche Tags können nichts umschließen, wie zum Beispiel der Zeilenumbruch `
` oder ein Bild ``. Diese kennzeichnet man mit einem integrierten Slash als selbstabschließend (das Leerzeichen vor dem Slash ist ein Muss!)⁵. Kommentare (einzeilig oder mehrzeilig) werden in HTML mit `<!--` eröffnet und mit `-->` geschlossen.

Jedes Tag kann zusätzlich noch Attribute enthalten, bei einem Bild ist das zum Beispiel der Dateiname (Attribut `src`). Diese Attribute stehen innerhalb der spitzen Klammern des öffnenden Tags. Die Werte für diese Attribute muss man in der Regel in Anführungszeichen setzen, dabei ist es egal, ob man einfache oder doppelte Anführungszeichen nimmt (siehe Beispiel).

Grundbestandteile eines HTML-Dokuments

Das Minimum für eine Seite sind der Dokumententyp, der Kopf (head) und der Körper

⁵ Man kann den Slash in der Praxis auch weglassen, `
` funktioniert auch in jedem Browser. Aber in meinem Unterricht wird der nicht weggelassen!

(body). Der Dokumententyp gibt an, in welcher HTML-Version die Seite geschrieben ist, `<!DOCTYPE html>` kennzeichnet einen Seitenquelltext als HTML 5-Dokument. Im Kopf (HEAD) stehen Information *über* die Seite, also Dinge, die nicht im Hauptfenster des Browser zu sehen sind. Der Body enthält den eigentlich Inhalt (Content) der Webseite.

Als Minimalkopf empfehle ich einen Titel und die Angabe über die Kodierung der Quelltextdatei. Das Beispiel auf Seite 1 zeigt solch einen Head. Durch Angabe der Kodierung kann man Umlaute und bestimmte Sonderzeichen direkt im Quelltext verwenden. Eine auf einem Windowsrechner erstellte Datei ist meistens iso-8859-15 kodiert, unter Linux kommt oft utf-8 zum Einsatz⁶.

HTML zur Strukturierung – und wie wird's bunt?

Ursprünglich wurde HTML entwickelt, damit Wissenschaftler im damals noch jungen Internet Text über Ihre Arbeit austauschen konnten. Es ging also um Texte nicht um hippe Werbeanzeigen oder Onlinespiele. Dementsprechend sind die HTML Kernbefehle dazu gedacht, Text zu strukturieren. Ursprünglich gab es auch Tags, um Text zu formatieren, aber mit steigenden Ansprüchen ans Format wurde dieser Ansatz zu unflexibel und man ist dazu übergegangen, dass man mit HTML nur noch strukturiert und mit einer zweiten Sprache (Cascaded Stylesheets → CSS) die Formatierung erledigt. Das klingt möglicherweise erstmal unnötig kompliziert, ist aber letztendlich mächtiger und programmiererfreundlicher. Natürlich sieht

```
<h1><color="red"><b>Intro</b></color></h1>
```

erstmal schön simpel aus. Aber man stelle sich vor, es gäbe die roten, fetten H1 Überschriften jetzt 300mal im Dokument und der Chef/Kunde kommt und will sie jetzt gerne in Mintgrün. Und morgen kommt er und will sie nicht fett aber kursiv. Und übermorgen unterstrichen. Natürlich gibt es „Suchen und Ersetzen“ im Editor, aber elegant ist anders. Heutzutage steht in der HTML-Datei nur noch

```
<h1>Intro</h1>
```

und es gibt ein CSS-Datei, in der steht

```
h1 { color:red; font-weight:bold}
```

Das Prinzip ist also: mit HTML den Text strukturieren, mit CSS die Strukturelemente formatieren. Das ist übrigens derselbe Ansatz, den Textverarbeitungsprogramme verfolgen: Formatvorlagen zuweisen, um den Text zu strukturieren und dann in den Einstellungen der Formatvorlagen die Textteile gesammelt formatieren.

Für den Unterricht bedeutet das: auch wer HTML-Format-Tags kennt, benutzt diese nicht und bis Ihr CSS lernt müssen wir die Formatierung der Seite halt so hinnehmen, wie sie in den Standardtags verankert ist.

⁶ Die im Beispiel gezeigte Schreibweise funktioniert ab HTML5, in älteren HTML-Versionen muss man

`<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-15" />` verwenden.

Sonderzeichen

Viele Sonderzeichen lassen sich nicht direkt im Quelltext angeben, entweder, weil sie gar nicht über die Tastatur eingegeben werden können, oder weil sie Teil von HTML-Kommandos sind. Die Zeichen < und > werden ja zum Beispiel vom Browser als Anfang bzw. Ende eines Tags interpretiert, sie werden also grundsätzlich nicht in der Ausgabe erscheinen. Will man doch einmal ein solches Zeichen ausgeben, kann man dies auf jeden Fall dadurch erreichen, dass man den Unicode des Zeichens nachschlägt und das Zeichen dann in dem Format

```
&#[nummer];
```

angibt⁷. Die Unicode Nummer kann man auch hexadezimal angeben, dann setzt man noch ein x davor, also

```
&#x[nummer];
```

Außerdem gibt es in HTML für sehr viele Sonderzeichen Namen, deren Verwendung den Code lesbarer machen. Damit gibt es für jedes Sonderzeichen also 3 Möglichkeiten. Beispiele:

Zeichen	Beschreibung	HTML - Name	HTML Unicode	HTML Unicode hexad.
©	Copyright-Zeichen	©	©	©
±	Plusminus	±	±	±
&	Ampersand, kaufm. Und	&	&	&
<	kleiner, less than	<	<	<
	erzwungenes Leerzeichen ⁸	 	 	

Die grundlegenden HTML-Tags

Ich liste die wichtigsten Tags zur Strukturierung hier einfach nur auf, ohne ausführlich zu erklären, wie sie im einzelnen wirken – das erschließt sich bei den meisten aus dem Namen oder der Kurzbeschreibung und wenn das nicht so ist hilft ausprobieren.

⁷ Folglich ist auch das & ein Sonderzeichen, dass man über diesen Umweg angeben muss.

⁸ „Normale“ Leerzeichen, die man mit der der Space-Taste in den Quelltext werden nicht eins zu eins in der Ausgabe angezeigt. Egal wie vieler solcher Leerzeichen man zwischen zwei Worten macht – angezeigt wird immer nur eins.

<h1> , <h2>...	Überschriften	<h1>Ein HTML Crash Kurs</h1>
<p>	Absatz	<p>Es grünt so grün.....</p>
	„Ordered List“ → nummerierte Aufzählung	Wir fordern HTML für alle Abschaffung des Compilerzwangs
	„Unordered List“ → nicht nummerierte Aufzählung	
	„List Item“ Aufzählungspunkt einer Liste	
 	„Break“ → Ein Zeilenumbruch ⁹	<p>Und dann BANG hat er einfach den Browser zugemacht</p><hr />
<hr />	„Horizontal Rule“ → Eine horizontale Linie über die ganze Seitenbreite	

Übung

8. Schreibe die Einladung zu einer Jahresversammlung Deines [Hacker | Taubenzüchter | Gesangs | Schützen | Was-auch-immer]vereins. Denk Dir einen aus, wenn Du in keinem bist. Diese enthält natürlich die Tagesordnung, die Tagesordnungspunkte sind durchnummeriert (→ nummerierte Liste) und enthalten mindestens: Begrüßung und Feststellung der Beschlussfähigkeit, Berichte, Entlastung des Vorstands, Sonstiges. Unter Berichte fallen mindestens Bericht des Vorstands und Bericht des Kassenwarts, hier kannst Du die ungeordnete Liste unterbringen und musst direkt ausprobieren, wie man eine Liste in einer anderen verschachtelt. Versuche, alle oben genannten Tags unterzubringen.

Links und URLs

Die Verknüpfung zwischen Webseiten geschehen durch Links, diese werden mit dem Anchor-Tag erstellt, wobei das href-Attribut die Linkadresse angibt (href → Hyertext Reference). Ein Anchor-Tag kann grundsätzlich alles mögliche umschließen, also sowohl ein Stück Text, als auch Text und ein Bild oder nur ein Bild oder einen ganzen Absatz.

```
<a href="http://www.html.net/">Hier ist ein Link zu HTML.net</a>
```

Als Verknüpfungsziel muss man eine gültige URL (Uniform Resource Locator) angeben, also im Zweifelsfall einen Ausdruck der Form:

```
protokoll://hostname/verzeichnis/verzeichnis.../dateiname
```

Als Protokoll wird man in den meisten Fällen natürlich http oder https verwenden, es gibt aber auch noch andere Möglichkeiten, zum Beispiel ftp:// .

Will man auf Seiten auf dem eigenen Server verweisen, so kann man eine relative URL verwenden, d.h. man lässt Protokoll und Servernamen weg und gibt das Verzeichnis relativ an:

```
<a href="seite2.html">Ein Link zur Seite 2</a>
```

ist ein Link auf die Datei seite1.html, die im gleichen Verzeichnis liegen muss wie die Datei, in der der Link steht.

⁹ Zeilenumbrüche, die man mit der Returntaste in den Quelltext einfügt, werden nicht angezeigt.

```
<a href="../index.html">Startseite</a>
```

verweist dagegen auf die Datei index.html im übergeordneten Verzeichnis, und

```
<a href="../beispiele/b1.html">Beispiel 1</a>
```

auf eine Seite im Verzeichnis beispiel im übergeordneten Verzeichnis. Angenommen, alle hier gezeigten Links stünden in einer Datei seite1.html, dann könnte die Verzeichnisstruktur so aussehen.

```
webseite
|- index.html
|- seiten
|  |- seite1.html
|  |- seite2.html
|- beispiele
|  |- beispiele1.html
|  |- beispiele2.html
```

Wenn man es genau nimmt sind das übrigens alles auch URLs, ausgeschrieben würde es heißen:

```
<a href="file:../beispiele/b1.html">Beispiel 1</a>
```

Das ist auch der Grund, warum man bei Links auf externe Seiten das http bzw. https nicht weglassen kann – der Browser interpretiert Angaben ohne Protokoll automatisch als Dateinamen !

Verweise innerhalb eines Dokumentes

Man kann jedem Tag in einem HTML-Dokument eine ID geben und diese dann verlinken. Dazu gibt man die ID mit dem Hashzeichen an:

```
<h1>Inhalt</h1>
<a href="#einf">Einführung</a>
...
<h1 id="einf">Einführung</h1>
...
```

Eine ID kann man prinzipiell jedem Tag hinzufügen, also auch einem Absatz, einer Tabelle, einer Tabellenzelle und sogar einem Zeilenumbruch.

Man kann IDs auch verlinken, wenn man auf eine andere Seite springt, dazu hängt man einfach den Ausdruck #<name der ID> an die URL an:

```
<a href="http://www.firma.de/abteilungen.html#fertigung">
  Fertigungsabteilung der Firma </a>
```

Zusatzinformation (Tooltip)

Mit dem title-Attribut legt man noch den Text für ein den Tooltip fest, also ein Infofenster, das erscheint, wenn man mit der Maus über den Link fährt:

```
<a href="uebung.html" title="Mach diese Übungen, sie sind wichtig">
  Übungen</a>
```

Übung

9. Schreibe eine Webseite, die den unten stehenden (natürlich aus Wikipedia extrahierten) Text enthält (den Namen des Herren als Überschrift, die Line als horizontal Rule). Verlinke „W3C“ mit <http://www.w3.org/> und „MIT“ mit <http://www.mit.edu>. Mache aus „Zum Anfang“ einen Link auf die Überschrift¹⁰.

Tim Berners-Lee

Sir Timothy John Berners-Lee, (* 8. Juni 1955 in London) ist ein britischer Physiker und Informatiker. Er ist der Erfinder der HTML (Hypertext Markup Language) und der Begründer des World Wide Web. Heute steht er dem World Wide Web Consortium (W3C) vor, ist Professor am Massachusetts Institute of Technology (MIT) und hat seit 2004 einen Lehrstuhl an der Universität Southampton inne.

[Zum Anfang](#)

Bilder

Bilder fügt man mit dem Tag ein, wobei das Attribut SRC den Link (!) zu der Bilddatei angibt. Außerdem kann man mit ALT einen Alternativtext angeben, der angezeigt wird, solange das Bild noch nicht geladen ist. In Browsern für optisch Herausgeforderte wird dieser Alternativtext außerdem vorgelesen.

Beachte, dass man als Bildquelle sowohl eine Datei auf dem eigenen Server als auch aus den Weiten des Internets angeben kann. Im letzteren Fall muss man aber die Möglichkeit erwägen, dass das Bild irgendwann weg ist.

```


```

Und nur zur Erinnerung: auch Bilder können in einem Link stehen:

```
<a href='http://www.wara.de'>  </a>
```

Tabellen

Tabellen entstehen durch Verschachteln der richtigen Tags:

- eine Tabelle besteht aus einem Tabellenkopf und einem Tabellenbody (<thead> und <tbody>)
- diese bestehen aus ein oder mehreren Zeilen (<tr> für „table row“)
- die wiederum aus mehreren Zellen bestehen (<td> für „table data“ oder <th> für Table Heade (Zelle im Tabellenkopf, man kann aber auch hier <td> benutzen)

Alles in allem also:

¹⁰ Zum Ausprobieren einfach das ganz im Browser so sehr vergrößern, dass man selbst diesen kurzen Text scrollen muss.

```

<table>
  <thead>
    <tr><th>x</th><th>f(x)</th></tr>
  </thead>
  <tbody>
    <tr><td>1</td><td>1</td></tr>
    <tr><td>2</td><td>4</td></tr>
    <tr><td>3</td><td>9</td></tr>
    <tr><td>...</td><td>...</td></tr>
  </tbody>
</table>

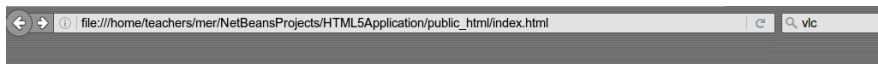
```

Die Unterteilung in `thead` und `tbody`¹¹ kann man auch weggelassen, sie bietet aber bessere Möglichkeiten, die Tabelle mit JavaScript zu manipulieren¹².

Bitte beachtet: `<table>` ist dazu da um tabellarische Daten als Tabelle anzuzeigen und nicht, um das Layout der Seite zu erstellen. Das findet sich zwar als Tipp immer noch in älteren Tutorials und Foren, entspricht aber nicht mehr der modernen Herangehensweise. Lediglich beim „Layouten“ eines Formulars (kommt noch) ist es meiner Meinung nach noch legitim eine Tabelle als Layouthilfe zu missbrauchen.

Übungen

10. Schreibe eine Seite `hoerprobe.html`, die etwa so aussieht:



Willkommen zum **Wara** MP3-Shop

	Song	Länge	
	The Linux Blues	22:41	
	Apfelbrei & Dosenkost	4:12	
	Hasta la vista, Vista	2:41	
	Fenster zur Hölle	13:12	

Zum Abspielen empfehlen wir den freien Multimedia-Player [VLC](#)

¹¹ Es gibt sogar noch `<tfoot>`, aber Tabellenfußzeilen werden eher selten benutzt, deshalb erscheint das Tag hier nur als Fußnote.

¹² Beim Ausdrucken von Tabellen die über mehrere Seiten gehen wird außerdem ein guter Browser den mit `<thead>` definierten Tabellenkopf am Anfang jeder neuen Druckseite wiederholen.

Die Bilder play.png und buy.png kannst Du Dir aus den Vorlagen kopieren. Lege sie in dasselbe Verzeichnis wie die HTML-Datei. Der Text VLC ist ein Link auf die Seite <http://www.videolan.org/vlc/>. Sowohl die Play-Icons als auch der Name der Songs sind gleichzeitig Links auf MP3 Dateien in einem Unterverzeichnis 'mp3'. Lege einfach mit dem Datei-Explorer (oder mit dem Konsolenbefehl touch) vier MP3-Dateien an, einfach nur leere Dateien, hauptsache da ist was zum verlinken. Das ganze müsste dann so aussehen:

```
- hoerprobe.html
- play.png
- buy.png
- mp3
  - song1.mp3
  - song2.mp3
  - hlvv.mp3
  - fzh.mp3
```

11. Schlage in selfhtml (Dokumente → it → selfhtml81 → index.htm¹³ doppelklicken¹⁴) den Punkt Tabellen – Zellen verbinden nach. Erstelle ein HTML-Dokument mit Deinem Stundenplan, bei dem ein Unterricht, der mehrere Schulstunden umfasst in entsprechend verbundenen Zellen angezeigt wird. Fasse die Zellen der ersten Zeile so zusammen, dass es nur zwei Zellen gibt, in die linke kommt der Klassenname in die rechte der des Klassenlehrers. Außerdem werden die Pausen als Zeilen eingezeichnet, in denen sich eine Zelle über die ganze Breite der Tabelle zieht.
12. Erstelle das Eingabeformular für einen LED-Widerstandsrechner. Verwende dabei eine Tabelle mit COLSPAN und ROWSPAN, um die Einzelteile anzuordnen. Die HTML-Befehle, die Du zusätzlich brauchst, heißen

```
<input type='number'>
<input type='text'>
<select><option>1</option><option>2</option>..</select>
<button type='submit'>Aufschrift</button>
```

LED Vorwiderstandsrechner

LED-Flussspannung V

gewünschter Strom A

Betriebsspannung V



¹³ Bitte dort und nicht im Internet. Du solltest wissen, wo Du selfhtml findest, wenn das Internet nicht zur Verfügung steht – so wie in der Prüfung.

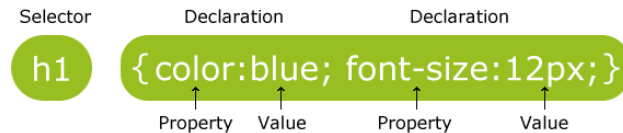
¹⁴ selfhtml ist eine Website und normalerweise enthält index.html bzw. index.html die Startseite einer Website

Jetzt wird's bunt – mit CSS

CSS- Befehle (aus <http://www.w3schools.com/css/default.asp>)

CSS Syntax

A CSS rule set consists of a selector and a declaration block:



The selector points to the HTML element you want to style.

The declaration block contains one or more declarations separated by semicolons.

Each declaration includes a property name and a value, separated by a colon.

CSS Example

A CSS declaration always ends with a semicolon, and declaration groups are surrounded by curly braces:

```
p {color:red;text-align:center;}
```

To make the CSS code more readable, you can put one declaration on each line.

In the following example all <p> elements will be center-aligned, with a red text color:

Die CSS-Befehle kann man übrigens auch direkt über Firefox eingeben (Wirkung wird „live“ angezeigt).

- eine Datei mit Stylesheet lokal öffnen (URL beginnt mit „[file:///...](#)“)
- Style Editor öffnen (Tools – Web Developer – Style Editor oder Umsch-F7)
- loseditieren, gelegentlich speichern (Strg-S)

CSS-Befehle in die HTML-Datei einbauen

CSS in den HTML-Head integriert (ausprobieren, prototyping)

```
<head>
  <title>Beispiel</title>
  <style type='text/css'>
    body { color : yellow; }
  </style>
</head>
```

Eine CSS-Datei verlinken (Normalfall)

```
<head>
  <title>Beispiel</title>
  <link rel="stylesheet" type='text/css' href="meinstil.css">
</head>
```

→ CSS Befehle stehen in meinstil.css

Kurzübersicht der CSS-Befehle, die Du zum Überleben brauchst¹⁵

<i>CSS-Befehl</i>	<i>für</i>	<i>Beispiel / Anmerkung / Typische Werte</i>
Zeichen		
font-family	Zeichensatz wählen	font-family: Comic Sans MS,sans; mehrere Schriften als Liste angeben (Komma), als letzte eine generische Schriftart (sans-serif, serif)
font-size	Zeichengröße	font-size : 200%; Angabe auch in pt, px etc möglich. Prozent bezieht sich auf die Standardschriftgröße, 200% → doppelte Größe
font-weight	Zeichendicke	font-weight : 400; font-weight: bold; (normal, bold)
color	Zeichenfarbe	color : green; color : #20a03f; color : rgb(20,160,63); Farbangaben in RGB (Rot-, Grün-, Blauanteil) mit Werten zwischen 0 und 255 (00 bis FF in der hexadezimalen Darstellung)
Textausrichtung		
text-align	Horizontale Ausrichtung	text-align : left ;
Formatierung von Boxen (aka Blockelemente)		
background-color	Hintergrundfarbe	background-color : yellow; background : #65432e;
background-image background-repeat	Hintergrundbild**	background-image: url(http://www.mein.de/tux2.png) ; background-image: url(pics/bild.jpg); background-repeat: no-repeat; background: #99bbcc url(back.jpg) repeat-y left top;
width, height	Breite, Höhe	div { width : 50%; height : 3em; } span#beispiel { width : 12ex; height : 10%; display : inline-block; }
display	Darstellungsart	a.wichtig { display: inline-block; } div.navi { display: inline-block; } display : block → Element wird als Box angezeigt, mit Zeilenumbruch davor und dahinter, kann mit width und height formatiert werden display : inline → Element wird im Textfluss angezeigt, kein Zeilenumbruch, reagiert nicht auf width/height, padding und margin nur links und rechts möglich display : inline-block → steht inline (mit anderen Elementen in einer Zeile), kann aber in Größe (width, height) und Abstand (margin, padding) beliebig formatiert werden

15 → Pflichtwissen für die KA, Ausnahmen sind mit ** gekennzeichnet

	<p>Für Angaben, die sich auf die vier Seiten der Box beziehen, gibt es immer die Möglichkeit die Angaben:</p> <p>für alle Seiten gleich zu machen: <code>margin : 1em</code></p> <p>für alle Seiten in einem Befehl anzugeben: <code>padding : 2ex 4% 3em 2em;</code> (oben, rechts, unten, links)</p> <p><code>padding : 5em 10%</code> (oben und unten, rechts und links)</p> <p>für jede Seite einzeln anzugeben: <code>margin-left : 10%;</code> (-left, -right, -top, -bottom)</p>	
border border-width border-style border-color	Rand	<code>border : 2px solid #1234a0;</code> <code>border-style : groove;</code> <code>border-bottom-width : 4px;</code> <code>border-top-color : red;</code> <code>border-left : 1ex dashed green;</code> <code>border-style: solid dashed dotted inset;</code>
margin	Äußerer Abstand (zwischen Rand und Rand des Eltern- bzw. Nachbarelements)	<code>margin : 1em;</code> <code>margin-right : 20%</code> <code>margin : 10px 20% 4ex 10%;</code>
padding	Innenabstand Abstand zwischen Rand und Text	<code>padding : 1em;</code> <code>padding-bottom : 0.5ex;</code> <code>padding : 1em 2ex;</code>
	Blöcke zentrieren	<code>margin: 10% 2em; width : auto</code> (→ linken und rechten Rand gleich machen) oder <code>width : 40em; margin: 3em auto;</code> (Breite vorgeben und linken/rechten Rand auto)
border-radius box-shadow	Runde Ecken und Schlagschatten für Rahmen **	<code>border-radius : 19px;</code> <code>box-shadow: -0.5em -0.6em 4px #12412b;</code> (Breite vertikal , Breite horizontal, Blur ,Farbe)

13. Schreibe ein Stylesheet für die Berners-Lee Seite, so dass sie in etwa folgendermaßen aussieht:



Hinweise

- Du darfst im HTML -Code des Body noch DIVs einfügen, sonst aber nichts

- Breite der horizontalen Linie und des Textes je 70%
- Abstand zwischen Browserrand und grauem Feld soll zwei Zeichen betragen
- Abstand vom Text zum Rand des grauen Bereichs ein Zeichen
- die Unterstreichung des Links entfernt man durch das CSS-Attribut `text-decoration:none`

14. Die Formatierung großer CSS-Dokumente steht und fällt damit, dass man die Selektoren beherrscht. Spiele Dich durch die 32 Level auf <https://flukeout.github.io/> damit Du in Zukunft Selektierprofi bist.

15. Formatiere die Seite `uebungLayout.html`. Verwende dabei alle CSS-„Pflichtbefehle“ und setze außerdem das Navigationsmenü neben den Block mit dem Inhalt.

Tabellen und CSS

Bei der Formatierung von Tabellen muss man ein paar kleine Details wissen:

- die Tabelle besteht aus lauter verschachtelten Tags – man muss darauf achten was man formatiert.
`table { border: 1pt solid black; }`
legt einen Rahmen um die ganze Tabelle, aber keinen Rahmen um die Zellen.

`table { padding: 0.7 em; }`

erzeugt einen Abstand zwischen dem Tabellenrahmen und dem Inhalt der Tabelle, es legt aber nicht einen einheitlichen Innenabstand für alle Zellen fest.

- Wer das Aussehen der Zellen formatieren will, muss `td` formatieren:

`td { border: 1pt solid black; padding: 0.4em; }`

legt einen Rahmen um jede einzelne Zelle und definiert Abstand zum Zelleninhalt. Allerdings liegt zwischen den Rahmen der verschiedenen Zellen noch Platz, so dass kein Gitternetz entsteht (was meistens gewünscht ist). Dazu braucht es noch `border-collapse`, das wiederum auf die Tabelle angewendet werden muss:

`table { border-collapse: collapse; }`

- Nicht alle CSS-Eigenschaften werden für alle Tabellenelemente interpretiert. Man kann z.B., keinen Border für `tr` oder `tbody` setzen. Ein `text-align` für `tbody` setzt dagegen die Ausrichtung für alle Zellen im Tabellen-Body. Hier hilft nur – probieren.

16. Schreibe ein Stylesheet für die Seite `hoerspiele.html`. Benutze eine Klasse um jede zweite Tabellenzeile zu markieren und farbig zu hinterlegen¹⁶. Setze

¹⁶ Seit CSS3 geht das auch mit `tr:nth-of-type(2)`. An dieser Stelle sollt Ihr aber den Einsatz von Klassen üben.

per CSS die Breite der Bilder mit dem Playsymbol auf 100 Pixel. Gib der Tabelle einen Rand und ein Gitter¹⁷ und formatiere die Zellen des Tabellenkopfes so, dass es wie ein Tabellenkopf aussieht. Mach das Ganze ein bisschen hübsch (Innenabstände, Schrift etc).

17. Schreibe ein Stylesheet für Deinen Stundenplan. Text grundsätzlich zentriert und vertikal mittig, jede zweite Spalte (Spalte nicht Reihe) hellgrau hinterlegt, gepunktetes Gitternetz, Rahmen um die Tabelle. In den Überschriften eine andere Schriftart als im Rest der Tabelle.

Zwischendurch – Mehr CSS-Selektoren

In den Beispielen der nächsten Kapitel kommen CSS-Selektoren vor, die mehr als nur ein HTML-Tag auswählen. Da wäre zunächst die Möglichkeit, Selektoren aufzuzählen:

```
div, p { background:red;}
```

formatiert sowohl <div> als auch <p> mit rotem Hintergrund.

Desweiteren gibt es Zugriff auf Elemente, die mit Klassen und IDs versehen sind:

```
#wichtigsterKasten { border: 10px solid red; }
.unwichtigeLinks { text-decoration:none; }
```

formatiert das Element mit der ID „wichtigsterKasten“ und alle Elemente mit der Klasse „unwichtigeLinks“:

```
<body>
...
<a href='...' class='unwichtigeLinks'>Nicht wichtig, aber interessant</a>
<div id='wichtigsterKasten'>...</div>
<a href='...' class='unwichtigeLinks'>Nicht wichtig, aber interessant</a>
...
</body>
```

Der Unterschied von IDs und Klassen liegen in der Verwendung: eine ID verwendet man nur einmal, bei Klassen kann man mehrmals die gleiche vergeben. Zuletzt wären da noch die Auswahl anhand der Verschachtelung:

```
ul a { text-decoration: none; }
```

formatiert nur <a>-Tags, die in einer Unordered List () vorkommen.

Layout Teil 1 – Die verschiedenen display-modes

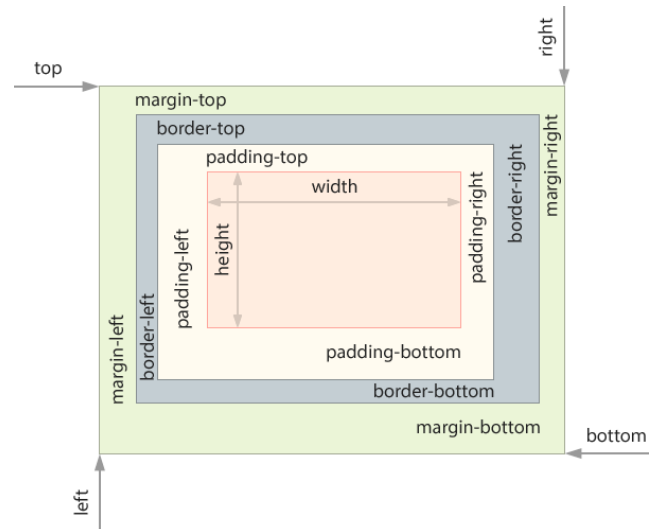
Sobald die Webseite etwas aufwändiger wird, stellen sich Fragen wie „Wie kann ich einen Navigationsmenü neben eine Contentbox setzen“. Bevor man diese Fragen beantworten kann, muss man sich erstmal über die verschiedenen Layout-Arten klar werden, die man für Boxen einstellen kann. Boxen? Ja, zuallererst muss wissen, dass (fast¹⁸) jedes HTML-Element eine Box definiert. Diese Box hat eine Größe, einen Rand, einen Innenabstand und Abstand zu den umliegenden Boxen.

17 Dafür gibt man den Tabellenzellen Ränder

18 Ein
 bildet zum Beispiel keine Box.

Wie eine Box angezeigt wird kontrollieren wir mit dem CSS-Befehl „display“ und hier interessieren uns zunächst mal die drei Eigenschaften „inline“, „block“ und „inline-block“:

Ein „inline“-Element steht mit anderen Elementen oder Text in einer Reihe. Typische Beispiele sind ``, `<a>` oder ``. Inline-Elemente können einen Rahmen haben. Man kann ihre Größen nicht mit Height oder Width festlegen – sie richten sich hier ausschließlich nach ihrem Inhalt. Einen Rand (margin) und Innenabstand (padding) kann man links und rechts setzen aber nicht für oben und unten!



```

*beispiel.html (/tmp) - gedit
Datei Bearbeiten Ansicht Suchen Werkzeuge Dokumente
Öffnen Speichern Rückgängig

<style>
em {
  margin-left: 1em;
  border-bottom: 1px solid blue;
  font-size: 140%;
  vertical-align: top;
}
</style>

<p>
Ein Inline Element ist immer <em>genau</em> so großszlig; wie sein Inhalt und
"fließt" mit dem Text (=steht mit dem Text in einer Zeile)
</p>
<p>
Unterschiedlich <em>großszlig;</em> <a> Inline-Elemente</a> kann man mit vertical-align
aneinander ausrichten.
</p>

```

file:///tmp/beispiel.html
 Zurück tmp/beispiel.html Duck Duck Go
 Ein Inline Element ist immer genau so groß wie sein Inhalt und "fließt" mit dem Text (=steht mit dem Text in einer Zeile)
 Unterschiedlich große Inline-Elemente kann man mit vertical-align aneinander ausrichten.

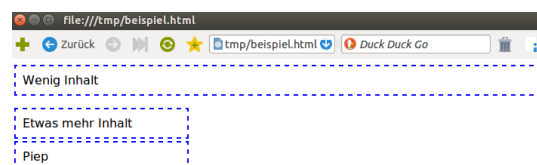
Ein Block-Element ist gerade das Gegenteil. Ein Blockelement steht grundsätzlich nicht mit anderen Element in einer Zeile. Grundsätzlich nimmt ein Blockelement die volle Breite ein, unabhängig von seinem Inhalt. Dafür kann man ein Blockelement mit height und width in der Größe anpassen (wobei es auch bei Breiten unter 100% NICHT mit einem anderen Element in einer Reihe stehen wird). Typische Vertreter von Blockelementen sind `<p>`, `<div>` oder `<h1>`.

```

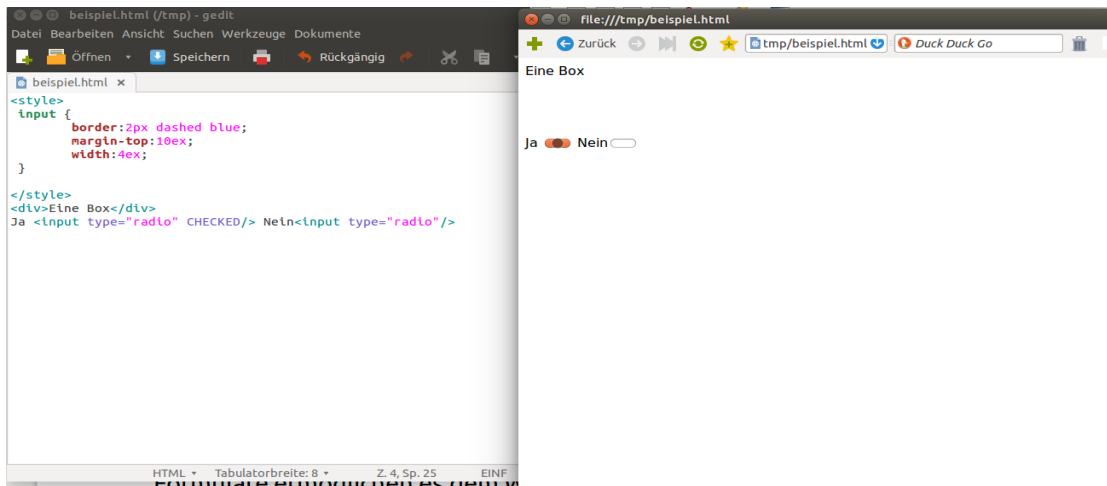
beispiel.html x
<style>
p,div {
  border: 2px dashed blue;
  padding: 1ex;
}
div {
  width: 30%;
  margin-top: 0.2em;
}
</style>

<p>Wenig Inhalt</p><div>Etwas mehr Inhalt</div><div>Piep</div>

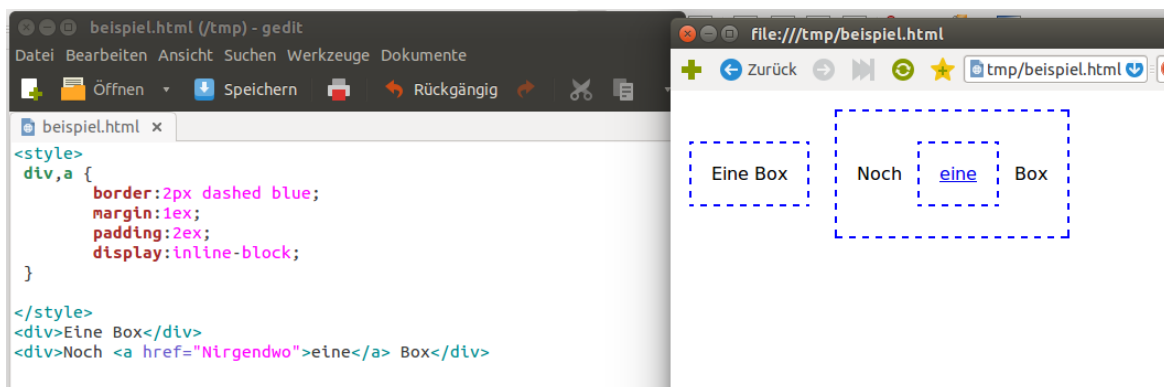
```



Als Mittelding gibt es die Inline-Block Elemente, also Boxen, die mit anderen Boxen in einer Reihe stehen können und bei denen sich Größe, Innenabstand und Rand beliebig festlegen lassen. Die `<input>` Elemente sind zum Beispiel standardmässig mit `display: inline-block` formatiert – sie stehen mit Text in einer Reihe, aber man kann sie mit `width` verbreitern oder ihnen einen oberen Rand geben:



Mit der Formatierungsanweisung `display:inline-block` kann man aus jedem Element ein solches Multitalent machen – man kann mehrer `<div>` dazu bringen, in einer Reihe zu stehen oder einen Link höher und breiter machen, als sein Inhalt vorgibt:



Wofür man das unter anderem konstruktiv verwenden kann, zeigt das nächste Kapitel.

Nur noch schnell vorne weg – wenn Du sehen willst, wie Deine Seite auf verschiedenen Endgeräten aussieht (Tablet, Desktop, Smartphone) kannst Du sie veröffentlichen und dann zum Beispiel diese Seite verwenden:

<http://quirktools.com/screenfly/>

Formulare

Formulare ermöglichen es dem Webseitenprogrammierer, Daten vom Benutzer abzufragen. Dazu stellen sie die üblichen Eingabeelemente grafischer Oberflächen zur Verfügung, also Eingabefelder, [Radio|Check|Klick]-Buttons und Auswahllisten.

```
<form action="http://web.wara.de/~mertens/formulartest.php">
  <div>
    <label for="txemail">Email</label>
    <input type="text" name="txemail" id="txemail">
  </div>
  <div>
    <label for="txmsg">Nachricht</label>
    <textarea name="txmsg" id="txmsg"></textarea>
  </div>
</form>
```

Email

Nachricht

Ein Formular besteht zuallererst aus einem `form`-Tag. Dieses enthält den Parameter `action`, mit dem man festlegt, an welche Adresse die Benutzereingaben geschickt werden.

Danach stellt sich schon die Frage der Anordnung. Früher hat man in Formularen oft Tabellen verwendet, um die Einzelteile aneinander auszurichten. Mehr Möglichkeiten hat man, wenn man die Ausrichtung mit CSS macht. Dafür muss man sein Formular aber entsprechend vorbereiten:

- für jede Zeile ein `DIV` (also eine Box)
- für die Beschriftung ein `LABEL`. Der `FOR` Parameter sorgt dafür, dass man auf das Label klicken kann und dann in der zugehörigen Eingabe landet

Das Beispiel zeigt zwei typische Vertreter von Eingabeelementen:

- auf dem `INPUT`-Tag basieren z.B. einzeilige Textfelder, Radio- und Checkboxes. Dieses Tag ist selbstschließend
- manche Elemente haben ein Start- und Endtag, z.B. `TEXTAREA` (mehrzeiliges Textfeld) und `SELECT` (Drop-down-Liste)

Wichtig !

Ein Eingabeelement muss immer einen Namen haben. Den Namen verwendet die Empfängerseite, um die Eingaben abzuholen!

Die ID im Beispiel dient dem Zusammenspiel mit dem `LABEL`-Tag. Wer im `LABEL` das `FOR` weglässt, kann sich auch die ID sparen.

Formular formatieren

Jetzt fehlt noch die Ausrichtung der Felder. Dazu kann man entweder die Felder in eine Tabelle stecken oder CSS verwenden. Letzteres geschieht zunächst einfach dadurch, dass man allen `LABEL`s die gleiche Breite gibt. Dabei muss man allerdings noch die Darstellungsform berücksichtigen: `LABEL` ist standardmäßig ein so genanntes „Inline“-Element, dass heißt es nimmt immer nur so viel Platz ein, wie es braucht und reagiert nicht auf `width`. Um ihm zu sagen, dass er auf seine „width“-Angabe auch hören soll, muss man es auf `display:inline-block` umsetzen.

```
label {
    width:15ex;
    display:inline-block;
    margin-bottom: 1em;
}
textarea {
    vertical-align: top;
}
form {
    background: rgb(218,225,100);
    display: inline-block;
    padding:1em;
}
```



Für `FORM` bedeutet das `inline-block`, dass es nicht die ganze Seitenbreite einnimmt, sondern nur so viel, wie nötig.

Zur Verdeutlichung von „display“ und dem Vorteil von CSS vs Tabellen: indem man die `LABEL`s als `display:block` formatiert (also jedes Label bildet einen Block, inklusive

Zeilenumbruch) kann man mit wenig Änderungen im CSS das Formular deutlich anders aussehen lassen. Im Zusammen

```
label {
  display: block;
  margin-bottom: 1em;
  margin-top: 1em;
}
```

Im Zusammenhang mit „media-queries“ kann man das sogar soweit automatisieren, dass dieselbe Seite auf einem großen Bildschirm so aussieht wie oben und auf kleinen Bildschirmen so wie unten.

Übung Formulare

Eine Übersicht über HTML-Formularelemente inklusive der Browserunterstützung und Beispiele für das Aussehen auf verschiedenen Browsern findest Du hier:

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/The_native_form_widgets

18. Schreibe eine Seite mit einem Formular, das so aussieht wie der Screenshot. Wichtig ist an dieser Stelle vor allem das HTML. Recherchiere selbst, wie man Radiobuttons erstellt¹⁹. Du kannst das Formular die Seite <https://capo.wara.de/~mer/formulartest.php> aufrufen lassen, das ist ein kleines Skript, das anzeigt, was Dein Formular an den Server übermittelt²⁰.

Hinweise:

- der Rahmen mit der eingesetzten Überschrift ist letztendlich HTML: man benutzt dazu ein Fieldset mit einer Legende:

```
<form...><fieldset><legend>Klassenfahrt...</legend>
  <!-- Formularelemente ..... -->
</fieldset></form>
```

- der zweite Button ist vom Typ „reset“. Probiere selbst aus, was er bewirkt
- etwas zentrieren kann man, indem man den rechten und linken Rand auf den gleichen Wert setzt, am besten benutzt man dabei Prozentangaben
- man kann mit CSS auch anhand eines Attributs selektieren. `input {..}` würde

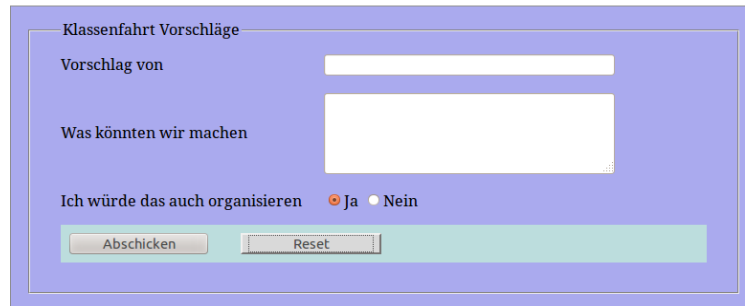
¹⁹ Recherchiere am besten nicht im Internet sondern in selfhtml oder Deinem IT-Handbuch, die hast Du auch in der Prüfung

²⁰ Ausnahme: enthält Dein Formular Checkboxes, so werden diese nur angezeigt, wenn sie angeklickt sind. Für eine nicht angeklickten Checkbox wird nichts an den Server übermittelt – gar nichts.

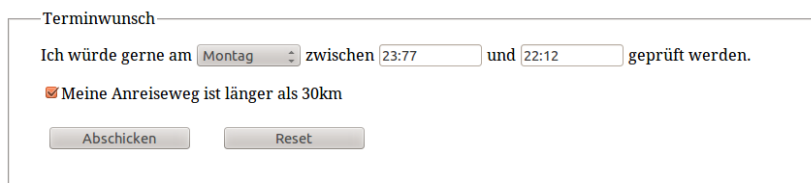
auf alle Button und Textfelder wirken

`input[type='submit'] {...}` formatiert nur den Submit-Button

- Du musst die Farben nicht genau reproduzieren, aber das Formular sollte eine andere Farbe haben als die Box mit den Buttons
- beachte und reproduziere: der Innenabstand zwischen Formularinhalt und dem Fieldset ist oben und unten anders als seitlich



19. Programmiere eine Seite mit einem Formular, bei dem man einen Terminwunsch für eine Prüfung angeben kann (siehe Screenshot). Für die Zeitangaben kannst Du ein Inputfeld vom Typ `time` verwenden. Die Auswahlbox musst Du nachschlagen oder Dich mit Netbeans und Trial-and-Error rantasten. Der ausgewählte Tag sollte als Nummer beim Server ankommen (0 für Montag, 1 für Dienstag ...). Probiere verschiedene Browser aus – der Typ „`time`“ für das Tag input ist eine Neuerung von HTML 5 die noch nicht besonders gut unterstützt wird. Wir haben als Browser Firefox, Midori und Chromium installiert (→ starten per Dashboard).



Positionierung der Seitenelemente & Seitenlayouts

Du hast jetzt einen ersten Einblick in die Grundtechniken der Webprogrammierung bekommen. Zum Aufbau einer kompletten Seite fehlt Dir noch das Wissen darüber, wie man die Seite im gesamten gestaltet – Überschrift, dann Navigationsbalken, dann Content, daneben Werbung. Oder Navigation neben Content. Oder so. Dieses Thema war mal kompliziert, aber mit CSS3 hat man das Prinzip der Flexbox eingeführt, das genau diese Arbeit massiv erleichtern soll. Die Unterstützung durch die Browser ist noch nicht 100%, aber schon recht gut und mittelfristig sollte die Flexbox Standard werden. Einarbeitung lohnt sich schon alleine, weil gerade die Browser der Smartphones sie schon recht gut unterstützen. Im Folgenden findest Du zwei Seiten zum Thema Flexbox. Die erste führt in das Grundprinzip ein, die zweite ist eher eine Übersicht über die Möglichkeiten:

<http://tanja-volk.de/arbeiten-mit-flexboxen/>

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Lies Dir die Seite von "Arbeiten mit Flexboxen" durch und mache danach das dort verlinkte Lernspiel "flexboxfroggy" (<http://flexboxfroggy.com/>).

Du willst mehr (und hast noch Zeit) ? Hier: <http://www.flexboxdefense.com/>

20. Programmiere die Seite mit dem Terminwunschoformular nochmal (siehe Screenshot). Verwende die Inputfelder für time und date. Benutze flexboxen, um die Zeilen horizontal im Verhältnis 2:3 aufzuteilen (bzw. 1:1 bei den Buttons).

Benutze für die Vorschau einen Browser, der die Zeit und Datumsfelder auch komplett interpretiert.....

Terminwunsch

Ich würde gerne am

zwischen

geprüft werden.

Abschicken

Reset

tt.mm.jjjj

November 2015

Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.
26	27	28	29	30	31	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	1	2	3	4	5	6

Terminwunsch

Ich würde gerne am

zwischen

geprüft werden.

Abschicken

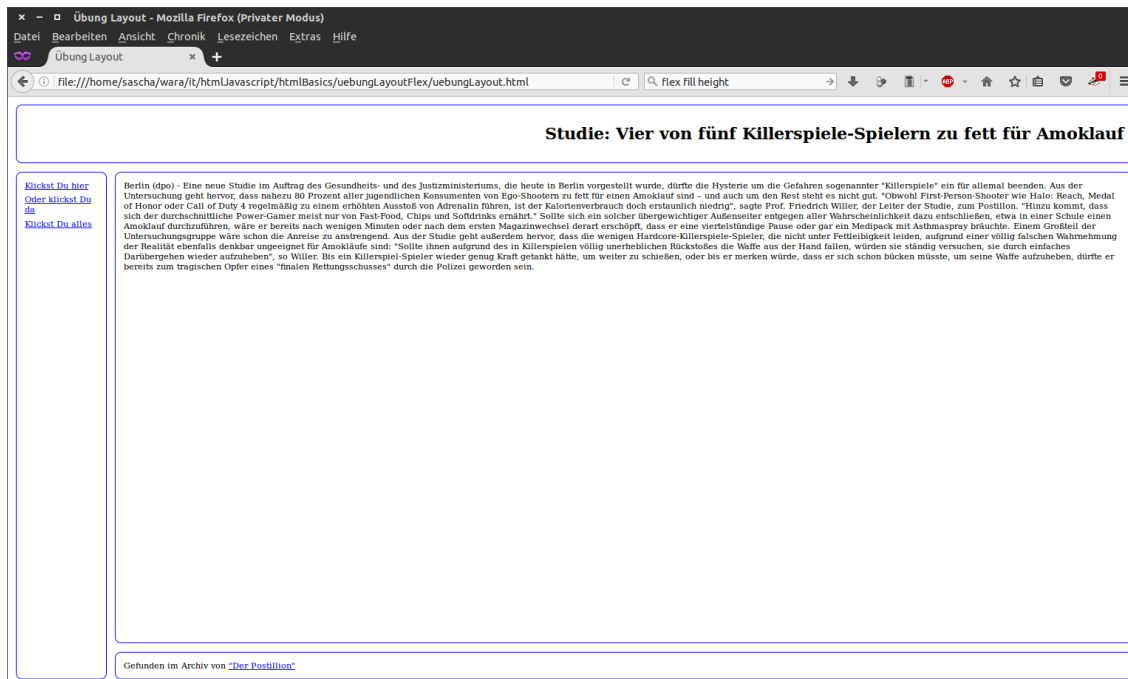
Reset

tt.mm.jjjj

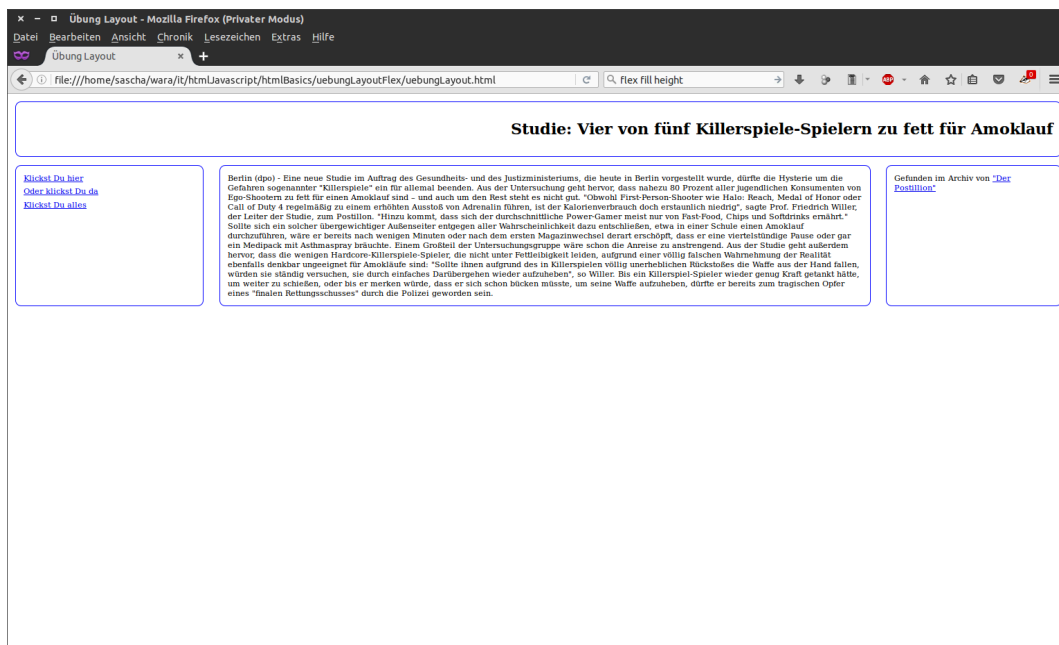
-- und --

☐ Meine Anreiseweg ist länger als 30km

21. Kopiere Dir die Vorlagen uebungLayout.html und uebungLayout1.css und ergänze das CSS, so dass die Seite so aussieht wie im folgenden Screenshot.



22. Erstelle die Dateien uebungLayout2.css und uebungLayout3.css und bilde auch die folgenden zwei Screenshots nach²¹.



21 Man kann in ein HTML-Dokument auch mehrere CSS-Dateien einbinden. Man gibt dazu bei einem rel="stylesheet" an und bei allen anderen rel="alternate stylesheet". Der User kann dann in seinem Browser einstellen, welchen Style er sehen will (bei Firefox unter Ansicht → Webseiten-Stil)

Übung Layout - Mozilla Firefox (Privater Modus)

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

Übung Layout

File:///home/sascha/wara/it/htmlJavaScript/htmlBasics/uebungLayoutFlex/uebungLayout.html

flex fill height

Studie: Vier von fünf Killerspiele-Spielern zu fett für Amoklauf

Gefunden im Archiv von "Der Postillon"

Berlin (dpa) - Eine neue Studie im Auftrag des Gesundheits- und des Justizministeriums, die heute in Berlin vorgestellt wurde, dürfte die Hysterie um die Gefahren sogenannter "Killerspiele" ein für allemal beenden. Aus der Untersuchung geht hervor, dass nahezu 80 Prozent aller jugendlichen Konsumenten von Ego-Shootern zu fett für einen Amoklauf sind - und auch um den Rest steht es nicht gut. "Obwohl First-Person-Shooter wie Halo: Reach, Medal of Honor oder Call of Duty 4 regelmäßig zu einem erhöhten Anstieg von Adrenalin führen, ist der Kalorienverbrauch doch erstaunlich niedrig", sagte Prof. Friedrich Wüster, der Leiter der Studie, zum Positionskongress. "Hinzu kommt, dass sich der durchschnittliche Power-Gamer meist nur von Fast-Food, Chips und Softdrinks ernährt." Sollte sich ein solcher übergewichtiger Außenseiter entgegen aller Wahrscheinlichkeit dazu entschließen, etwa in einer Schule einen Amoklauf durchzuführen, wäre er bereits nach wenigen Minuten oder nach dem ersten Magazinwechsel derart erschöpft, dass er eine viertelstündige Pause oder gar ein Medipack mit Aftershave brauche. Einem Großteil der Untersuchungsgruppe wäre schon die Anreise zu anstrengend. Aus der Studie geht außerdem hervor, dass die wenigen Hardcore-Killerspiele-Spieler, die nicht unter Fettigkeit leiden, aufgrund einer völlig falschen Wahrnehmung der Realität ebenfalls deshalb ungeeignet für Amokläufe sind: "Sollte ihnen aufgrund des in Killerspielen völlig unerheblichen Rückstoßes die Waffe aus der Hand fallen, würden sie ständig versuchen, sie durch einfaches Darübergehen wieder aufzuheben", so Wüster. Bis ein Killerspiel-Spieler wieder genug Kraft gelangt hätte, um weiter zu schießen, oder bis er merken würde, dass er sich schon blicken müsste, um seine Waffe aufzuheben, dürfte er bereits zum tragischen Opfer eines "fiktionalen Rettungsschusses" durch die Polizei geworden sein.

Klickst Du hier

Oder klickst Du da

Klickst Du allen