Webdesign – Tutorial

Einleitung & Vorschau

Heutzutage kann sich jeder ohne größere Anstrengungen eine eigene Homepage zulegen. Es gibt zahlreiche Anbieter, wo man sich kostenlos und ohne HTML-Kenntnisse eine kleine Webseite mittels „Baukasten“ zusammenbasteln kann (siehe Homepage-Hilfe)

Leider sieht das Ganze dann auch dementsprechend aus – wer seine Seite ansprechend gestalten und vielleicht auch ein eigenes Design haben möchte, kommt auf kurz oder lang nicht um Kenntnisse in HTML und CSS herum. Sicherlich könnte man sich ein Layout erstellen und es von anderen umsetzen lassen (umgangssprachlich auch als „coden“ bezeichnet), aber auf die Dauer nervt es schon tierisch auf andere angewiesen zu sein – oder?

Deshalb habe ich beschlossen einen Kurs zum Thema **Websitegestaltung, HTML und CSS** zu starten – ich möchte euch das nötige Wissen vermitteln, um eine eigene Webseite aufzuziehen

Dieser Kurs soll kein komplettes Kompendium mit allen erdenklichen HTML-Befehlen werden, sondern das Wissen vermitteln, eine eigene kleine Webseite zu erstellen und nach seinen Wünschen zu gestalten. Ein umfangreiches Nachschlagewerk zum Thema HTML und CSS ist www.selfhtml.org – man muss nicht alles wissen, man muss nur wissen wo’s steht ;) !

Meine Planung für den Kurs sieht wie folgt aus:

0. Einleitung & Vorschau
1. HTML – Allgemeines & Regeln
2. Die Grundstruktur von HTML-Dokumenten
3. Textstrukturierung
4. Text- & Schriftformatierung
5. Farben
6. Hyperlinks
7. Listen & Aufzählungen
8. Bilder einbetten
9. Tabellen
10. Frames
11. Formulare
12. Vermischtes, eure Wünsche
13. CSS – Einstieg (Syntax, Definition, Selektoren)
14. CSS – Werte & Eigenschaften
15. CSS – Layer erzeugen & positionieren

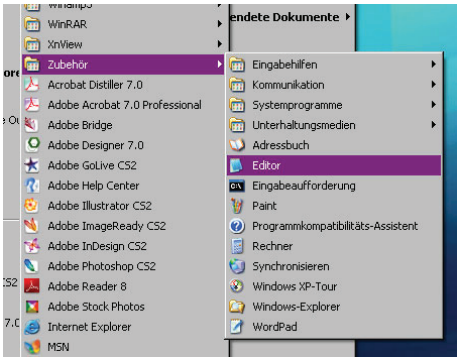
Darauf folgen dann in unregelmäßigen Abständen weitere diverse (kleinere und größere) Tutorials zu interessanten Themen und Problemen – dabei könnt ihr auch jederzeit Fragen stellen, ich werde mich bemühen sie zu beantworten.

1. HTML – Allgemeines und Regeln

Einfache Webseiten haben das **Dateiformat** *.html oder *.htm.
Bei der Erstellung von Dateien für die Website (also HTML-Dateien, Bilder, Scripte usw.) müssen einige Kleinigkeiten bei den **Dateinamen** beachtet werden, um mögliche Ausgabefehler zu vermeiden:
- es sollten grundsätzlich nur Kleinbuchstaben verwendet werden
- nur Buchstaben, Ziffern, Unter- und Bindestriche benutzen; Umlaute (ä, ö, ü), „ß“ und Leerzeichen sind absolut tabu!
- immer auf die richtige Dateiendung achten (*.htm, *.html, *.php, *.jpg, *.gif, *.png, *.js, *.ico, *.pdf, *.css usw.)

Um eine **HTML-Seite** zu **erstellen** werden mindestens 2 Programme benötigt: ein Editor zum Erstellen einer Website und ein Browser, um sich das Ergebnis anzusehen. Um eine größere Zielgruppe zu erreichen, sollte die Seite in mehreren Browsern getestet werden. Desweiteren wird ein FTP-Programm benötigt, um die erstellten Dokumente ins Internet hochzuladen.
Diese ganze Software bekommt man kostenlos im Internet zum Download oder ist meist bereits auf dem PC installiert.

Auf jedem PC ist der **Browser** „Internet Explorer“ sowie der **Editor** „Notepad“ installiert. Notepad ist ein einfaches Textprogramm, mit der normalerweise Textdateien (*.txt) geschrieben werden – quasi eine abgespeckte Variante von Microsoft Word, ohne große Möglichkeiten zur Textformatierung etc. Notepad findet ihr unter Start → Programme → Zubehör → Editor.



Um sich die Arbeit zu erleichtern kann man sich den Editor Phase5 runterladen.
Als zusätzlicher Browser zum Testen der Website sollte in jedem Fall der Mozilla Firefox verwendet werden. Weitere interessante Browser zum Testen wären Netscape und Opera.

Zusätzlich zu den Quelltext-Editoren gibt es so genannte „**WYSIWYG**“-Editoren („What you see is what you get“) – man tippt also keinen Quellcode ein, sondern baut die Seite so zusammen, wie sie später im Browser erscheinen soll. Das ist allerdings Wunschdenken, da jeder Browser seine Eigenheiten hat und man um das Testen und manuelle nachbearbeiten des Quelltextes nicht herum kommt.

Die 3 bekanntesten WYSIWYG-Editoren sind „Dreamweaver“ und „GoLive“ von Adobe sowie „FrontPage“ von Microsoft. Besonders letzterer ist jedoch nicht zu empfehlen, da sehr viel unnötiger Quellcode-Müll entsteht. Die Wahl zwischen Dreamweaver und GoLive ist reine Geschmackssache. Beide Programme haben ihre Vor- und Nachteile. Ich selber habe mit GoLive angefangen und arbeite nach wie vor sehr gut damit.
Wer auf den Geldbeutel achten muss, sollte auf GoLive zurückgreifen – die aktuelleste Version habe ich schon hab 30 € bei eBay gesehen, Dreamweaver hingegen für 200 € aufwärts.

2. Die Grundstruktur von HTML-Dokumenten

HTML ist eine Abkürzung für „Hypertext Markup Language“, also „Auszeichnungssprache für Hypertext“ (Hypertext: Text mit eingebauten Querverweisen).

Ein HTML-Dokument besteht hauptsächlich aus so genannten **Tags**. Ein Tag wird von spitzen Klammern (< und >) umschlossen.
Grundsätzlich gibt es 2 Arten von Tags:
Einfache Tags, die an einer bestimmten Stelle im Dokument eine Funktion haben (z.B. ein Zeilenumbruch
) und die **Container-Tags**, welche durch ein einleitendes und ein ausleitendes Element funktionieren und so den Geltungsbereich des Befehls deutlich machen (z.B. <u>text</u> für unterstrichenen Text).
Die Container-Tags finden die häufigste Verwendung.
Viele Elemente (Tags) besitzen **Attribute**, denen man bestimmte **Werte** zuweisen kann. Der Aufbau eines solchen Tags ist immer:

HTML-Code:

```
<element attribut="wert"> Text </element>
```

Ein konkretes Beispiel hierzu:

HTML-Code:

```
<font color="red"> roter Text </font>
```

aber dazu später mehr.
Wichtig ist, dass ihr euch die **Syntax** einprägt: Attribute (z.B. color) werden durch ein Leerzeichen getrennt, durch ein Gleichheitszeichen (=) wird ihnen ein Wert zugewiesen, der in Anführungsstrichen („red“) steht.

Nun genug geredet, gehen wir zur Praxis über.
Startet den Editor eures Vertrauens (ich werde während dieses Kurses mit Notepad arbeiten).

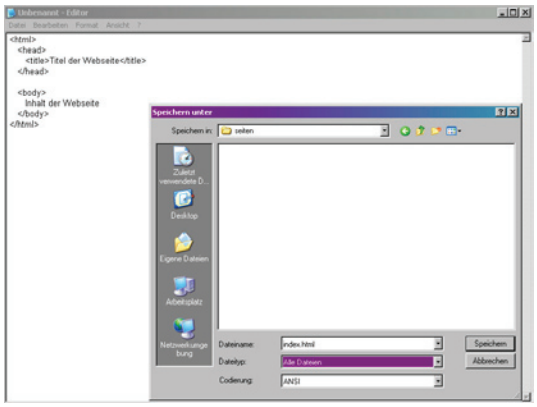
Jede HTML-Seite besteht aus einem Kopf-Bereich (Head) und einem Körper (Body).
Im **Head** werden Titel des Dokuments und bei Bedarf Meta-Angaben geschrieben. Dieser Bereich wird vom Besucher nicht gesehen.
Im **Body** steht der eigentliche Inhalt der Webseite, den jeder User einsehen kann.
So ist eine HTML-Seite wie folgt aufgebaut:

HTML-Code:

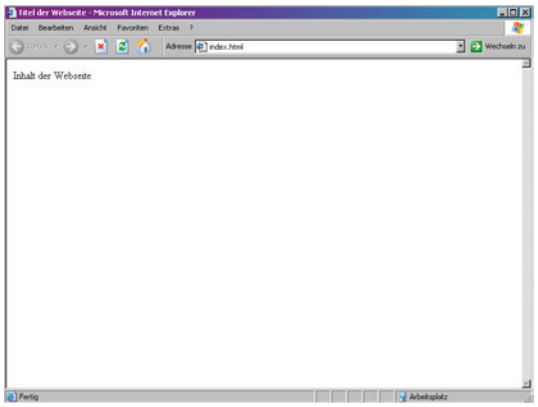
```
<html>
  <head>
    <title>Titel der Webseite</title>
  </head>

  <body>
    Inhalt der Webseite
  </body>
</html>
```

Erklärung der Zeilen:
- <html> teilt dem Browser mit, dass es sich um ein HTML-Dokument handelt
- <head> leitet den Kopfbereich ein
- <title> öffnet den Titel, welcher in jedem HTML-Dokument vorhanden sein muss
- </titel> schließt den Titel
- </head> schließt den Kopfbereich
- <body> leitet den sichtbaren Inhalt des Dokuments ein
- es folgt Text oder weitere HTML-Tags, um selbigen zu formatieren
- </body> schließt den Körperbereich
- </html> schließt das HTML-Dokument

Das gebt ihr nun in eurem Editor ein. Um den Text zu einer **HTML-Seite** werden zu lassen, müsst ihr ihn im entsprechenden Format speichern. Dazu geht ihr auf Datei Speichern unter..., wählt bei Dateityp „Alle Dateien“ aus und gebt dem Dokument einen Namen (Regeln aus Part 1 beachten) und legt darin die Endung **.html** fest. In meinem Beispiel habe ich dem Dokument den Namen „index.html“ gegeben (Abb. rechts).



Anschließend könnt ihr euren **Browser** öffnen und die soeben gespeicherte Datei über Datei Öffnen aufrufen, im Browser sollte nun die Textzeile „Inhalt der Webseite“ zu lesen sein (Abb. unten):



Je nach dem, was ihr im „title“ und „body“ geschrieben habt, wird es natürlich entsprechend angezeigt – probiert ruhig mal ein bisschen rum.

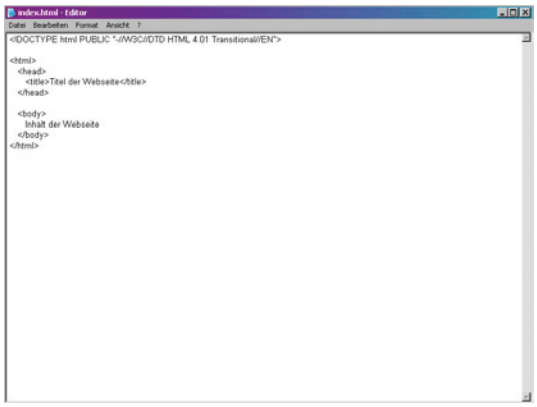
Abschließend benötigt jede HTML-Seite eine **DOCTYPE-Deklaration**. Diese teilt dem Browser mit, mit welcher Auszeichnungssprache ihr arbeitet und welche Version ihr verwendet.

Der gängigste Doktype lautet:

HTML-Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

Diese Zeile fügt ihr direkt an den Anfang des HTML-Dokuments ein (Abb. rechts).

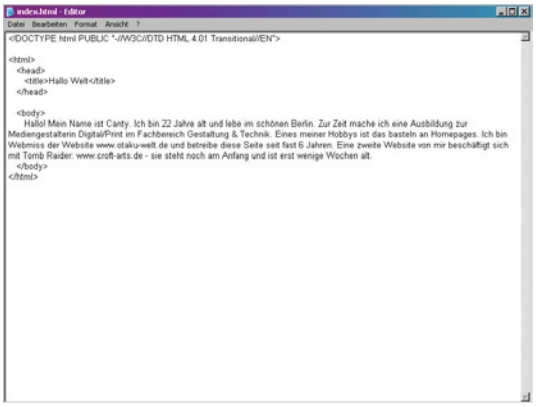


Glückwunsch, ihr habt eine HTML-Seite erstellt :) – so schwer war das doch nicht, oder?

3. Textstrukturierung

Ich habe einen kleinen Text über mich verfasst. Mit diesem Arbeite ich jetzt weiter. Ihr könnt euch ebenfalls einen Text erstellten oder mit einem Blindtext arbeiten.

Öffnet eure HTML-Datei, sie müsste nun also etwa so aussehen (Abb. rechts):



Der Browser braucht eine klare Anweisung wie er einen Text zu strukturieren hat. Dafür gibt es spezielle Tags, die ich euch im Folgenden erklären werde.
Wichtig ist: macht ihr im Quelltext einen Zeilenumbruch (also “Enter” drücken), interessiert das den Browser herzlich wenig. Genauso verhält es sich mit Texteinzügen und mehr als einem Leerzeichen – alles muss speziell ausgezeichnet werden. Diese Umstand kann man jedoch nutzen und so seinen Quelltext übersichtlich und ordentlich gestalten. Durch Umbrüche und Einzüge bekommt er eine gute Struktur und ihr verliert auch bei sehr großen Webseiten nicht den Überblick.

Um Texte zu strukturieren stehen euch Zeilenumbrüche, Absätze und Überschriften zur Verfügung.

Zeilenumbrüche

Soll der Browser einen Zeilenumbruch erzeugen, muss folgendes Tag benutzt werden:

HTML-Code:

```
<br>
```

Das “br” steht für “break” (engl. Bruch, Trennung) und ist eines der wenigen einfachen Tags.

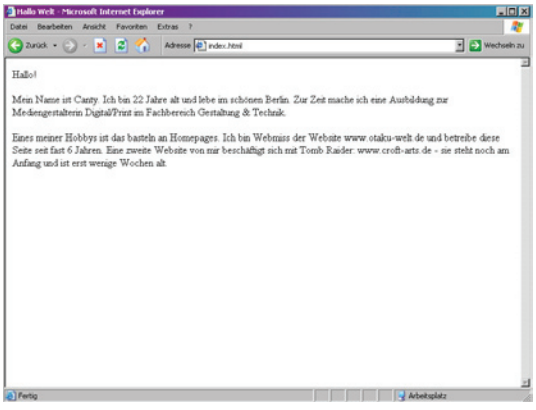
Absätze

Um Absätze zur Kennzeichen wird der Container-Tag benötigt:

HTML-Code:

```
<p> ... Text ... </p>
```

“p” bedeutet “paragraph” (engl. Absatz, Abschnitt). Der Tag erzeugt einen etwas größeren Abstand zum nächsten, als ein einfacher Zeilenumbruch.
Ein angewendeter Absatz sieht im Quelltext und Browser wie folgt aus:



Überschriften
Überschriften werden in 6 “Hierarchien” unterteilt, mit denen man seinen Text übersichtlich gliedern kann:

HTML-Code:

```
<h1> ... Text ... </h1>
<h2> ... Text ... </h2>
<h3> ... Text ... </h3>
<h4> ... Text ... </h4>
<h5> ... Text ... </h5>
<h6> ... Text ... </h6>
```

Standardmäßig werden Überschriften fett und groß dargestellt – das kann man aber manuell ändern (wie das geht behandeln wir im nächsten Kapitel, die optimale Lösung lernt ihr erst später mit CSS kennen).
Wichtig sind diese Überschriften für Suchmaschinen (wobei hier nur h1 bis h3 beachtet werden) und für die Übersichtlichkeit im Quelltext.

Ausrichtung
Absätze und Überschriften können mit dem Attribut “align” nach links, rechts, mittig oder blockartig ausgerichtet werden.

HTML-Code:

```
<p align=”left”> ... Text ... </p>
```

ergibt einen linksbündigen Text – das ist die Standardeinstellung und wird auch ohne das align=”left” erzeugt

HTML-Code:

```
<p align=”right”> ... Text ... </p>
```

ergibt einen rechtsbündigen Text

HTML-Code:

```
<p align=”center”> ... Text ... </p>
```

ergibt einen zentrierten Text

HTML-Code:

```
<p align=”justify”> ... Text ... </p>
```

ergibt einen Blocktext

4. Text- & Schriftformatierung

Innerhalb einer Seite kann man Zeichen, Wörter oder Absätze besonders formatieren oder hervorheben. Dies funktioniert mit sogenannten "**Strukturtags**" oder "**Layouttags**".
Die Tags können beliebig ineinander verschaltet werden, solange man die richtige Reihenfolge beachtet: das Tag, das zuletzt geöffnet wurde, wird auch zuerst geschlossen. So kann man zum Beispiel einen Text fett und unterstrichen setzen:

HTML-Code:

```
<b><u> ... Text ... </u></b>
```

Folgende Tags stehen zur **Zeichenformatierung** zur Verfügung:

Tag	Beschreibung	Beispiel
Text	fett	**fett**
Text	kursiv	*kursiv*
Text	fett (bold), veraltet: heute wird benutzt	**fett**
Text	kursiv (italic), veraltet: heute wird benutzt	*kursiv*
<u> *Text* </u>	unterstrichen	unterstrichen
<strike> *Text* </strike >	durchgestrichen	~~durchgestrichen~~
^{*Text*}	hochgestellt (superscript)	hochgestellt
_{*Text*}	tiefgestellt (subscript)	tiefgestellt
<code> *Text* </code>	Quellcode, zur Darstellung von Codebeispielen	`Code`

Mit HTML kann man natürlich auch die **Erscheinung der Schrift** im Browser beeinflussen. Es kann eine **Schriftart**, **-größe** sowie **-farbe** zugefiesen werden. Diese Schriftattribute werden innerhalb des <font>-Tags eingesetzt. Die optimale Weise um Schrift zu formatieren erlernt ihr in einem späteren Kapitel, wenn es um CSS geht.

Der folgende Tag allein hat keine Auswirkung auf die Schrift.

HTML-Code:

```
<font> ... Text ... </font>
```

Erst, wenn man ihn mit Attributen versieht regelt er die Erscheinung der Texte. Folgende Attribute stehen zur Verfügung: face, family, size und color.

Die **Schriftart** wird mit dem Attribut "face" oder "family" angegeben. Es gibt nur wenige Schriften, die auf allen Rechnersystemen installiert sind und die man damit im Web verwenden kann:
Verdana
Arial
Trebuchet MS
Georgia
Times New Roman
Courier New

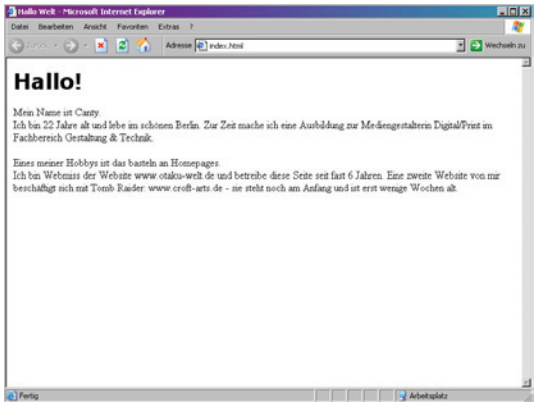
Mit folgendem Code wird also ein Text in der Schriftart Verdana erzeugt:

HTML-Code:

```
<font face="verdana"> ... Text ... </font>
```

Ich demonstriere das mal anhand der HTML-Datei aus den letzten Teilen.

In der Überschrift füge ich das -Tag mit dem Attribut “face” ein und weise ihm den Wert “Verdana” zu. Im Browser wird das ausgegeben wie auf dem rechten Screenshot:



die Überschrift “Hallo” wird in der Schrift Verdana dargestellt.

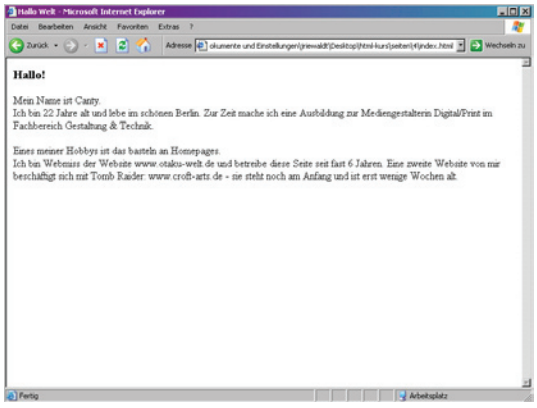
Es können auch mehrere Schriften angegeben werden, welche mit Komma und Leerzeichen getrennt werden. Der Browser wird, sollte die erste Schriftart nicht auf dem Rechner vorhanden sein, die zweite aufrufen usw.
Wer eine andere ungewöhnliche Schrift, z.B. für Überschriften, benutzen will, muss diese als Bilder vorbereiten und in die Webseite einbinden.

Mit dem Attribut “size” kann die **Schriftgröße** angegeben werden. Hier stehen in HTML die Zahlen 1 bis 7 zur Verfügung. 1 ist die kleinste, 7 die größte Schrift:

HTML-Code:

 ... *Text* ...

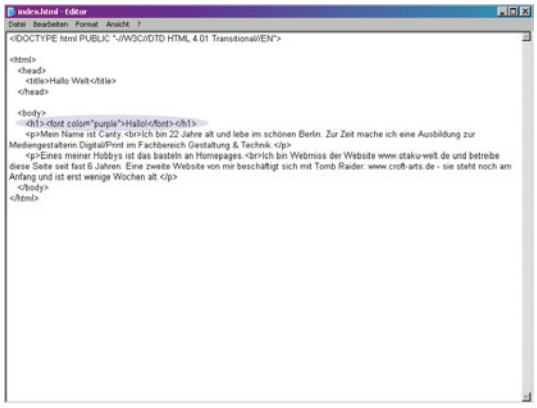
Wird die Schriftgröße nicht angegeben, so wird der Text in Größe 3 angegeben.
Im folgenden Beispiel habe ich meiner Überschrift eine Größe von 4 zugewiesen:

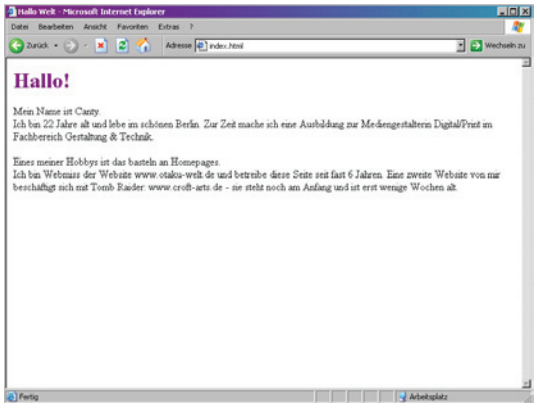


Natürlich kann man auch die **Farbe der Schrift** beeinflussen. Dies funktioniert mit dem Attribut color:

HTML-Code:

 ... *Text* ...



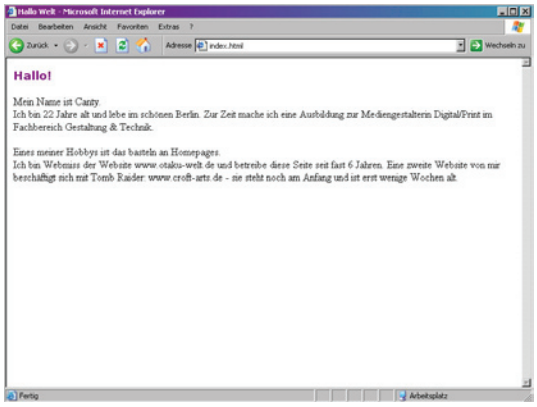


Als **Farbwerte** stehen euch folgende Standardnamen zur Verfügung:
aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow.
Desweiteren könnt ihr eine schier unendlich große Farbpalette nutzen (16,7 Millionen) – die Hexadezimalfarben. Diese werden wir im nächsten Kapitel genauer beleuchten.

Selbstverständlich können alle genannten font-Attribute “face”, “size” und “color” **in ein font-Tag** geschrieben werden. Sie werden einfach mit einem Leerzeichen getrennt. Das sieht dann zum Beispiel so aus:

HTML-Code:

 ... *Text* ...



5. Farben

Auf Monitoren werden Farben im so genannten “RGB-Modus” dargestellt. RGB steht für “Rot – Grün – Blau”.
Um Farben auf Webseiten zu definieren, stehen euch die **Farbnamen** aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white und yellow zur Verfügung, die bereits im letzten Kapitel erklärt wurden. Diese Art der Farbangabe wird jedoch nicht unbedingt empfohlen.

Besser geeignet zur Farbdefinition sind die **Hexadezimalfarben**, bei denen ihr aus 16,7 Millionen Farben wählen könnt.
Vor einem Hexadezimal-Farbcode steht immer ein #-Zeichen (“Raute”). Ein solcher Farbcode besteht aus einer sechsstelligen Kombination aus den Ziffern 0-9 und den Buchstaben A-F.

Die Ziffern zeigen, wie viele Teile rot, grün und blau in einer Farbe anhalten sind – 0 steht dabei für “gar nicht” und F für “vollständig”.

So sieht z.B. der Farbcode von weiß so aus:

HTML-Code:

```
#ffffff
```

der Farbcode von schwarz lautet:

HTML-Code:

```
#000000
```

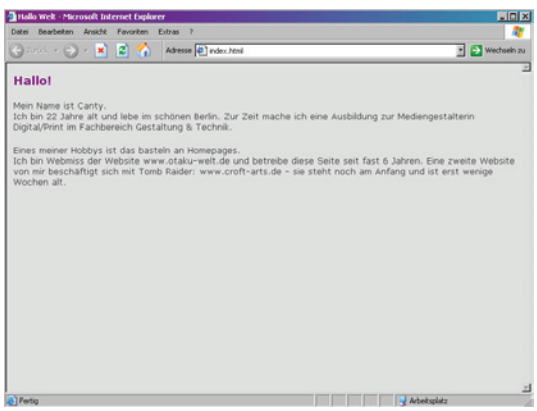
und z.B. grün

HTML-Code:

```
#009933
```

Mit den Hexadezimalcodes können dann Schrift-, Hintergrund-, Rahmen- usw. -farben definiert werden.

Um die **Standardfarben einer HTML-Seite festzulegen**, stehen im BODY-Tag diverse Attribute zur Verfügung, denen Farbwerte zugewiesen werden können:



HTML-Code:

```
<body bgcolor="#e0e0e0" text="#333333" link="#800080" vlink="#4f154f" alink="#800080">
```

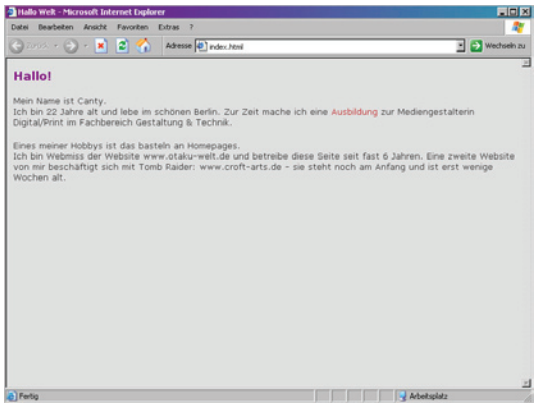
bgcolor (hier hellgrau) bestimmt die Hintergrundfarbe
text (hier fast schwarz) bestimmt die Textfarbe
link (hier violett) bestimmt die Farbe von Links
vlink (hier dunkles violett) bestimmt die Farbe von besuchten Links
alink (hier violett) bestimmt die Farbe vom Link, der gerade angeklickt wird

Innerhalb eines Textes kann die Farbe, die als Standardtextfarbe festgelegt ist, einfach mittels font-Tag überschrieben werden:

HTML-Code:

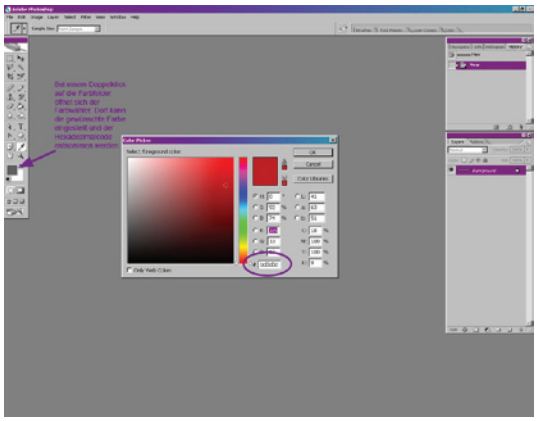
```
<font color="#4f154f"> ... Text ... </font>
```

Ich habe hier einmal als Beispiel das Wort "Ausbildung" hervorgehoben:



Wie kommt man nun zum gewünschten Farbcode? Sicherlich könnte man mittels "ausprobieren" die gewünschte Farbe erraten. Dafür braucht man aber viel Zeit und es erfordert ein gewisses Maß an Erfahrung, die Farbanteile im Kopf zu errechnen.
Einfacher geht der Weg über ein Bildbearbeitungsprogramm wie z.B. Photoshop:

Mit einem Doppelklick öffnet sich der Farbwähler, wo die gewünschte Farbe eingestellt und der entsprechende Code rauskopiert werden kann.



Eine Übersicht mit verschiedensten Hexadezimalcodes ist hier zu finden:
http://www.html-php-mysql.de/generatoren/colors.php

Idealerweise werden alle Farbeinstellungen zentral in einer CSS-Datei festgelegt – dazu werdet ihr hier später mehr erfahren.

6. Hyperlinks

Links sind der wichtigste Bestandteil des Internets. Mit ihnen kommen wir von Seite zu Seite und machen das “WorldWideWeb” zu dem, was es ist.
Hyperlinks können zu internen Seiten, externen Seiten, eMail-Adressen sowie Dokumenten (also Bilder, PDFs usw.) verweisen.

Der grundsätzliche **Aufbau eines Links** ist wie folgt:

HTML-Code:

```
<a href=”Linkziel”>Linktext</a>
```

Als Linktext kann natürlich jeder beliebige Text verwendet werden, Bilder können auch verlinkt werden.
Bei “Linkziel” wird der Pfad oder die URL des verlinkten Dokumentes angegeben.
Man unterscheidet hierbei zwischen **absoluten und relativen Pfaden**.

Absolute Pfade beinhalten die komplette URL, z.B.:

HTML-Code:

```
http://www.otaku-welt.de/homepage-hilfe/tipps-tricks.html
```

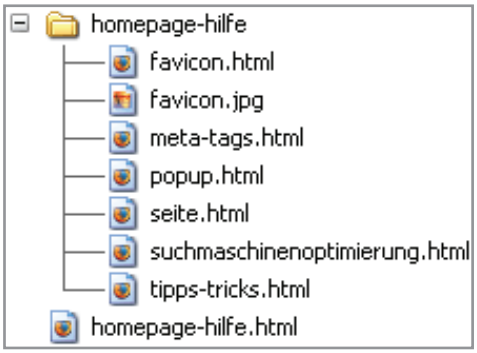
Ein **relativer Pfad** ist wie folgt aufgebaut und ist abhängig vom Ausgangsdokument:

HTML-Code:

```
homepage-hilfe/tipps-tricks.html
```

Ich denke die absoluten Pfade sind eindeutig, auf die relativen werde ich noch etwas genauer eingehen. Zu diesem Zweck habe ich hier mal einen Ausschnitt meiner Dateistruktur skizziert.

Man erkennt in er ersten Ebene den Ordner “homepage-hilfe” sowie die Datei “homepage-hilfe.html”. In der zweiten Ebene, im oben genannten Ordner befinden sich die Dateien “favicon.html”, “favicon.jpg”, “meta-tags.html” usw.



Wird eine Datei mit einem Dokument verlinkt, das sich in der gleichen Ebene / im **gleichen Ordner** befindet, muss nur der Dateiname als Linkziel angegeben werden:

HTML-Code:

```
<a href=”favicon.html”>Linktext</a>
```

Möchte ich von der Seite homepage-hilfe.html auf favicon.html verlinken (also **in einen Ordner**), so muss der relative Pfad so aussehen:

HTML-Code:

```
<a href=”homepage-hilfe/favicon.html”>Linktext</a>
```

Damit ruft der Browser den Ordner “homepage-hilfe” auf und sucht sich dort die Datei raus.

Wenn nun die Seite homepage-hilfe.html von der Datei favicon.html verlinkt werden soll (also **in einen Überordner**), sieht der relative Pfad so aus:

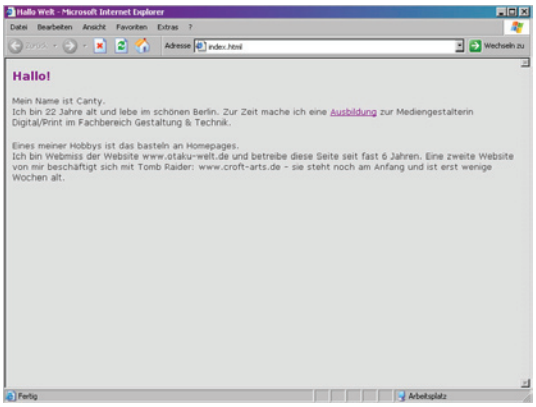
HTML-Code:

```
<a href="../homepage-hilfe.html">Linktext</a>
```

../ ist die Anweisung, in den darüberliegenden Ordner (in unserem Fall also die oberste Ebene) zu springen und dort die Datei aufzurufen. Sollte sich das Dokument mehrere Ordner über der Ausgangsdatei befinden, kann man das ../ natürlich auch mehrmals hintereinander schreiben – für jeden Ordner ein mal:
../../../ wären also 3 Ordner.

WYSIWYG-Editoren editieren diese relativen Pfadangaben automatisch.

Auf meiner Beispielseite habe ich mal das Wort "Ausbildung" beispielhaft mit dem Wikipedia-Artikel verlinkt:

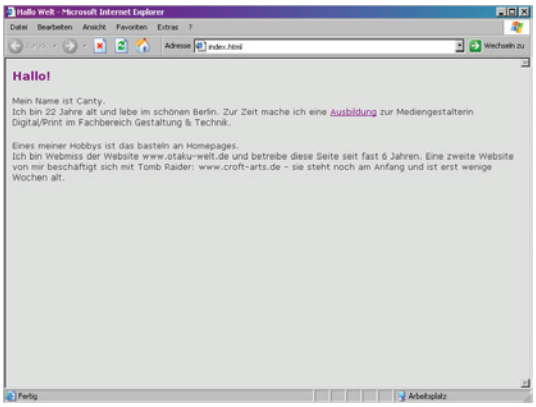


Natürlich kann man auch festlegen, **wie und wo der Link geöffnet wird**. Dazu gibt's das Attribut Target. Folgende Werte können zugewiesen werden:
_blank Verweis öffnet sich im neuen Fenster
_self Verweis öffnet sich im aktuellen Fenster (Standardeinstellung)
_parent Verweis öffnet sich bei Framesets im darüber liegenden Fenster
_top Verweis öffnet sich bei Framesets im obersten Fenster

Soll ein Link also in einem neuen Fenster geöffnet werden, sieht das Ganze so aus:

HTML-Code:

```
<a href="Linkziel" target="_blank">Linktext</a>
```



Zusätzlich zu den “normalen” Links, gibt es 2 interessante und hilfreiche Arten von Links: **eMail-Links** sowie **Links innerhalb einer HTML-Seite**.

Ergänzt man die href-Anweisung um ein **mailto**, wird aus dem Link eine Verknüpfung mit einer **eMail-Adresse**. Wenn dieser geklickt wird, öffnet sich das eMail-Programm des Nutzers - z.B. Outlook oder Thunderbild - und es kann eine eMail verfasst werden.
Ein eMail-Link sieht also so aus:

HTML-Code:

```
<a href=”mailto:name@provider.de”>eMail senden</a>
```

Dieser Link kann erweitert und ein Betreff vorgegeben werden:

HTML-Code:

```
<a href=”mailto:name@provider.de?subject=Betreff”>eMail senden</a>
```

Innerhalb einer HTML-Seite können sogenannte **Anker** definiert und aufgerufen werden. So kann schnell z.B. zurück zum Anfang einer Seite “gesprungen” werden.
An jeder beliebigen Stelle der Seite kann ein Anker gesetzt werden:

HTML-Code:

```
<a name=”name_des_ankers”></a>
```

Wird dieser Tag direkt unter das <body> geschrieben, “springt” man durch den Link ohne die Seite neu laden zu müssen direkt zurück an den oberen Rand der Seite.
Der Link, um zum Anker zu springen sieht dann so aus:

HTML-Code:

```
<a href=”#name_des_ankers”>Linktext</a>
```

Der Name eines Ankers kann natürlich frei vergeben werden (die Raute (#) muss jedoch stehen bleiben). Jedoch sollte auf Großschreibung, Umlaute, Sonder- sowie Leerzeichen verzichtet werden. Vergebt logische Ankernamen, da sie bei Klick in der Adressleiste des Browser angezeigt werden (also kein Schmuddelkram ;)).

7. Listen & Aufzählungen

Listen und Aufzählungen eignen sich ideal, um Informationen anzuordnen.
Außerdem werden Listen benutzt, um eine Seitennavigation zu erstellen – mittels CSS werden diese "gestylt" (dazu aber später mehr).

Grundsätzlich unterscheidet man zwischen **nicht nummerierten Aufzählungen** und **nummerierten Listen**.

Die nicht **nummerierten Aufzählungen** werden mit folgendem Tag umschlossen:

HTML-Code:

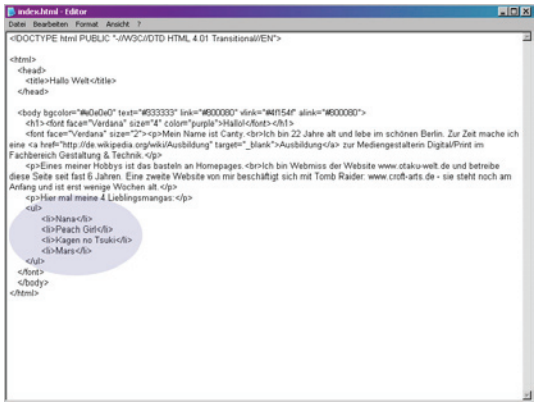
```
<ul> ... Text ... </ul>
```

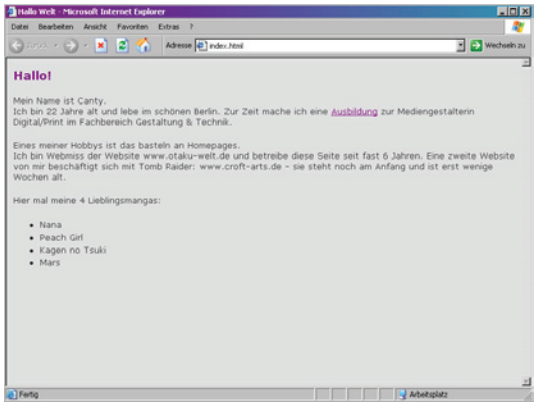
"ul" steht für "unordered list" (engl. unsortierte Liste).
Innerhalb dieses Bereiches wird jeder einzelne Listenpunkt du ein li-Tag ("list item", engl. Listenpunkt) eingeschlossen:

HTML-Code:

```
<li> ... Info ... </li>
```

So sieht also eine nicht nummerierte Liste aus:

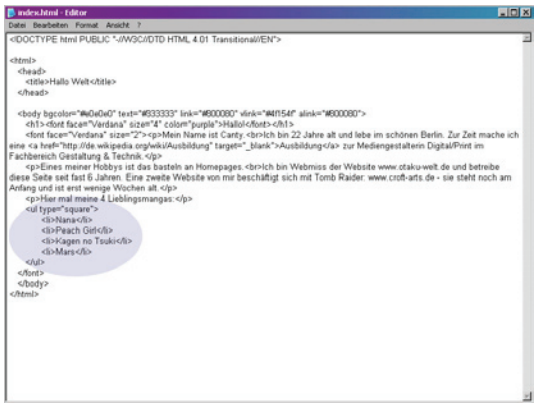


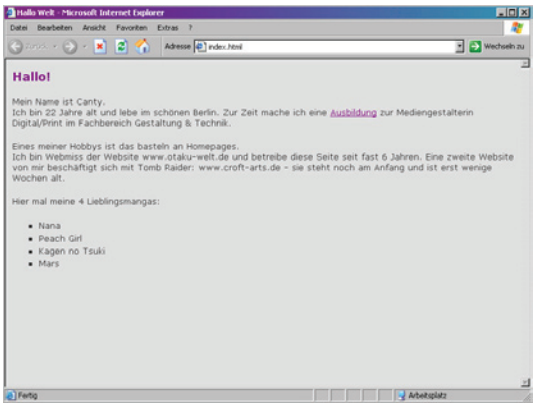


Das **Aufzählungszeichen** ist normalerweise ein gefüllter runder Punkt, kann aber auch anders eingestellt werden. Man verwendet das Attribut "type", das sowohl beim ul- also auch beim li-Tag stehen kann.
Mögliche Werte sind:
"disc" (gefüllter runder Punkt, Standardwert)
"circle" (leerer Kreis)
"square" (quadratischer Punkt)

HTML-Code:

```
<ul type="square">
  <li>Info 1</li>
  <li>Info 2</li>
  <li>Info 3</li>
</ul>
```



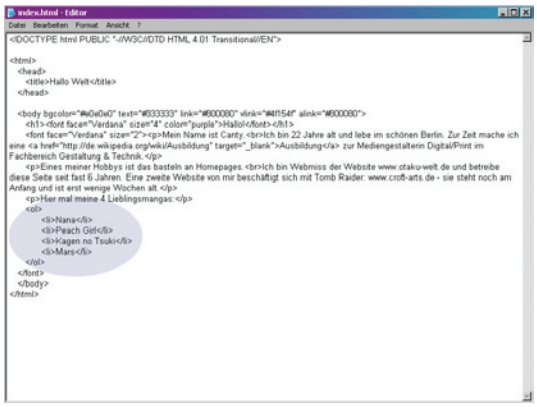


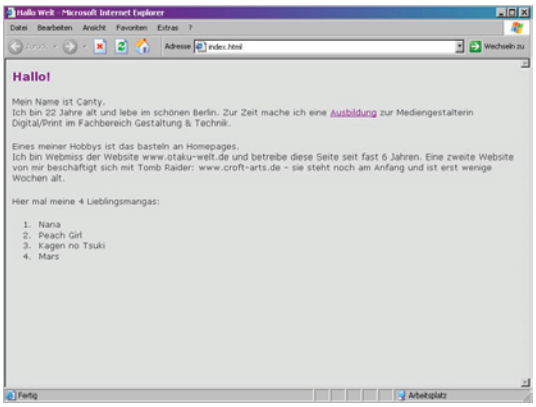
Die **nummerierten Listen** werden von folgendem Tag umschlossen:

HTML-Code:

 ... *Text* ...

“ol” steht für “ordered list” (engl. sortierte Liste).
Auch in dieser Liste werden Listenpunkte durch <li> angegeben.





Auch das ol-Tag kennt das Attribut “type”, mit dem man die **Nummerierungsart** auswählen kann. Als Werte stehen zur Verfügung:
“1” (arabische Ziffern, Standardwert)
“A” (Großbuchstaben)
“a” (Kleinbuchstaben)
“I” (römische Zahlen)
“i” (kleingeschriebene römische Zahlen).
Desweiteren steht das Attribut “**start**” zur Verfügung, mit dessen Wert die Nummerierung beginnt. Will man also eine nummerierte Liste erstellen, die mit der römischen Zahl “III” (3) startet, wird folgender Code benutzt:

HTML-Code:

<ol type=”I” start=”3”> ... *Text* ...

8. Bilder einbetten

Ein wichtiger Bestandteil einer Website sind Bilder. Durch sie kann man einer tristen Seite Leben einhauchen, Texte illustrieren usw.

Wenn ihr eine eigene kleine Homepage habt, bekommt ihr für gewöhnlich auch etwas Speicher, wo ihr Bilder & co hochladen könnt. Wichtig ist, dass ihr nie Bilder anderer Webseiten direkt einbindet, ohne sie vorher auf eurem eigenen Webspace hochzuladen. Das gilt auch für Bilder von Google. Ihr verursacht dem betroffenen Webmaster zusätzliche Kosten, mit großer Wahrscheinlichkeit werdet ihr erwischt und er kann euch anzeigen – Bilder querzuverlinken ist nämlich eine Straftat.
Zusätzlich solltet ihr bei fremden Bildern auch immer das Urheberrecht beachten – also den Webmaster der Seite fragen, ob ihr das Bild verwenden dürft.

Bilder werden auf einem Webserver geladen und sind von dort aus durch eine URL wie jede andere Datei abrufbar. Sie liegen in den **Dateiformaten JPG, PNG oder GIF** vor. Jedes Format hat seine Vor- und Nachteile – Dazu gibt es später noch ein Einzeltutorial. Jetzt sei nur kurz und knapp gesagt:
JPG – Fotos mit vielen Farben
PNG – Bilder mit Transparenz und Halbtransparenz
GIF – flächige Grafiken (max. 256 Farben), Transparenz und Animationen

Mit folgendem Tag kann man in eine Website **Bilder einbetten**:

HTML-Code:

```
<img src="URL_des_Bildes">
```

Für die URL besteht die Möglichkeit absolute oder relative Pfade anzugeben – ihr erinnert euch, das hatten wir schon mal in Kapitel 6 (Hyperlinks).

Für jedes verwendete Bild sollte ein **“alt”-Attribut** definiert werden, in der der **Inhalt des Bildes** mit wenigen Worten beschrieben wird:

HTML-Code:

```
<img src="URL_des_Bildes" alt="Beschreibung_des_Bildes">
```

Warum? Weil dieser Text angezeigt wird, wenn das Bild noch nicht geladen wird oder nicht angezeigt werden kann. Außerdem können sich sehbehinderte Menschen Webseiten vorlesen lassen – ist ein Alternativtext vorhanden, wird dieser auch vorgelesen und die betroffene Person kann etwas damit anfangen. Dazu kommt, dass Google bei der Positionierung der Suchergebnisse u.a. auch die Alt-Attribute von Bildern ausliest.
Bilder, die zum Layouten einer Seite dienen, z.B. Headerbild oder leere Gifs, benötigen keine Beschreibung und bekommen ein leeres Alt-Attribut zugewiesen: alt=””.

HTML-Code:

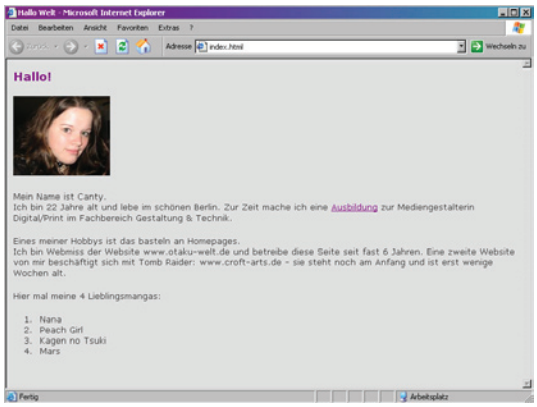
```
<img src="URL_des_Bildes" alt="">
```

Zusätzlich kann und sollte auch die **Größe eines Bildes** angegeben werden:

HTML-Code:

```
<img src="URL_des_Bildes" height="Höhe_in_Pixeln" width="Breite_in_Pixeln">
```

So wird der Ladeprozess beschleunigt und Texte können sich schon aufbauen.
Eine Skalierung des Bildes (vergrößern / verkleinern / verzerren) sollte nicht über diese beiden Attribute gemacht werden, da das Ergebnis qualitativ sehr unzufriedenstellend ist. Besser ist es, das Bild vorher in einem Bildbearbeitungsprogramm (Photoshop, Gimp) in die gewünschte Größe zu bringen.



Mit dem Attribut “align” kann ein **Bild positioniert** werden:

HTML-Code:

```
<img src="URL_des_Bildes" align="left">
```

Es stehen euch die Werte “left” (Standardeinstellung) und “right” zur Verfügung.

Durch das Attribut “border”, kann auch die **Breite des Rahmens** festgelegt werden:

HTML-Code:

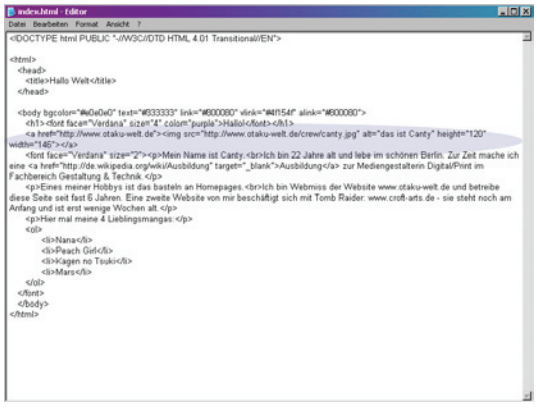
```
<img src="URL_des_Bildes" border="2">
```

erzeugt also einen Rand von 2 Pixeln.
border=”0” sorgt dafür, dass es gar keinen Rahmen gibt. Der Rahmen hat die Farbe, die auch der Text im Dokument hat.

Um ein **Bild zu verlinken**, z.B. für einen Banner, wird einfach anstatt eines Linktextes der Code eines Bildes eingefügt:

HTML-Code:

```
<a href="URL_der_Seite"><img src="URL_des_Bildes"></a>
```

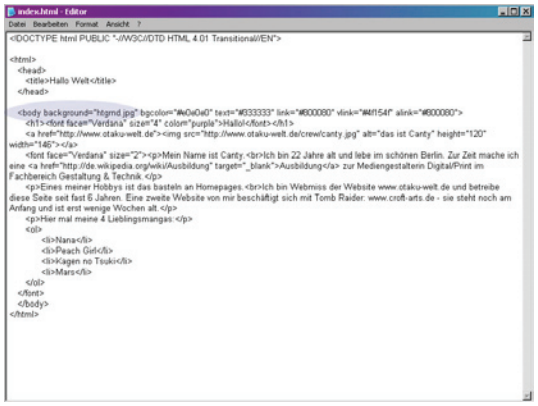


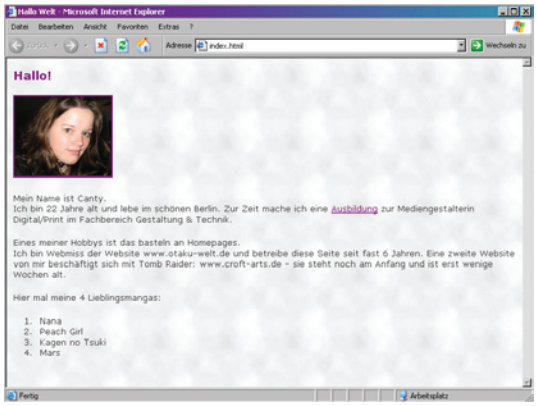
Natürlich kann man einer HTML-Seite auch ein **Hintergrundbild** verpassen. Dafür muss der BODY-Tag durch das Attribut “background” ergänzt werden:

HTML-Code:

```
<body background="URL_des_Bildes">
```

Das Bild wird automatisch horizontal und vertikal wiederholt, bis das ganze Browserfenster befüllt ist. Mit CSS besteht die Möglichkeit, diese Wiederholung zu beeinflussen (nur horizontal/vertikal, gar nicht wiederholen) und das Hintergrundbild überall im Fenster (oben/unten/links/rechts usw.) auszurichten.





9. Tabellen

Tabellen sind durch Zweckentfremdung ein wichtiges Gestaltungsmittel geworden: viele Leute bauen ihre Webseitenlayouts mit Tabellen, wobei diese nicht unbedingt dafür gemacht sind. Sicherlich geht es, aber mit einem CSS-Layout fährt man doch sehr viel besser (schnellere Ladezeiten, Layout jederzeit durch wenige Handgriffe komplett veränderbar usw.).
Der eigentliche Sinn einer Tabelle ist die (tabellarische) Anordnung von Daten und Informationen. Dies können Texte, Bilder, Links usw. sein.

Eine Tabelle wird **von folgenden Tags umschlossen**:

HTML-Code:

```
<table> ... </table>
```

Sie besteht aus mehreren Zeilen (tr → table row = Tabellenzeile), welche in Zellen (td → table data = Tabellendaten) aufgeteilt sind. In den Zellen steht dann der eigentliche Inhalt.

Eine Tabelle ist also wie folgt aufgebaut:

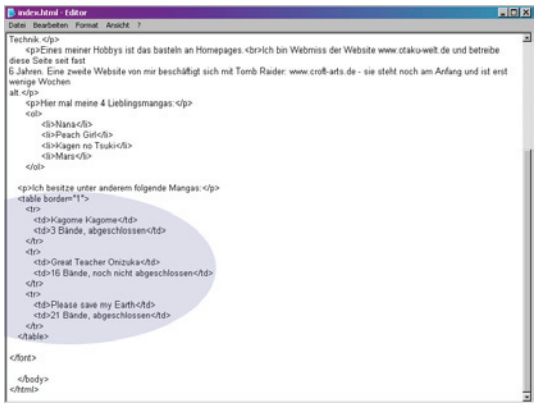
HTML-Code:

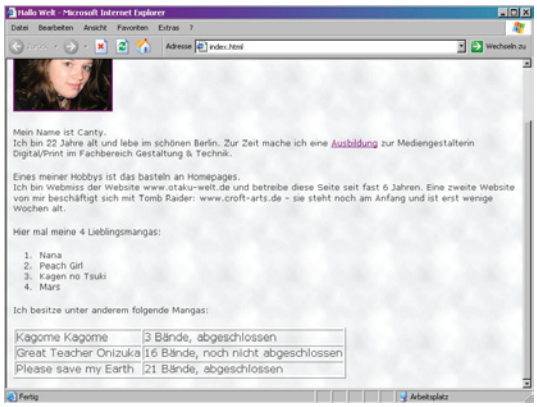
```
<table>
  <tr>
    <td> … Text in Zeile 1, Spalte 1 … </td>
    <td> … Text in Zeile 1, Spalte 2 … </td>
  </tr>
  <tr>
    <td> … Text in Zeile 2, Spalte 1 … </td>
    <td> … Text in Zeile 2, Spalte 2 … </td>
  </tr>
</table>
```

Damit ein Gitternetz sichtbar wird, muss dem table-Tag noch das Attribut “border” mit dem Wert 1 (je höher der Wert, umso dicker wird der Tabellenrahmen) hinzugefügt werden, dass es so aussieht:

HTML-Code:

```
<table border=”1”>
    ...
</table>
```





Bei Tabellen stehen uns **zahlreiche Attribute** zur Verfügung, um diese nach unseren Wünschen zu formatieren.

Folgende Möglichkeiten gibt es, die **ganze Tabelle mittels <table>-Tag** anzupassen:

align=”*left*”
richtet die gesamte Tabelle innerhalb des Browserfensters aus
Werte: “left” oder “right” oder “center”

width=”*500*”
gibt die Breite der Tabelle in Px an, Prozentangaben sind auch möglich; benötigt der Inhalt einer Tabelle mehr Platz, wird der Wert ignoriert
Werte: z.B. “400” oder “80%”

height=”*800*”
gibt die Höhe der Tabelle in Px an, Prozentangaben sind auch möglich; benötigt der Inhalt einer Tabelle mehr Platz, wird der Wert ignoriert
Werte: z.B. “800” oder “90%”

border=”*1*”
erzeugt einen Rahmen um die Tabelle; bei einer unsichtbaren Layout-Tabelle, sollte immer border=”0” gesetzt sein
Werte: z.B. “4”

bordercolor=”*#000000*”
6-stelliger Farbwert (siehe Kapitel 5) für den Rahmen; es ist empfehlenswerter, den Rahmen mit CSS zu stylen
Werte: z.B. “#ffffff” oder “blue”

bgcolor=”*#ffffff*”
6-stelliger Farbwert (siehe Kapitel 5) für den Hintergrund der Tabelle, überdeckt die Hintergrundfarbe der Seite
Werte: z.B. “#444444” oder “red”

background=”*URL_des_Bildes*”
fügt ein Hintergrundbild ein
Werte: absoluter oder relativer Pfad zum Bild

cellpadding="*10*"
gibt den Abstand vom Text / Bild des Zelleninhaltes zum Rand der Zelle in Px an
Werte: z.B. "50" oder "5"

cellspacing="*3*"
gibt die Breite der Zellränder in Px an
Werte: z.B. "10" oder "40"

Natürlich kann man auch **einzelne Zeilen im <tr>-Tag** anpassen:

align="*left*"
richtet die Inhalte einer Zeile horizontal aus
Werte: "left" oder "right" oder "center" oder "justify"

valign="*top*"
richtet die Inhalte einer Zeile vertikal aus
Werte: "top" oder "middle" oder "bottom"

bgcolor="*#ffffff*"
6-stelliger Farbwert (siehe Kapitel 5) für den Hintergrund der Zeile, überdeckt die Hintergrundfarbe der Seite
Werte: z.B. "#444444" oder "red"

Ebenso kann man auch **einzelne Zellen mit <td>-Tag** anpassen:

align="*left*"
richtet die Inhalte einer Zelle horizontal aus
Werte: "left" oder "right" oder "center" oder "justify"

valign="*top*"
richtet die Inhalte einer Zelle vertikal aus
Werte: "top" oder "middle" oder "bottom"

width="*200*"
gibt die Breite der Zelle in Px an, Prozentangaben sind auch möglich; benötigt der Inhalt einer Zelle mehr Platz, wird der Wert ignoriert
Werte: z.B. "300" oder "60%"

height="*600*"
gibt die Höhe der Zelle in Px an, Prozentangaben sind auch möglich; benötigt der Inhalt einer Zelle mehr Platz, wird der Wert ignoriert
Werte: z.B. "445" oder "40%"

colspan="*3*"
definiert, dass sich eine Zelle über die angegebene Anzahl von Spalten der Tabelle erstrecken soll
Werte: z.B. "3"

rowspan="*5*"
definiert, dass sich eine Zelle über die angegebene Anzahl von Zeilen der Tabelle erstrecken soll
Werte: z.B. "4"

Gerade Tabellen sind für einen HTML-Neuling immer sehr verwirrend. Da helfen WYSIWYG-Editoren meist Wunder und erleichtern die Erstellung ungemein :)
Hier ist wohl "üben üben üben" angesagt, so schwer ist es nicht.

10. Frames

Frames erlauben es, mehrere HTML-Dokumente gleichzeitig in einem Browserfenster anzuzeigen. Ein typischer Aufbau ist eine feste Navigation links und der Contentbereich rechts. Beide Teile sind eigenständig steuer- und scrollbar.
Das Benutzen von Frames ist unter Webmastern sehr umstritten, da gibt es mächtig viele Diskussionen. Frames haben ihre Vorteile, aber auch entscheidende Nachteile. Da das den Rahmen hier sprengen würde, werde ich das Thema “Frames: Pro – Contra” in einem Mini-Tutorial gesondert behandeln.

Bei Frames unterscheidet man zwischen ganzen **Framesets** (spezielle Seiten, in denen mehrere Dateien laden, die sonst keinen Inhalte enthalten) und **iFrames** (liegen eingebettet in normalen HTML-Seiten und laden eine andere HTML-Seite in die bestehende hinein; hier kann um das iFrame bereits Inhalt bestehen).

Um das **Layout einer Seite als Frameset** aufzubauen, werden mindestens 3 Dateien benötigt:
die Index-Seite, in der die Frames eingebettet sind, 2 Dateien, die in dem Frameset geladen werden sollen (z.B. eine navigation.html und eine inhalt.html). Natürlich können auch weitere Dateien geladen werden.

Der **Aufbau einer Frameset-Seite** unterscheidet sich ein bisschen von der Standard-HTML-Seite. Wir erinnern uns, dies ist der Aufbau einer HTML-Seite:

HTML-Code:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Titel der Webseite</title>
  </head>

  <body>
    Inhalt der Webseite
  </body>
</html>
```

Der Aufbau einer Frameset-Seite sieht wie folgt aus:

HTML-Code:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN">
<html>
  <head>
    <title>Titel der Webseite</title>
  </head>

  <frameset>
    Inhalt des Framesets
  </frameset>
</html>
```

2 Dinge müssen beim Aufbau von Framesets beachtet werden:

- der Doktype wird verändert (statt “Transitional” wird “Frameset” verwendet)
- statt des <body>-Tags gibt es ein **<frameset>-Tag**, das die Inhalte einschließt; zusätzlich kann aber innerhalb des <frameset>-Tags ein <body> eingerichtet und mit Inhalten (Seitenbeschreibung, Sitemap usw.) befüllt werden – diese Seite wird dann angezeigt, wenn der Browser des Users keine Frames unterstützt (ist nur bei sehr alten Browsern der Fall)

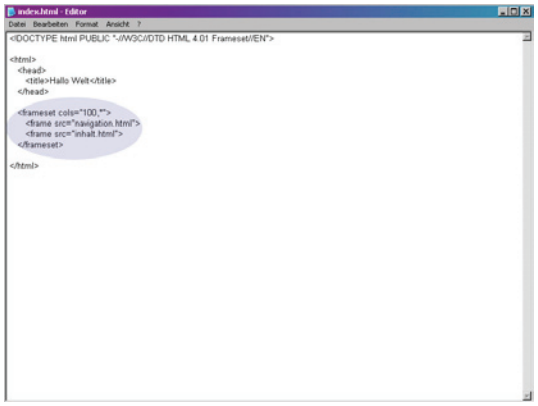
In dem <frameset>-Tag wird entweder eine horizontale (mit dem Attribut “cols”) oder vertikale (mit dem Attribut “rows”) Unterteilung des Browserfensters vorgenommen. Die Werte des Attributs werden durch Kommas getrennt und sind letztendlich eine Liste mit den jeweiligen Frame-Maßen in Pixeln (mind. 1 Frame braucht eine bestimmte Breite, das Frame mit variabler Breite bekommt den Wert “*”) oder Prozent (müssen zusammen immer 100% ergeben).
Zwischen <frameset> ... </frameset> stehen die <frame>-Tags, welche die zu ladenden Frames definieren und die ensprechenden HTML-Dokumente laden. Zusätzlich kann innerhalb eines <frameset>-Tags ein weiteres <frameset>-Tag stehen – so kann man verschachtelte Framesets erstellen.

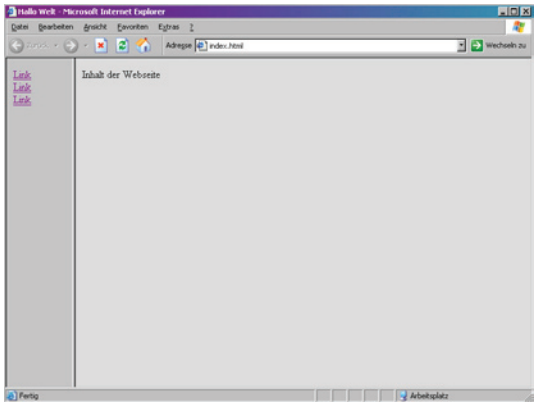
Hier erstmal ein **klassisches Beispiel für ein Frameset**:

HTML-Code:

```
<frameset cols="100,*">
  <frame src="navigation.html">
  <frame src="inhalt.html">
</frameset>
```

Dieser Code erstellt ein 2-Spaltiges Frameset, bestehend aus Navigation und Inhalt. Die Navigation hat eine Breite von 100 Pixeln, der Inhaltsbereich passt sich an die Breite des Browserfensters an.
In die beiden Frames werden die Dateien navigation.html und inhalt.html geladen.



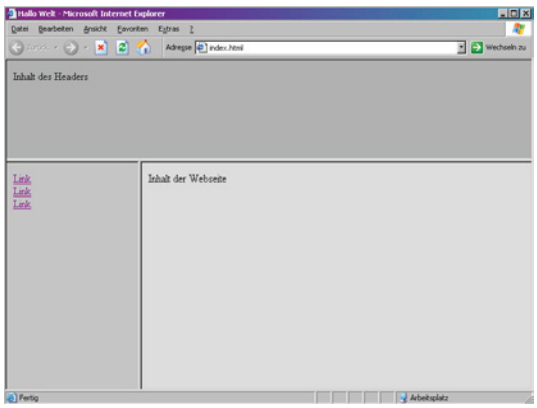


Mit Hilfe von verschalteten Framesets können **komplexere Framesets** erstellt werden.
Häufig verwendet wird auch folgendes Beispiel:

HTML-Code:

```
<frameset rows="150,*">
  <frame src="oben.html">
    <frameset cols="200,*">
    <frame src="navigation.html">
    <frame src="inhalt.html">
  </frameset>
</frameset>
```

Hier haben wir ein 2-Zeiliges Frameset. In der oberen Zeile befindet sich ein Header: oben.html. In der unteren ist ein weiteres Frameset eingebunden, welches ein 2-Spaltiges Frameset erzeugt: navigation.html und inhalt.html.



Jeder Benutzer kann mit der Maus die **Größe des Frames zurechtziehen**. Möchte man das unterbinden, kann man folgendes Attribut dem <frame>-Tag zuweisen:

HTML-Code:

```
noresize="true"
```

Um die **Linien zwischen den Frames komplett auszublenden**, fügt man folgende Attribute und Werte in das <frameset>-Tag ein:

HTML-Code:

```
border="0" frameborder="no"
```

Das Ganze sieht dann zum Beispiel so aus:

HTML-Code:

```
<frameset cols="100,*" border="0" frameborder="no" >
  <frame src="navigation.html" noresize="true">
  <frame src="inhalt.html" noresize="true">
</frameset>
```

Standardmäßig öffnet sich ein **Link innerhalb eines Framesets** im gleichen Frame. Das ist natürlich, z.B. bei Navigationslinks, nicht erwünscht.
Um das zu vermeiden, müssen die einzelnen Frames einen Namen erhalten:

HTML-Code:

```
<frame src="URL_der_Seite" name="Name_des_Frames">
```

Soll nun der **Link eine Datei in einem bestimmten Frame öffnen**, muss dies als Wert im target-Attribut des Hyperlinks angegeben werden:

HTML-Code:

```
<a href="URL_des_Links" target="Name_des_Fensters"> ... Linktext ... </a>
```

Bei den Frames mit der Navigation, kann im <head>-Bereich der Webseite ein **Standard-Target** definiert werden:

HTML-Code:

```
<base target="Name_des_Fensters">
```

Damit werden alle Links auf dieser HTML-Seite immer im angegebenen Frame geöffnet – auch ohne die spezielle Target-Angabe.

Ein **iFrame** liegt, wie bereits oben beschrieben, in einer normalen HTML-Seite eingebettet und läd eine andere HTML-Seite in die bestehende hinein.
Um ein iFrame zu erzeugen, muss an der gewünschten Stelle folgender Code eingefügt werden:

HTML-Code:

```
<iframe src="URL_der_Seite" name="Name_des_Frames" width="300" height="500"></iframe>
```

Auch bei einem iFrame wird die Quelle (eingebundene Seite) angegeben und ein Name kann zugewiesen werden. Zusätzlich sollte eine Höhen- und Breitenangabe in Pixeln oder Prozent definiert werden.

11. Formulare

Mit HTML kann man **Formulare erstellen**, in denen der User Texte eingeben, Buttons klicken und aus Menüs auswählen kann. Es stehen euch eine Vielzahl von Formularelementen zur Verfügung: Text- und Passwortfelder, Radiobuttons, Checkboxen usw. Dazu kommen unsichtbare Elemente, die das Formular steuern und z.B. eine Bestätigungsseite anzeigen lassen.
Per Knopfdruck wird der Inhalt des Formulares dann an den Empfänger gesendet. Dies passiert für gewöhnlich über ein serverseitiges Script (z.B. CGI), dass euer Webhoster zur Verfügung stellt. Alternativ können die Formulardaten auch einfach per eMail-Programm gesendet werden.

Grundsätzlich steht jedes Formular zwischen folgenden Tags:

HTML-Code:

```
<form action="URL" method="Versandmethode">
  ... Formularelemente ...
</form>
```

Als URL des **Action**-Attributes wird meist die **Adresse des Server-Scriptes** angegeben. Die ist von Webhoster zu Webhoster unterschiedlich und muss entsprechend beim jeweiligen erfragt werden.
Alternativ kann man auch eine **eMail-Adresse als Wert des Action-Attributes** angeben. Diese wird dann wie in einem eMail-Hyperlink als “mailto:name@provider.de” eingesetzt.

Letztere Vorgehensweise ist allerdings aus mehreren Gründen **nicht zu empfehlen**:
- klicken des “Senden”-Buttons öffnet letztentlich nur das eMail-Programm des Absenders (Outlook, Thunderbird o.ä.) und sendet die Formulardaten als eMail weg -> hat der Benutzer gar kein eMail-Programm installiert (z.B. in einem Internetcafé), kann er das Formular nicht absenden
- der Internet Explorer, auch in neueren Versionen, versteht das Action-Attribut falsch und öffnet einfach ein leeres eMail-Fenster – ohne Formulardaten
Habt ihr also keine Adresse des Server-Scriptes zur Verfügung (das ist z.B. bei einigen Gratis-Hostern der Fall), nutzt besser einen der zahlreichen kostenlosen Forumlar-Anbieter wie Onlyfree.de oder OneTwoMax.de – damit fahrt ihr in dem Fall besser.

Das Attribut **Method** bezeichnet die Versandmethode, mit der die Daten übertragen werden sollen. Als Werte stehen euch “get” und “post” zur Verfügung.
Post ist der zu empfehlende Wert – er versendet die Daten separat und wird bei jedem Formular, das mit einem serverseitigen Script arbeitet angegeben. Außerdem wird “Post” auch als passende Methode für den eMail-Versand benutzt.
Get sollte eher bei Suchmasken (z.B. wie in Google) verwendet werden. Es hat da den Vorteil, dass der Benutzer seine Suchanfrage direkt bei den Favoriten abspeichern kann, da “Get” die Formulardaten an die Action-URL anfügt. Für Kontaktformulare ist “Get” nicht geeignet.

Zwischen <form> und </form> können nun unterschiedliche **Formularelemente** definiert werden. Beispiele dafür habe ich bereits oben genannt.
Die meisten Formularelemente werden mit den **<input>-Tag** erzeugt. Das Attribut “type” kennt viele verschiedene Werte, die dann entsprechend einen Radiobutton, Textfeld o.ä. erzeugen
Hier mal der **Grundaufbau eines input-Tags**:

HTML-Code:

```
<input type="Elementtyp" name="Elementname" value="Wert">
```

Type definiert die Art des Formularelementes (z.B. “radio” → Radiobutton oder “submit” → Senden-Button). Name und Value bilden ein Paar, dass als Name=Value mit den Formulardaten versendet wird. Außerdem haben einige Elementtypen Spezialattribute, dazu aber später mehr.

Euch stehen viele verschiedene **input-Tags** zur Verfügung (hier die wichtigsten):

Textfeld:

HTML-Code:

```
<input type="text" name="..." size="..." maxlength="...">
```

Hier kann ein Text eingegeben werden (z.B. ein Name, URL, eMail-Adresse). Bei Bedarf kann mit "size" die Feldbreite in Zeichen und mit "maxlength" die maximale Eingabelänge festgelegt werden. Wollt ihr dies nicht vorschreiben, lasst die entsprechenden Attribute einfach weg. Ebenfalls optional ist hier der Parameter "value" – er würde dafür sorgen, dass in dem Textfeld bereits ein voreingetragener Text steht.

Radiobutton:

HTML-Code:

```
<input type="radio" name="..." value="...">
```

Radiobuttons ermöglichen dem Nutzer die Auswahl einer Option aus mehreren Alternativen (Single-Choice). "Name" benennt die Gruppe, zu der ein Radiobutton gehört (bei mehreren Buttons, die zur gleichen Gruppe gehören muss "name" also gleich sein) und "value" ist der Inhalt des Buttons. Der Text, mit dem der Button beschriftet werden soll, wird als einfacher HTML-Text vor oder hinter den Button gesetzt. Der zusätzliche Parameter checked="true" bewirkt, dass der Button bereits ausgewählt ist, wenn das Formular aufgerufen wird.

Checkbox:

HTML-Code:

```
<input type="checkbox" name="..." value="...">
```

Mit Checkboxen gibt man dem Benutzer die Möglichkeit, mehrere Optionen an- und / oder abzuwählen.
"Name" benennt wieder die Gruppe, "value" den Inhalt der Checkbox. Auch hier gibt es den zusätzlichen Parameter checked="true".

Passwortfeld:

HTML-Code:

```
<input type="password" name="..." size="..." maxlength="...">
```

Das Passwortfeld funktioniert genau wie das Textfeld. Allerdings werden die getippten Zeichen verschlüsselt als Sternchen (***) angezeigt.

Senden-Button:

HTML-Code:

```
<input type="submit" value="...">
```

Eines der wichtigsten Input-Tags, da mit diesem Button der Formularinhalt an das angegebene Script bzw. an die angegebene eMail-Adresse gesendet wird. Der Text, der in "value" definiert wird, erscheint auf dem Button als Beschriftung.

Reset-Button:

HTML-Code:

```
<input type="reset" value="...">
```

Dieser Button setzt alle Formulardaten zurück. "Value" definiert wieder die Beschriftung des Buttons. Ich persönlich genieße diesen Button mit Vorsicht, da einige Benutzer ihn versehendlich klicken (statt "senden") – das kann jedem mal passieren. Doch leider sind dann alle Formulareingaben gelöscht und man muss von vorn anfangen.

Unsichtbare Felder:

HTML-Code:

```
<input type="hidden" name="..." value="...">
```

Dies sind keine Eingabefelder, die vom Benutzer ausgefüllt werden können, sie werden auch nicht vom Browser angezeigt. Es sind eher festgelegte Angaben, die zusammen mit den eingegebenen Formulardaten an den Empfänger gesendet werden. Sinnvoll ist das zum Beispiel für die Übersichtlichkeit bei mehreren Formularen auf einer Site: man kann dem Formular einen Namen geben, der mit dem Senden übertragen wird. Außerdem können Weiterleitungen zu HTML-Seiten eingerichtet werden, die nach erfolgreichem senden des Formulars angezeigt werden usw.

Neben den input-Tags gibt es noch **2 weitere wichtige Formularelemente**:

Auswahlmenüs bieten die Möglichkeit aus einer Liste eine oder mehrere Optionen auszuwählen. Grundsätzlich sind sie wie folgt aufgebaut:

HTML-Code:

```
<select name="..." size="...">
  <option value="..."> 1. Option </option>
  <option value="..."> 2. Option </option>
</select>
```

"Name" ist wie immer der Name des Formularabschnittes, mit "size" wird die Anzahl der sichtbaren Zeilen angegeben (ist diese geringer als die Anzahl der Optionen, erscheint ein Scrollbalken). Wird "size" auf 1 gesetzt, entsteht ein PullDown-Menü (Sonderfall). Das zusätzliche Attribut multiple="true" (bei select) ermöglicht eine Mehrfachauswahl mit [Strg] – das sollte dem Benutzer mitgeteilt werden. Das <option>-Tag beinhaltet das Attribut "value", welches wie immer den zu sendenden Wert beinhaltet. Innerhalb des <option>-Containers steht dann die ausformulierte Option, welche der Besucher auswählen kann. Natürlich kann man beliebig viele Optionen angeben. Zusätzlich existiert hier der Parameter "selected" (ohne Wert), welcher dafür sorgt, dass die entsprechende Option vorausgewählt ist.

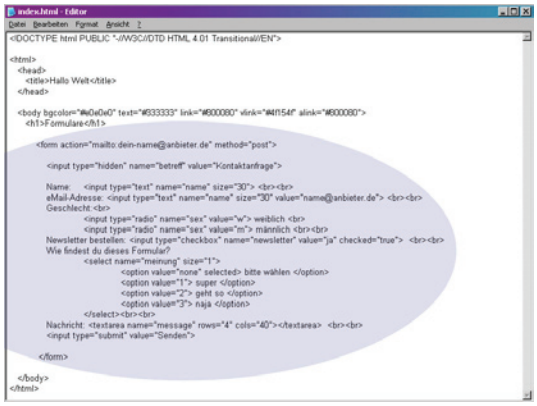
Mehrzeilige Textbereiche (Eingabefelder) sind für Nachrichtenfelder geeignet. Sie entstehen über das folgende Tag:

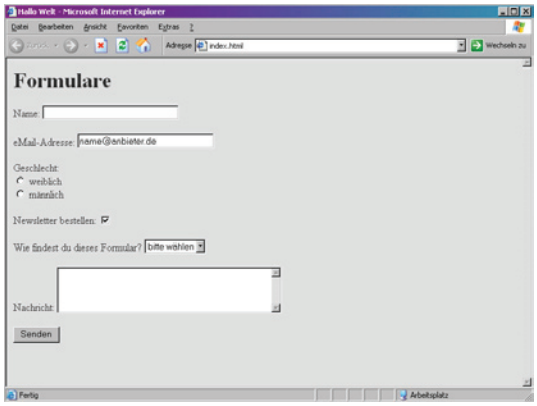
HTML-Code:

```
<textarea name="..." rows="..." cols="..."></textarea>
```

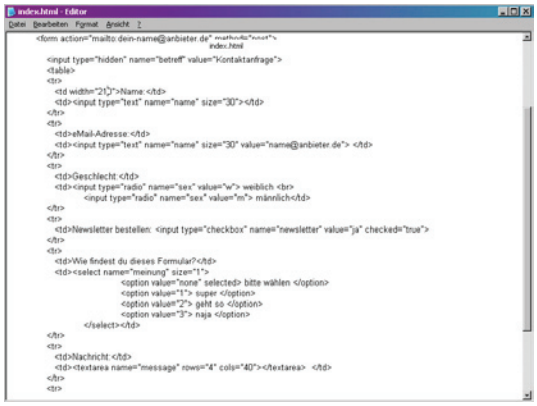
"Rows" definiert die Höhe in Zeilen, "cols" die Breite in Zeichen. Schreibt ihr zwischen dem öffnenden und schließendem Tag Text, erscheint dieser als vorgefertigter Text im Eingabefeld.

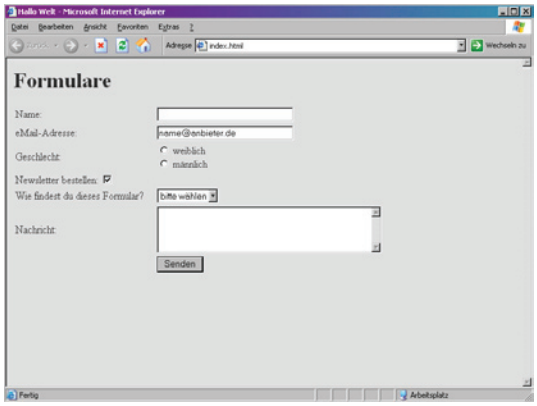
Ich habe hier mal ein **beispielhaftes Formular** erstellt, bei dem ich einige der oben beschriebenen Tags verwende:





Allgemein hat es sich “eingebürgert”, dass Formulare in Tabellen geschrieben werden. So wird das ganze gleich sehr viel **übersichtlicher**:





12. Vermischtes

In diesem Kapitel will ich ein paar Details erklären, die zu klein wären, um damit ein ganzen Kapitel zu füllen.

Sonderzeichen
HTML versteht keine Sonderzeichen wie “ä, ö, ü, ß, &” usw. Dafür gibt es spezielle Codes, die stattdessen in den Quelltext eingegeben werden müssen. Eine sehr gute Übersicht mit den gängigsten Sonderzeichen und Umlauten ist hier zu finden: Selfhtml
Die meisten Sonderzeichen besitzen 2 Schreibweisen: Name in HTML und als Unicode. Beide können gleich verwendet werden.
Viele der Sonderzeichen wird man nie benutzten, man muss diese Tabellen deshalb auf keinen Fall auswendig lernen (“Man muss nicht alles wissen, man muss nur wissen wo es steht” Zitat von meiner LK Biologielehrerin aus der Schulzeit).
Ich selber benutze, neben den Umlauten häufig folgende Sonderzeichen: Gedankenstrich und geschütztes Leerzeichen.
Der Gedankenstrich (auch Halbgeviert genannt) ist in vielen dieser Sonderzeichenlisten leider nicht enthalten, obwohl ich ihn recht wichtig finde. Ich benutze ihn bevorzugt in Überschriften, im “normalen” Fließtext kann man ihn auch durch ein Minuszeichen ersetzen (es sei denn, man ist nun typographievernarrt und legt sehr viel Wert auf solche Details). **Der Unicode des Gedankenstrichs** lautet: **–**

Ein **geschütztes Leerzeichen** (auch erzwungenes Leerzeichen genannt) bewirkt, dass 2 Worte nicht getrennt werden, wenn eine Zeile umbricht. Beispiel: bei den Worten “Windows Vista” ist ein Umbruch nicht erwünscht. Deshalb ersetzt man das normale Leerzeichen im HTML-Code durch den Code ** ** (non breaking space):

HTML-Code:

```
Windows Vista
```

Trennlinien
Eine horizontale Trennlinie lässt sich mit dem Tag **<hr>** (“horizontal ruler”) einfügen. Die Breite der Linie lässt sich über das Attribut “width” in Pixeln oder Prozent angeben. Die Höhe wird mit dem Attribut “size” in angegeben (Pixel). Mit “align” lässt sich die Linie wie üblich links, rechts oder zentriert ausrichten – Standardwert ist hier “center”. Desweiteren gibt es das Attribut noshade=”true”, welches bei Bedarf den 3D-Schatten unterdrückt.
Eine 1px hohe Trennlinie ohne Schatten, die über die gesamte Fensterbreite verläuft halt also beispielsweise folgenden Code:

HTML-Code:

```
<hr width=”100%” size=”1” noshade=”true”>
```

Span
Das HTML-Element ist ein sogenanntes “Inline-Element”. Es erzeugt, im Gegensatz zu z.B. <p> keinen Absatz. Dieses Element wird in Verbindung mit CSS sehr interessant. So können z.B. einzelne Worte oder Passagen besonders hervorgehoben werden. Im CSS-Teil dieses Kurses gehe ich genauer darauf ein.

Div
Dieses HTML-Element ist ein “Block-Element”. Es erzeugt einen Block, der beliebig gestylt, positioniert und mit Inhalt gefüllt wird. Mit dem -Tag erstellt man mit CSS Layouts. Auch dazu gibt es später nähere Infos.

13. CSS – Einstieg (Syntax, Definition, Selektoren)

CSS steht für „**Cascading Style Sheets**“ und gehört nicht direkt zu HTML. Es ist eine Formatierungssprache, die HTML ergänzt.
Der **Sinn von Stylesheets** ist, den Inhalt einer Webseite vom Design zu trennen: das ermöglicht mehr Gestaltungsmöglichkeiten, eine bessere Angepasstheit an Suchmaschinen, einen schlanken Quelltext und – besonders für Webdesigner vorteilhaft – mit wenigen Handgriffen kann das komplette Design einer Website überarbeitet und verändert werden.
Ein **weiterer Vorteil** besteht darin, dass wiederkehrende Elemente (z.B. Schriftarten, Überschriftenformatierung usw.) **zentral definiert** werden – so entfallen die ganzen <font>-Tags usw. im HTML.
Desweiteren ist es mit CSS möglich, für **verschiedene Ausgabemedien** (Monitor, Beamer, Drucker, PDA etc.) unterschiedliche Darstellungen vorzugeben – besonders interessant ist hier die Ausgabe für den Drucker (Header, Navigationen usw. sollen ja schließlich nicht mitgedruckt werden – dazu gibt es später ein Einzel-Tutorial).

Zusammenfassend kann man also sagen: mit CSS kann man HTML-Elemente (Tabellen, Absätze, Container, Listen etc. pp.) Aussehen verleihen (Hintergrund, Farben, Rahmen, Position, Überlappung, Abstand...). Dieses Aussehen wird z.B. in einer zentralen Datei definiert und kann so auf beliebig viele HTML-Seiten angewendet werden.

Syntax von CSS

Der grundlegende **Aufbau einer CSS-Anweisung** unterscheidet sich vom Aufbau eines HTML-Tags:

CSS-Code:

```
selektor { attribut: wert; }
```

Der „Selektor“ wählt das gewünschte Element aus (er „selektiert“), das bearbeitet werden soll (z.B. alle h1-Überschriften). Einem Attribut (z.B. font-size) wird dann ein Wert zugewiesen (bei font-size z.B. „10pt“).
Natürlich kann ein Element/Selektor mehrere Attribute und Werte haben. Sie werden durch das Semikolon getrennt.

Um in einer CSS-Datei nicht den Überblick zu verlieren, schreibt man die Anweisungen untereinander:

CSS-Code:

```
selektor {
  attribut: wert;
  attribut: wert;
  attribut: wert;
}
```

Definieren von Stylesheets

Stylesheets können an **verschiedenen Positionen** in einem Dokument stehen:
- in einer externen Datei
- im <head>-Bereich der HTML-Datei
- direkt in einem einzelnen HTML-Tag

Ich persönlich nutze für Layouts immer **Variante 1: das CSS wird in eine externe Datei ausgelagert**. Diese besteht aus einer normalen Textdatei mit der Endung .css und darf nur CSS-Anweisungen und keine HTML-Befehle enthalten. Diese Datei wird über folgenden Code, der im <head>-Bereich der HTML-Datei eingefügt wird, eingebunden:

HTML-Code:

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

„style.css“ ist der Dateiname der externen CSS-Datei und kann natürlich benannt werden wie ihr wollt ;-) Dieser Code muss in jeder HTML-Datei stehen, die durch das CSS „gestylt“ werden soll.

Variante 2: das CSS wird in den <head>-Bereich eines Dokuments geschrieben und gilt deshalb auch nur für dieses eine Dokument.
Dafür wird im <head> ein Style-Bereich definiert, in dem dann die CSS-Anweisungen stehen:

HTML-Code:

```
<style type="text/css">
<!--
  hier stehen die CSS-Anweisungen
-->
</style>
```

<!-- Text --> ist übrigens ein HTML-Kommentar, der verhindert, dass ältere Browser, die CSS nicht verstehen, die Anweisungen in der HTML-Seite ausgeben.

Variante 3: mit dem HTML-Attribut **style=“...“** können **HTML-Elemente direkt mit CSS definiert** werden, hier ein Beispiel:

HTML-Code:

```
<p>das ist ein ganz normaler Absatz</p>
<p style="font-size: 10px;">dieser Absatz hat dank CSS eine Schriftgröße von 10 Pixeln</p>
```

Arten von Selektoren

Grundsätzlich gibt es 4 verschiedene Arten von Selektoren, wobei einige erweiterbar sind.

Elementselektoren

Hier stehen euch alle bekannten **HTML-Elemente** zur Verfügung: h1, body, td... Der definierte Style gilt dann genau für dieses Element.
Beispiel:

CSS-Code:

```
p { color: black; }
```

Setzt die Schriftfarbe aller p-Elemente (Absatz) auf schwarz.

CSS-Code:

```
body { font-family: arial; }
```

Definiert alle Schriften im Dokument als Schriftart Arial.

Es ist möglich, **mehrere Elemente** nacheinander zu schreiben (durch ein Komma getrennt), wenn für sie gemeinsame Eigenschaften gelten sollen.
Beispiel:

CSS-Code:

```
h1, h2, h3 { color: red; }
```

Die Überschriften h1-h3 werden nun in roter Schrift angezeigt.

Klassenselektoren

Die Definition eine Klasse beginnt mit einem **Punkt**, dem ein **frei vergebbarer Name** folgt (ohne Leerzeichen, keine Ziffer zu Beginn, Zeichen a-z, Unter- & Bindestriche). Außerdem gibt es auch die Möglichkeit, Klassen nur speziellen HTML-Elementen zuzuordnen.

Dem Element im HTML wird mit dem **Attribut class="*klassenname*"** eine bestimmte Klasse zugewiesen. Hier kommt das HTML-Element häufig zum Einsatz, dem eine Klasse zugewiesen wird.

Beispiele:

CSS-Code:

```
.zitat { font-style: italic; }
```

Jedes beliebige HTML-Element, dem die Klasse „zitat“ zugewiesen wird, erscheint so kursiv, z.B.:

HTML-Code:

```
Das ist ein <span class="zitat">lehrreiches Zitat</span> von Schiller.
```

„lehrreiches Zitat“ wird kursiv angezeigt.

Hat man schon eine allgemeine Stylezuweisung für Überschriften getätigt, braucht aber für eine bestimmte Überschrift eine Extra-Zuweisung, vergibt man dieser einfach eine passende Klasse:

CSS-Code:

```
h1.wichtig {color: red; }
```

HTML-Code:

```
<h1 class="wichtig">sehr wichtige Überschrift</h1>
```

Diese Klasse kann also nur an h1-Elemente vergeben werden.

Ein Element kann auch **mehrere Klassen** bekommen, es nimmt dann die Formatierung von beiden Klassen an:

HTML-Code:

```
<h1 class="zitat wichtig">Wichtige und zitierte Überschrift</h1>
```

ID-Selektoren

Werden am häufigsten zum Layouten mit DIVs benutzt. Ein Div ist ein HTML-Element.
ID-Selektoren beginnen mit einer Raute (#), der ein frei vergebbarer Name folgt. Es ist wichtig, dass dieser Name einzigartig im Dokument ist. Im HTML erhält dann das gewünschte Element das Attribut id="name" (ohne #).

Beispiel:

CSS-Code:

```
#navigation { background-color: red; }
```

HTML-Code:

```
<div id="navigation">dies ist eine Navigation</div>
```

Erzeugt bei dem DIV-Element #navigation einen roten Hintergrund. Hier werden dann beim Layouten Position usw. angegeben.

Pseudoformate

Pseudoformate sind eine Sonderform der Stylesheet-Angabe. Mit ihnen können **Links und ihre Zustände** gestyled werden.

Folgende Möglichkeiten gibt es, wobei die **Reihenfolge** wichtig ist:

CSS-Code:

```
a:link { color: blue; }
a:visited { color: red; }
a:hover { color: green; }
a:active { color: yellow; }
```

a:link ist der Grundzustand eines Links, a:visited ist ein besuchter Link, a:hover ein Link, wenn man mit der Maus drüber fährt, a:active ist ein Link, der gerade angeklickt wird.

Zusätzlich gibt es die **Kontextbezogenen Selektoren**. Sie definieren **einzelne Elemente innerhalb eines Selektors**.

Beispiele:

CSS-Code:

```
h2 a { color: blue; }
```

Definiert alle Links innerhalb eines h2-Elements blau.

CSS-Code:

```
#content a { color: green; }
```

Gibt allen Links im DIV-Container mit der id „content“ eine grüne Schriftfarbe.

CSS-Code:

```
.zitat a { color: grey; }
```

Setzt die Schriftfarbe aller Links, die in einem Element mit der Klasse „zitat“ stehen, auf grau.

Praktischerweise können **IDs und Klassen miteinander kombiniert** werden. Das macht sich besonders bei wechselnden Bildern, die aber an der gleichen Stelle stehen, gut – z.B. Headerbilder, die von Unterseite zu Unterseite wechseln.

Das sieht dann so aus:

CSS-Code:

```
.startseite { color: red; }
.kontakt { color: green; }
```

HTML-Code:

```
<div id="header" class="startseite">der Headertext der Startseite ist rot</div>
<div id="header" class="kontakt">der Headertext der Kontaktseite ist grün</div>
```

14. CSS – Werte & Eigenschaften

Stylesheet-Wertangaben

Bevor wir uns mit den Attributen des CSS befassen, will ich kurz einen Überblick über die Werte, die diese annehmen können, schaffen. Sie unterscheiden sich ein wenig, von den Werten des klassischen HTML – oft sind sie eindeutiger und man viel mehr Auswahlmöglichkeiten.

Grundsätzlich stehen 3 Arten von Werten zur Verfügung: **festgelegte**, **numerische** und **Farbwerte**.

Festgelegte Werte sind bereits vordefinierte Worte, die in Verbindung mit bestimmten Attributen wirken. Viele Attribute kennen mehrere unterschiedlicher Angaben. z.B. kann dem Attribut font-family eine Liste bevorzugter Schriftarten vorgeben werden; das Attribut text-align versteht die festgelegten Werte left, right, center und justify usw.

Numerische Werte erfordern eine Zahl und eine angehängte Maßeinheit (ohne Leerzeichen), z.B. font-size: 10pt; Folgende mögliche **Maßeinheiten** existieren:

px (Pixel) – häufigste und wichtigste Maßeinheit
pt (Punkt) – der DTP-Punkt, wird bei Schriftgrößen verwendet
mm (Millimeter) – von Bildschirmgröße und -auflösung abhängig; gebrochene Werte sind mit Punkt zu trennen; in der Praxis eher ungeeignet
cm (Centimeter) – von Bildschirmgröße und -auflösung abhängig; gebrochene Werte sind mit Punkt zu trennen; in der Praxis eher ungeeignet
in (Inch) – von Bildschirmgröße und -auflösung abhängig; gebrochene Werte sind mit Punkt zu trennen; in der Praxis eher ungeeignet
em (= Breite eines „m“) – entspricht einem Geviert
ex (= Breite eines „x“) – wird ggf. als relative Angabe für Wort-, Zeichen- oder Zeilenabstände benutzt
% (Prozent) – steht allg. als relativ zum umgebenen Element (Browserfenster, Tabellenzelle o.ä.)

Farben kennt ihr bereits aus dem HTML: euch stehen die Farbnamen und Hexadezimalzahlen zur Verfügung.

Stylesheet-Attribute

Nun stelle ich euch die wichtigsten Stylesheet-Attribute vor. Ein gutes Nachschlagewerk für eine komplette Referenz ist die Website http://www.css4you.de - wenn man das Prinzip, nach dem CSS funktioniert verstanden hat, kann man dort alles Nötige zum „CSSen“ nachschlagen.
Nicht jedes Attribut kann bei jeden beliebigen Selektor (Element) benutzt werden. Meistens ist aber offensichtlich, welches Attribut zulässig ist.

Textformatierung (bei fast jedem Element zulässig)

font-family
Hier wird eine Liste von Schriftarten eingetragen, getrennt durch Komma. Festgelegte Werte sind die Schriftarten. Achtet darauf, nur Systemschriften zu verwenden – auf exotische Schriften sollte verzichtet werden (der Besucher hat sie wahrscheinlich nicht auf dem Rechner installiert).
Bsp.:

CSS-Code:

```
font-family: arial, verdana, sans-serif;
```

font-size
font-size ist die Angabe der Schriftgröße. Hier wird ein numerischer Wert eingetragen, meist in Punkt (pt), Pixel (px)
oder em.
Bsp.:

CSS-Code:

```
font-size: 12pt
```

font-style
Wird genutzt, um Schrift kursiv zu setzen. Die festgelegten Werte sind normal, italic (kursiv) und oblique (elektronisch schräggestellt).
Bsp.:

CSS-Code:

```
font-style: italic;
```

font-weight
Dient dazu, eine Schrift fett zu setzen. Festgelegte Werte sind normal und bold (fett).
Bsp.:

CSS-Code:

```
font-weight: bold;
```

text-decoration
Legt fest, ob ein Text mit Linien versehen ist. Festgelegte Werte sind none (keine Linie, Standardeinstellung bei normalem Text), underline (unterstrichen; Standardeinstellung bei Links), overline (Linie über Text), line-through (durchgestrichen).
Bsp.:

CSS-Code:

```
text-decoration: underline overline;
```

letter-spacing
Gibt den Anstand der Buchstaben an (Textlaufweite). 0pt ist der Standardwert.
Bsp.:

CSS-Code:

```
letter-spacing: 3pt;
```

Zusammenfassung font
Die Schriftformatierungen lassen sich in einer Zeile zusammenfassen. So wird aus:

CSS-Code:

```
font-style: italic;
font-weight: bold;
font-size: 12pt;
font-family: arial, verdana, sans-serif;
```

in einer Zeile (**Achtung**, die Reihenfolge style – weight – size – family muss eingehalten werden!):

CSS-Code:

```
font: italic bold 12pt arial, verdana, sans-serif;
```

Bereichs- & Absatzformatierung (alle Absatzbildenden Elemente, z.B. <p>; ebenso Tabellenzellen usw.)

text-align
Ist die Ausrichtung des Textes. Mögliche Werte sind left (linksbündig; Standardwert), center (zentiert), right (rechtsbündig) oder justify (Blocksatz).
Bsp.:

CSS-Code:

```
text-align: right;
```

text-indent
Ermöglicht eine Einrückung der ersten Zeile des Bereiches. Hier werden numerische Werte angegeben.
Bsp.:

CSS-Code:

```
text-indent: 3em;
```

line-height
Legt die Zeilenhöhe fest. Der Standardwert ist etwas höher als die gewählte Schriftgröße. Es stehen numerische Werte zur Verfügung.
Bsp.:

CSS-Code:

```
line-height: 14pt;
```

vertical-align
Ist die vertikale Ausrichtung eines Elements, entspricht dem valign von Tabellenzellen. Mögliche Werte sind baseline (Standardwert), middle, top sowie nummerische Werte.
Bsp.:

CSS-Code:

```
vertical-align: middle;
```

display
Bestimmt, ob ein Element absatzbildend ist oder nicht. Werte sind none (Element wird nicht angezeigt), block (absatzbildend, füllt die Zeile aus, erzeugt Umbruch) und inline (keine neue Zeile, normaler Textfluss).
Bsp:

CSS-Code:

```
display: none;
```

Farben & Bilder (jedes Element mit Textinhalt)

color
Legt die Textfarbe fest.
Bsp:

CSS-Code:

```
color: #660000;
```

background-color
Stellt die Hintergrundfarbe ein. Jedes Element kann seine eigene Hintergrundfarbe haben!
Bsp:

CSS-Code:

```
background-color: #339933;
```

background-image
Definiert das Hintergrundbild eines Elements. Wert ist die URL des Bildes.
Bsp:

CSS-Code:

```
background-image: url(hintergrundbild.jpg);
```

background-attachment
Bestimmt, ob das Hintergrundbild mit der Seite mitscrollt oder fixiert ist. Die festgelegten Werte sind fixed oder scroll (Standardeinstellung).
Bsp:

CSS-Code:

```
background-attachment: fixed;
```

background-repeat
Erlaubt festzulegen, ob und wohin ein Hintergrundbild wiederholt („gekachelt“) wird. Werte sind repeat (in beide Richtungen wiederholen, Standardwert), repeat-y (nur vertikal wiederholen), repeat-x (nur horizontal wiederholen) oder no-repeat (keine Wiederholung – Einzelbild).
Bsp:

CSS-Code:

```
background-repeat: no-repeat;
```

background-position
Ist abhängig vom background-repeat und legt die Position des Hintergrundbildes fest. Der erste Wert bestimmt die horizontale, der zweite die vertikale Position. Wird nur ein Wert angegeben, bestimmt dieser die horizontale Position, vertikal wird automatisch auf 50% gesetzt.
Mögliche Werte: nummerische Angaben sowie left, right, center, top, bottom.
Bsp:

CSS-Code:

```
background-position: 200px 70%;
```

Zusammenfassung background
Man kann die ganzen Angaben zum Hintergrund eines Elementes auch zusammenfassen. So wird aus:

CSS-Code:

```
background-color: #339933;
background-image: url(hintergrundbild.jpg);
background-repeat: no-repeat;
background-attachment: fixed;
background-position: 200px 70%;
```

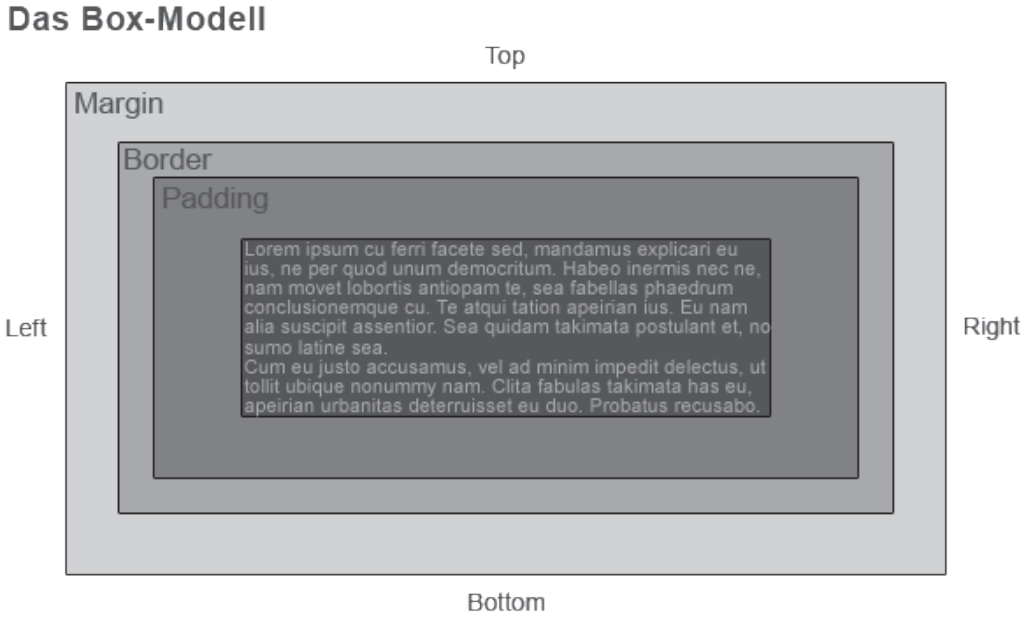
in einer Zeile:

CSS-Code:

```
background: #339933 url(hintergrundbild.jpg) no-repeat fixed 200px 70%;
```

Rahmen & Linien (alle Elemente)

Box-Modell
Das Boxmodell beschreibt den rechteckigen Bereich, der für jedes Element reserviert ist. Dieser Bereich setzt sich aus Inhalt, Innenabstand (Padding), Rahmen (Border) und Außenabstand (Margin) zusammen.



padding
Padding bestimmt den Abstand eines Elementes zum Rand – der Innenabstand. Ein einfaches „padding“ ermöglicht einen einheitlichen Abstand zu allen 4 Seiten. padding-top bestimmt den Abstand nach oben, padding-right nach rechts, padding-bottom nach unten und padding-left den Abstand nach links.
Die 4 Attribute können wie folgt zusammengefasst werden: wird bei „padding“ nur ein Wert angegeben, gilt dieser für alle 4 Seiten (siehe oben), 2 Werte bestimmen den Abstand oben/unten (1. Wert) und links/rechts (2. Wert). 3 Werte bestimmen den Abstand oben (1. Wert), links/rechts (2. Wert) und unten (3. Wert). 4 Werte bestimmen den Abstand oben (1. Wert), rechts (2. Wert), unten (3. Wert) und links (4. Wert).
Bsp:

CSS-Code:

```
padding-top: 10px;
```

Bsp. Zusammenfassung:

CSS-Code:

```
padding: 20px 10px 5px;
```

margin
Margin definiert den Abstand vom Rand eines Elementes zum nächsten Element (z.B. Browserfenster). Es gelten die gleichen Regeln und Möglichkeiten wie bei „padding“.
Bsp:

CSS-Code:

```
margin-top: 20px;
```

Bsp. Zusammenfassung:

CSS-Code:

```
margin: 10px 15px;
```

border
Border ist der Rahmen, der um ein Element und dessen Innenabstand liegt. Dem Rahmen kann eine Breite (width, nummerische Werte) und eine Art (style) zugewiesen werden, Werte für den Style sind: solid (durchgezogen), dashed (gestrichelt), dotted (gepunktet) und double (doppelt). Natürlich kann auch eine Farbe bestimmt werden.
Diese Dinge können allgemein allen 4 Seiten zugewiesen werden (border), oder nur speziellen Seiten (border-top, border-right, border-bottom,
boder-left).
Bsp:

CSS-Code:

```
border-color: red;
border-top-width: 3px;
border-left-style: dashed;
border-bottom-color: #003300;
```

Zusammenfassung border
Man kann einige ganzen Angaben des border zusammenfassen. So wird aus:

CSS-Code:

```
border-width: 5px;
border-style: solid;
border-color: red;
```

in einer Zeile:

CSS-Code:

```
border: 5px solid red;
```

15. CSS – Layer erzeugen & positionieren

Mit sogenannten Layern (engl. für Ebene/Schicht) werden die meisten CSS-Layouts erstellt.
Man verwendet dafür das Blockelement DIV (Abkürzung von „division“, engl. für Bereich), welches ich euch bereits in Kapitel 12 vorgestellt habe.

DIVs sind eigenständige Ebenen, die entweder frei über der Webseite „schweben“ oder sich darin anpassen. Sie können beliebig positioniert werden, was ein pixelgenaues Arbeiten möglich macht.

Dank JavaScript ist es möglich, die Layer nachträglich zu verändern, z.B. ein-/ausblenden, positionieren usw. Mal schaun, vielleicht erstelle ich dazu ein paar Mini-Tutorials :)

Der **Aufbau eines DIVs** kennt ihr schon aus Kapitel 13:

CSS-Code:

```
#name { eigenschaft: wert; }
```

HTML-Code:

```
<div id="name">dies ist ein Div</div>
```

Um aus einem normalen DIV ein Layer zu machen, ist folgende Eigenschaft von Nöten: **position**. Position bestimmt die **Art der Positionierung**. Die beiden meistverwendeten Werte sind **absolute** (für eine pixelgenaue Positionierung, unabhängig von allen anderen Elementen der Seite) und **relative** (für eine relative Positionierung, abhängig von den benachbarten Elementen). Ein weiterer Wert ist **fixed** (fixiert das DIV – es scrollt nicht mit), dieser wird aber vom Internet Explorer erst ab der Version 7 unterstützt und wird deshalb nicht empfohlen (immerhin hat ein großer Teil der Surfer noch den IE6).

Nachdem die Art der Positionierung eines Layers festgelegt wurde, muss die **Position** selber bestimmt werden. Dafür gibt es die Eigenschaften **left**, **top**, **right** und **bottom**. Es werden maximal 2 dieser Angaben gemacht – der Abstand links oder rechts sowie der Abstand oben oder unten. Meist wird top und left benutzt.

Hier ein **Beispiel für einen absolut positionierten Layer**:

CSS-Code:

```
#navi {
position: absolute;
top: 100px;
left: 300px;
}
```

HTML-Code:

```
<div id="navi">dies ist ein Div mit Navigation</div>
```

und hier eines für einen **relativ positionierten Layer**:

CSS-Code:

```
#navi {
position: relative;
top: 15%;
left: 10%;
}
```

HTML-Code:

```
<div id="navi">dies ist ein Div mit Navigation</div>
```

Als zusätzliche Eigenschaft gibt es die **Layerreihenfolge: z-index**. Mit dem z-index können Layer, die sich überlagern, angeordnet werden, ähnlich wie die Ebenen im Photoshop. Je höher der Wert (Ziffern von 0 bis 99, negative Zahlen sind auch möglich), umso weiter oben liegt der Layer.

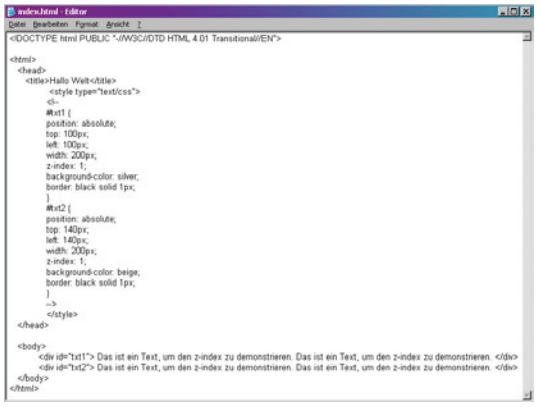
Ich demonstriere das mal an einem Beispiel:

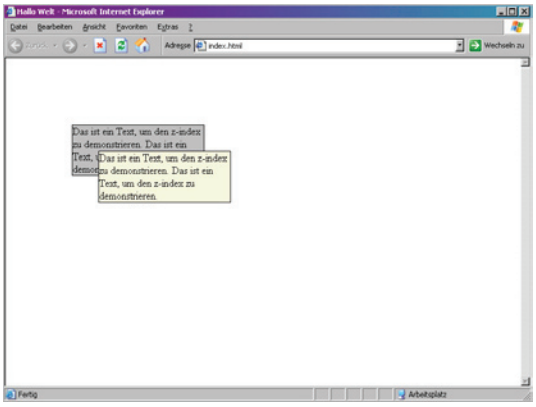
CSS-Code:

```
#txt1 {
position: absolute;
top: 100px;
left: 100px;
width: 200px;
z-index: 1;
background-color: silver;
border: black solid 1px;
}

#txt2 {
position: absolute;
top: 140px;
left: 140px;
width: 200px;
z-index: 1;
background-color: beige;
border: black solid 1px;
}
```

HTML-Code:

```
<div id="txt1"> Das ist ein Text, um den z-index zu demonstrieren. </div>
<div id="txt2"> Das ist ein Text, um den z-index zu demonstrieren. </div>
```





Ein weiteres interessantes Attribut ist **overflow** – damit macht man eine **DIV scrollbar** und erzeugt einen iFrame-Effekt. Werte sind auto (scrollt bei bestimmter Textlänge), scroll (scrollt immer) und hidden (scrollt nicht und schneiden überschüssigen Text ab.

Desweiteren macht es Sinn, die **Größe des DIVs** mit den bekannten Attributen width und height anzugeben. Gerade bei Texten ist das width wichtig, da der Text sonst einfach über die gesamte Seite verläuft.