4 Formatierung mit CSS

Worum es geht

HTML ist nicht die einzige Technologie, die man zum Erstellen einer Homepage benötigt, bloss die Elementarste. In den vergangenen Jahren haben sich parallel zum Wachstum des Internets die darin verwendeten Technologien vervielfacht. Einen besonderen Stellenwert hat dabei CSS (Cascading Style Sheets) bekommen. Mit CSS wird es möglich, HTML-Tags weitreichend zu formatieren und eine Webseite grafisch zu gestalten.
Du wirst in diesem Kapitel beispielhaft die Funktionsweise folgender Konzepte kennenlernen:

- Die Integration von CSS mit HTML anhand der CSS-Selektoren
- Der Einsatz verschiedener CSS-Eigenschaften zur Gestaltung
- Der Gebrauch einer externen "zentralen" CSS-Datei
- Die Anwendung von CSS auf Tags, Klassen und Einzel

Material und Referenzen

CSS ist an verschiedenen Orten ausführlich beschrieben:
https://developer.mozilla.org/en-US/docs/Web/CSS/Reference
http://www.w3schools.com (englisch).
http://www.css4you.de
http://wiki.selfhtml.org/wiki/CSS

4.1 CSS: Cascading Style Sheets

Stylesheets sind eine Ergänzung zu HTML. Es handelt sich dabei um eine von den meisten Browser verstandene Sprache zur Definition von Formateigenschaften (Farben, Schriften, Ränder, Abstände etc.) von HTML-Elementen. CSS wirkt sich also auf die Darstellung von Webseiten aus. Mit Hilfe von Stylesheets kannst du beispielsweise bestimmen, dass Links in der Schriftart Helvetica, aber weder fett noch unterstrichen erscheinen und mit der Farbe grün versehen werden. Angaben dieser Art sind mit reinem HTML nicht möglich. Ausserdem lassen sich per CSS auch Inhalte bzw. HTML-Elemente positionieren. Da man in der Regel für eine Webseite in sämtlichen HTML-Dokumenten dieselben Formateigenschaften anwenden möchte, gibt es zudem die Möglichkeit, die CSS-Angaben zentral in einem eigenen Dokument festzuhalten und alle HTML-Dokumente darauf zugreifen zu lassen. Will man seine Website möglichst korrekt programmieren, versucht man die semantische Strukturierung mit HTML und die grafische Formatierung mit CSS umzusetzen. Dieses Kapitel kann nur eine knappe Einführung in die Funktionsweise von CSS liefern, um die vielfältigen Möglichkeiten dieser Technologie ausreizen zu können, braucht es jedoch eine darüber hinaus gehende Vertiefung.

4.2. CSS-Regeln definieren

Um einen HTML-Tag mittels CSS genauer zu definieren, deklariert man eine sogenannte Regel, die der folgenden Syntax entspricht:

```
Selektor { Eigenschaft:Wert; }
```

An der Stelle des Selektors muss ein HTML-Elementtyp eingesetzt werden, den man neu formatieren möchte. So kannst du beispielsweise die Formatierung des Tags für einen Textabsatz, `<P>` ändern, indem du eine CSS-Regel wie folgt benutzt:

```
P { color:#0000ff; font-family:Verdana;}
```

Damit bestimmst du, dass ein Textabsatz blau und in der Schriftart Verdana ausgegeben wird. Wie du an dem Beispiel erkennen kannst, lassen sich mehrere Eigenschaften innerhalb derselben Regel definieren, indem sie durch ein Semikolon voneinander getrennt werden. Die verschiedenen möglichen Regeln findest du in SelfHTML aufgeführt.

4.3 Die Integration von CSS in HTML

Nun stellt sich die Frage, wie diese CSS-Defintionen in das HTML eingebunden werden, damit der Browser es korrekt anwendet. CSS-Angaben werden jeweils auf eine von drei Arten eingesetzt, wobei die Differenz v.a. darin besteht, wie «nah» am HTML bzw. wie «lokal» sie sind:

4.3.1 Globale CSS-Definitionen in einer externen CSS-Datei

Hast du eine Website, die aus mehreren HTML-Dokumenten besteht, deren Tags alle in der gleichen Weise mit CSS formatiert werden sollen, empfehlen wir dir, die CSS-Angaben in eine separate Datei zu schreiben und alle beteiligten HTML-Dokumente anzuweisen, bei dieser einen externen CSS-Datei («stildatei.css») die Stilangaben zu holen. Dazu fügst du in die betroffenen HTML-Dokumente lediglich die folgende Zeile in den `<HEAD>`-Bereich ein:

```
<LINK HREF="stildatei.css" REL="stylesheet"
TYPE="text/css" media="all">
```

Diese globale Vorgehensweise garantiert, die Darstellung durch alle Dokumente einer Website konsistent zu halten – zudem ist es dadurch ein Einfaches, durch eine Änderung der CSS-Datei auf einen Schlag das Aussehen einer ganzen Website zu ändern, ohne an einem einzigen HTML-Dokument eine Änderung vornehmen zu müssen. Ein Beispiel dafür findest du in unseren Demopages, wo zwei Webpages auf dieselbe CSS-Datei zugreifen.

4.3.2 Zentrale CSS-Definition innerhalb einer HTML-Datei

Wenn du CSS-Angaben für ein ganzes Dokument definieren möchtest, empfehlen wir dir, die Definitionen im `<HEAD>` des HTML-Dokumentes in einem eigenen Bereich unterzubringen, der mit dem `<STYLE>`-Tag markiert wird. Die Angaben gelten dann für alle einzelnen Vorkommnisse des entsprechenden Elements («Tags») innerhalb dieses Dokumentes, nicht aber für andere HTML-Dokumente.

```
<STYLE  TYPE="text/css">
      P {font-size:16px;}
</STYLE>
```

4.3.3 Direkt zum HTML-Element hinzuschreiben

Man kann eine Stilangabe mittels des `STYLE`-Attributs direkt zu einem HTML-Element hinzuschreiben, beispielsweise so:

```
<P STYLE="font-size:16px;">Text</P>
```

Die Besonderheit dieser Lösung ist, dass die Angabe nur genau für dieses einzelne Vorkommnis eines Tag-Elements gilt.

Bei der Auswertung der CSS-Angaben geht der Browser so vor, dass er lokale Angaben stärker gewichtet als globale.

4.4 CSS-Quellcode im BlueGriffon bearbeiten

CSS wird in der Regel in einem eigenen Dokument festgehalten, das mit dem HTML-Dokument verknüpft wird (s. Kapitel 4.3.3). Wie man ein solches CSS-Dokument erstellt, steht in Kapitel 5.4. An dieser Stelle geht es nur darum, wie man ein bestehendes und schon mit meinem HTML-Dokument verknüpftes CSS-Dokument im BlueGriffon öffnen und als Klartext bearbeiten kann. Um im BlueGriffon an den Quellcode eines CSS-Dokuments zu gelangen, musst du das dazugehörige HTML-Dokument in der WYSIWYG-Ansicht geöffnet haben. Dann wählst du im Hauptmenü unter «Konsole» die Option «Stile» an. Es öffnet sich ein kleines Dialogfenster, das sämtliche mit der HTML-Datei verknüpfte CSS-Dateien auflistet (Abbildung 4.1). Mit einem Mausklick auf das gewünschte CSS-Dokument öffnest du den entsprechenden CSS-Quellcode, den du nun bearbeiten kannst. Um die Änderungen abzuspeichern, musst du auf den Button «Änderungen speichern» klicken (Abbildung 4.2).

4.5 Spezifität: HTML-Tag, Klasse, Id

4.5.1 Elementweite CSS-Formatierung

CSS erlaubt es uns, die Darstellung sämtliche Vorkommnisse eines HTML-Elements (neu) zu formatieren. In diesem Fall setze ich an die Stelle des Selektors einfach den HTML-Tag-Namen (wie oben in Kapitel 4.2 bzw. Abbildung 4.2).

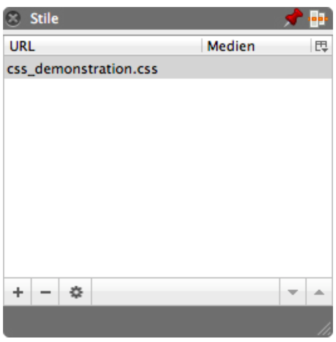

Abbildung 4.1: Die CSS-Stilkonsole

Abbildung 4.2.: Der Quellcode des CSS-Dokumentes mit Speicher-Button.

4.5.2 Klassenweite CSS-Formatierung

Es kann jedoch auch vorkommen, dass ich nicht sämtliche Vorkommnisse eines HTML-Elements spezifisch gestalten will, sondern nur eine Untergruppe davon, beispielsweise wenn ich für bestimmte Textabschnitte eine eigene Farbgebung möchte, die sie von den übrigen im Dokument abhebt. Dafür gibt es die Möglichkeit sogenannte Klassen zu bilden (z.B. eine Klasse für diese besonderen Abschnitte).
Wenn ich ein spezifisches Vorkommnis eines HTML-Elements einer Klasse zuordnen möchte, muss ich ihm das CLASS-Attribut hinzufügen. Diesem kann ich als Wert einen von mir gewählten Klassennamen zuweisen, der die Funktion der Klasse möglichst präzise bezeichnet. Das sieht dann z.B. so aus:

```
<P CLASS="notabene">Beachte: </P>
```

In den CSS-Angaben kann ich nun meine Definitionen dieser Klasse direkt zuweisen, indem ich den Namen der Klasse als CSS-Selektor verwende. Somit ist die Klassendefinition apriori keinem speziellen HTML-Tag zugewiesen. Um eine Klasse als solche kenntlich zu machen, muss ich vor den Klassennamen einen Punkt «`.`» setzen. Also gemäss obigem Beispiel so:

```
p.notabene { color:red;}
```

Nun weiss der Browser, dass alle Links mit dem `<CLASS>`-Attribut und dem Wert «`notabene`» rot und ohne Unterstrich dargestellt werden müssen.

4.5.3 CSS-Formatierung eines einzelnen Elements

Es kann auch vorkommen, dass ich nur ein einziges Vorkommnis eines HTML-Elements formatieren möchte. Da kann ich entweder auf das `<STYLE>`-Attribut zurückgreifen (s.o. 4.3.1) oder - und das empfehlen wir aus Gründen der Code-Stilistik - ich verknüpfe dieses Tag-Vorkommnis mit den CSS-Definitionen, indem ich ihm anhand des `ID`-Attributs eine eindeutige Identität gebe. Das sieht dann beispielsweise so aus:

```
<P ID="hauptartikel">
      ...
</P>
```

Innerhalb der CSS-Angaben kann ich eine individuelle Definition für genau dieses einzelne Element vornehmen, indem ich den ID-Namen als Selektor verwende und zu dessen Kennzeichnung als individueller ID ein Rautezeichen («#») voransetze:

```
#hauptartikel {
      background-color:#eaeae;}
```

4.6 Kombination und Vererbung der CSS-Definitionen

Selektoren lassen sich auch kombinieren, um ganz spezifische Untergruppen von HTML-Vorkommnissen anzusprechen. So bedeutet der folgende Selektor

```
.sidebar h1 {font-size:18px;}
```

dass die Schriftgrösse von 18 Pixel nur auf jene Titel erster Ordnung (`<H1>`) angewendet werden soll, die sich innerhalb eines Elements befinden, das der Klasse «`sidebar`» zugeordnet wurde.
Es gibt eine Reihe von unterschiedlichen Kombinationsmöglichkeiten. Mehr dazu findest du hier:
http://www.css4you.de/wscss/css04.html

Der Aufbau eines HTML-Dokumentes besteht aus einer hierarchischen Verschachtelung von verschiedenen HTML-Elementen, angefangen beim `<HTML>`-Element. Nun ist es so, dass einige CSS-Angaben sich von einem übergeordneten Elternelement auf die untergeordneten Kindelemente weitervererben lassen. Dies gilt insbesondere für die Textgrösse und -art. Daher genügt es, wenn ich meine Standardtextdarstellung für den `<BODY>`-Tag definiere:

```
body {
       color:#fee7e7;
       font-family:Times, serif;
       color:#2c2c2c;}
```

Die obige Angabe wird nun nicht nur auf den Body, sondern auch auf alle Kindelemente (z.B. `<P>`, `<H1>`) angewendet, sofern sie keine eigenen Definitionen erhalten haben. Es sind jedoch nicht alle CSS-Eigenschaften vererbbar, daher muss man sich immer bewusst sein, ob die eingesetzte Eigenschaft vererbt wird und ob man das überhaupt möchte. Einen Überblick über die Vererbbarkeit der wichtigsten CSS-Eigenschaften findest du hier:
http://www.css4you.de/shortref.html
Wenn du eine nicht vererbbare Eigenschaft von einem Elternerlement an ein Kindelement vererben möchtest, kannst du dem Browser dies gezielt mitteilen:

```
P { margin:inherit;}
```

Nun übernimmt er für den Textabschnitt den Aussenabstand des übergeordneten HTML-Tags.

4.7 Textformatierung

Text wird meistens innerhalb eines <P>-Elements geschrieben. Dieser Tag verfügt über einige Attritubte, mit denen man die Darstellung beeinflussen kann, z.B. COLOR, FACE etc. Diese Attribute sind jedoch sehr limitiert in ihren Gestaltungsmöglichkeiten und gelten auch nach der aktuellen Auffassung als veraltet, da sie die Formatierung nicht sauber von der Strukturierung unterscheiden. Die korrekte Formatierung von Text nimmt man daher mittels CSS vor. Dafür gibt es eine ganze Reihe von definierbaren Eigenschaften. Einige seien hier genannt:

`font-family` dient der Bestimmung der Schriftart. Als Wert kann die gewünschte Schriftart oder eine Liste von Schriftarten angegeben werden, z.B.

```
h1 { font-family:Courier,Baskerville; }
p { font-family:Helvetica,"Courier New";}
```

Der Browser versucht der Reihenfolge nach, die gewünschte Schrift auf dem System zu finden und sie zu verwenden. Beachte, dass Schriften, die ein Leerzeichen im Namen besitzen, in Gänsefüsschen gesetzt werden müssen (siehe ”Courier New”).

Achtung: Da nicht alle Rechner die gewünschten Schriften installiert haben (und der Browser dann womöglich auf eine unschöne Standartschrift zurückgreifen muss, um den Text darzustellen), gibt es die Möglichkeit auf dem Server eine Schrift bereit zu legen, die der Browser dann herunter lädt und verwendet. Dazu musst du die `@font-face`-Regel einsetzen.

`font-size` dient der Bestimmung der Schriftgrösse. Sie kann entweder in relativen Masseinheiten wie rem, em, % oder in absoluten Einheiten wie Pixeln angegeben werden. Relative Angaben beziehen sich immer auf die Grössenangaben eines übergeordneten Elternelements. Die Einheit rem bezieht sich beispielsweise immer auf die im Browser eingestellt Standardgrösse; em und % sind relativ zur Schriftgrösse des direkten Elternelements bezogen. Beispiele für Grössenangaben sind:

```
p { font-size: 1.5rem; }
p { font-size: 16px; }
p { font-size: 90%; }
```

Die Eigenschaft `line-height` dient der Bestimmung der Zeilenhöhe und kann wiederum sowohl absolut (Pixel, Standardzeilenhöhen) oder relativ (rem, em, %) angegeben werden.

```
p { line-height:115%; }
```

Weitere wichtige typografische CSS-Eigenschaften sind:

`text-decoration`: Mit text-decoration kann man Texte unter- oder überstreichen, aber auch durchstreichen oder blinken lassen. Ebenso kann man die standartmässig vorgenommene Unterstreichung von Hyperlinks aufheben durch text-decoration:none. Die entsprechenden Werte für die Eigenschaft `text-decoration` heissen `underline`, `overline`, `line-through`, `blink` und `none`.

`font-weight`: Gibt die «Dicke», das Gewicht der Schrift an. Kann Werte der folgenden Liste erhalten: `100`, `200`, `300`, `400` (`normal`), `500`, `600`, `700` (`bold`), `800`, `900`. Beachte, dass du nur Schnitte von Schriften einsetzen solltest, die extra erstellt wurden (also zum Beispiel eine Baskervill Bold, nicht aber einen Schnitt, den der Browser aus dem normal-Schnitt auf bold hochrechnet (und dabei unschöne Dinge tut) (siehe zur Erklärung auch http://praegnanz.de/weblog/webfonts-fette-fehlerquelle).

Beispiele:
```
h1 {font-weight: 700;}
span.extra {font-weight: bold;}
```

A propos: Bei der Benutzung der @font-face Regel zur Einbindung von Webfonts ist es wichtig, die richtigen Schnitte den richtigen font-weights zuzuordnen (also normal/regular zu 400, bold zu 700), um dann die Eigeschaft `font-weight` auf die eingebundene Schrift einsetzen zu können.

`font-style`: Ermöglicht es, eine Schrift normal, kursiv oder mittels einer Schrägschrift darzustellen.
```
h3 {font-style: normal;}   /* Standart */
p {font-style: italic;}  /*Kursivschrift*/
h4 {font-style: oblique;} /*Schrägschrift*/
```

`letter-spacing`: Hiermit kann vom normalen Buchstabenabstand abweichender, zusätzlicher Raum zwischen den Buchstaben angegeben werden. Möglichkeiten sind
```
p { letter-spacing: 1em; }
h2 { letter-spacing: -0.05rem; }
h5 { letter-spacing: 6px; }
h6 { letter-spacing: normal;} /* Standart */
```

`word-spacing`: Definiert den Abstand zwischen den Wörtern - ähnlich wie letter-spacing und kann in den Einheiten rem, em, px, nicht aber in % angegeben werden. Ein Beispiel:
```
p { word-spacing: 2em;}
```

`text-indent`: Gibt die Einrückung der ersten Zeile eines Blocks an. Kann in % (relativ zur Breite des Blocks) oder in Einheiten (wie em, rem, px oder sogar mm) angegeben werden.

Beispiele:
```
p { text-indent: 30%;}
p { text-indent: 4em;}
```

`color:` Diese Eigenschaft setzt die Schriftfarbe (und müsste deshalb logischerweise text-color: heissen, diese Eigenschaft wurde an sich falsch benannt). Dabei können verschiedene Farbnotationen verwendet werden:

`p {color:#rrggbb;}` rr, gg und bb sind hexadezimale Helligkeitswerte für die Subpixel rot, grün und blau, z.B.
`p {color:#dd223e;}`.

`p {color:rgb(rot, grün, blau);}` definiert die Farbe anhand RGB-Werten in dezimaler Notation, z.B.
`p {color:rgb(120, 44, 190);}`

`p {color: rgba(rot, grün, blau, opazi);}` beinhaltet neben den RGB-Werten in dezimaler Notation auch noch einen Alpha-Wert, der die «Deckkraft» oder «Opazität» festlegt. Dabei ist ein a-Wert von 0 (nicht deckend) bis 1 (voll deckend) möglich, z.B.
`p {color: rgba(34, 12, 64, 0.3);}`.

`p {color: pink;}` lässt einem eine der benannten Farben verwenden (http://wiki.selfhtml.org/wiki/Grafik/Farbpaletten)

`p {color: currentcolor;}` Ändert nichts an der Farbe.

`background-color:` ermöglicht das Setzen der Hintergrundfarbe eines Elements. Die Farbdefinitionen entsprechen denen von `color`.

Ausserdem als erwähnenswert erachten wir das Pseudoelement `::first-letter`. Damit kann das erste Zeichen Inhalt eines Elements separat ausgezeichnet werden:

```
header::first-letter{font-size:200%;}
h3::first-letter{color:lightskyblue;}
```

Beachte, dass man in der Regel die Zeilenhöhe (`line-height`) des `::first-letter` gleich hoch wie der Rest des Textes des Abschnitts setzen sollte, da ansonsten das Schriftbild leidet.

Eine komplette Liste der CSS-Eigenschaften findest du online, zum Beispiel unter
https://developer.mozilla.org/en-US/docs/Web/CSS/Reference.

4.8 Das Box Modell, Block- vs Inline-Elemente

Wenn ein Browser einen Inhalt darstellt, dann sieht er dafür immer eine rechteckige Fläche als Inhaltscontainer vor. Dieses Rechteck wird als Box bezeichnet. Die Platzierung der Inhalte mittels CSS muss sich somit an diesem Box Modell orientieren. Dabei gilt es zu berücksichtigen, dass man sämtliche HTML-Elemente für Inhalte in zwei Gruppen unterscheiden kann: Block-Elemente und Inline-Elemente.

Der Unterschied besteht darin, dass Block-Elemente per Voreinstellung auf einer neuen Zeile beginnen und die gesamte Breite ihres Elternelementes ausfüllen (CSS-Eigenschaft `display:block;`). Sie erzwingen somit einen Zeilenumbruch. Typische Block-Elemente sind `<P>`, `<H1>`, `<LI>`, `<DIV>`.

Inline-Elemente hingegen nehmen nur die Breite ein, die ihr Inhalt tatsächlich beansprucht (z.B. `<A>`, `<IMG>`, `<SPAN>` etc). Sie sind per Voreinstellung mit der CSS-Eigenschaft `display:inline;` ausgezeichnet.

Inline-Elemente befinden sich immer innerhalb von Block-Elementen.

Bei Block-Elementen kann man sowohl die Höhe und Breite per CSS definieren (Eigenschaften «padding» (Innenabstand), «margin» (Aussenabstand)), während ich bei Inline-Elementen nur die Abstände in horizontaler Richtung formatieren kann.

Nota Bene :: :: :: :: :: :: :: :

Bitte beachte, dass die Positionierung mit CSS eine ziemlich komplexe Angelegenheit ist, die selbst versierte WebdesignerInnen regelmässig ins Schwitzen bringt. Es gibt eine Vielzahl von Ausnahmen, die teilweise nicht hinreichend dokumentiert sind. Daher können wir die Thematik hier auch nur oberflächlich darstellen.
Für weitere Informationen: http://w3schools.com/css/css_positioning.asp

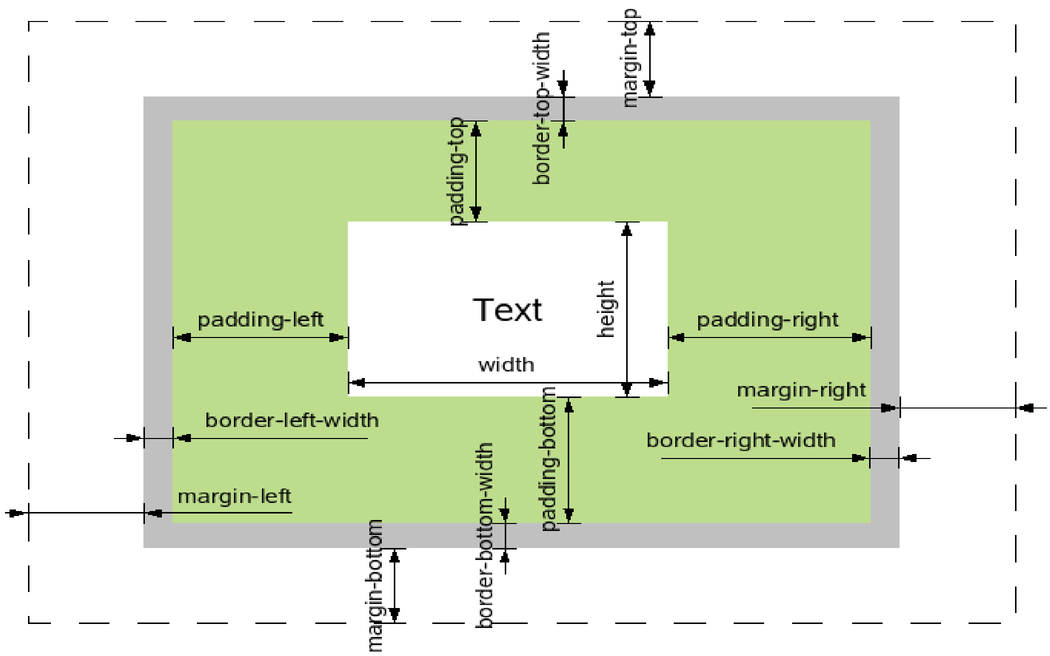

Abbildung 4.3: Aufbau des Boxmodells.

Anhand der CSS-Zuweisung «display: inline-block» kann ich ein eigentliches Block-Element gegen innen als Block-Element und gegen aussen als Inline-Element definieren. Dadurch kann ich Block-Elemente wie <P> gleichzeitig mit einer spezifischen Breite versehen und sie im Dokumentfluss jedoch nebeneinander statt untereinander anordnen.

Mit CSS kannst du Block-Elemente individuell grafisch gestalten und positionieren. Dabei musst du verstehen, wie eine Box aufgebaut ist. Die Abbildung 4.3 illustriert diesen Aufbau. Die Breite bzw. Höhe einer Box setzt sich zusammen aus der Breite bzw. Höhe des Inhalts plus jener des Innenabstands (padding) (mal 2) plus jener des Rahmens (border) (mal 2) plus jener des Aussenabstands (margin) (mal 2). Dabei musst du jedoch berücksichtigen, dass man dem Aussenabstand keine Hintergrundfarbe geben kann. Ausserdem gilt es zu beachten, dass die Aussenabstände von zwei vertikal übereinander liegenden Boxen sich überschneiden (Stichwort "collapsing margin"). Das heisst, wenn ich zwei Boxen mit Aussenabständen von 20 Pixeln respektive 30 Pixeln habe, wird der Aussenabstand zwischen ihnen 30 Pixel betragen. Der Aussenabstand kann übrigens auch einen negativen Wert wie -10 Pixel einnehmen, wodurch das betreffende Element sein vorangehendes um 10 Pixel überschneidet und von seinem nachfolgenden überdeckt wird.

Von den überschneidenden Aussenabständen ausgenommen sind u.a. "gefloatete" Elemente (Kapitel 4.9.2), absolut positionierte Elemente (Kapitel 4.9.1) und Inline-Blockelemente.

4.9 Positionierung und Layout mit CSS

Mit CSS lassen sich die Inhalte auch innerhalb des Browserfensters wunschgemäss positionieren. Das kann man erreichen, indem man ein bestimmtes Block-Element direkt definiert, oder man verwendet das unsichtbare Blockelement <DIV> und setzt dieses als Container-Box für mehrere darzustellende Inhalte (Texte, Links, Grafiken etc.) ein, in den ich diese dann einbette. Block-Elemente lassen sich mit der CSS-Eigenschaft `position` sowohl absolut wie auch relativ positionieren (siehe dazu unten Kapitel 4.9.1). Alternativ kann man Block-Elemente mit der Eigenschaft `float` im Dokumentfluss aneinander reihen (siehe unten Kapitel 4.9.2). Aufbauend auf diesen beiden Techniken lassen sich diverse Layoutkonzepte umsetzen; und unter Einbezug von Javascript lassen sich Boxen auch bewegen und unsichtbar machen.
Um auf ein Block-Element per CSS zuzugreifen, gibst du ihm entweder mit dem ID-Attribut eine eindeutige oder mit dem CLASS-Attribut eine gruppenspezifische Kennzeichnung.

4.9.1 Die Eigenschaft «position»

Wenn der Browser eine Seite darstellt, hält er sich in der Reihenfolge der Boxen an den Dokumentenfluss im Quellcode, in dem ein Element nach dem anderen folgt. Mit der Eigenschaft `position:relative;` änderst du zunächst einmal noch nichts an dieser Aneinanderreihung. Aber wenn du die Abstands-Eigenschaften `left` oder `right`, `top` oder `bottom` hinzu

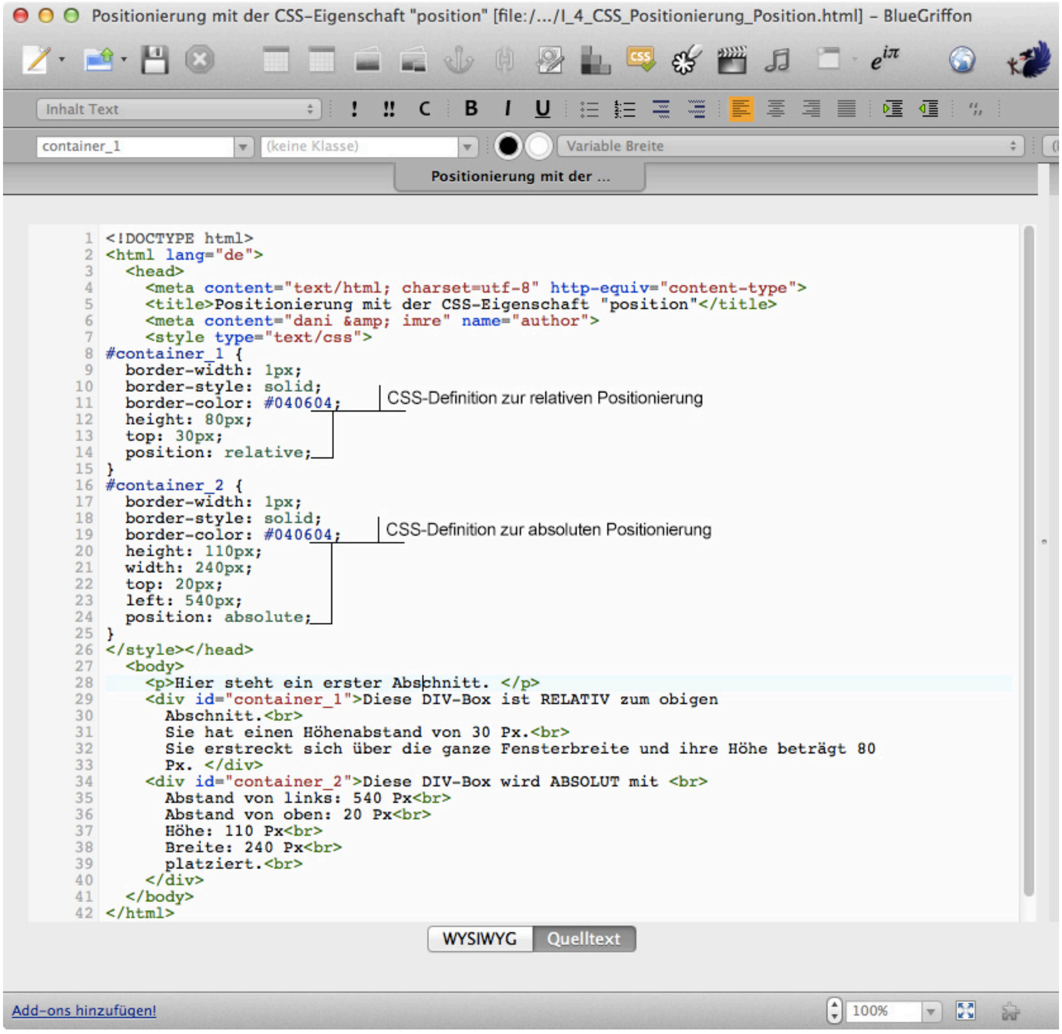

Abbildung 4.4: Der Quellcode einer Seite, die einen DIV-Container relativ und einen absolut positioniert.

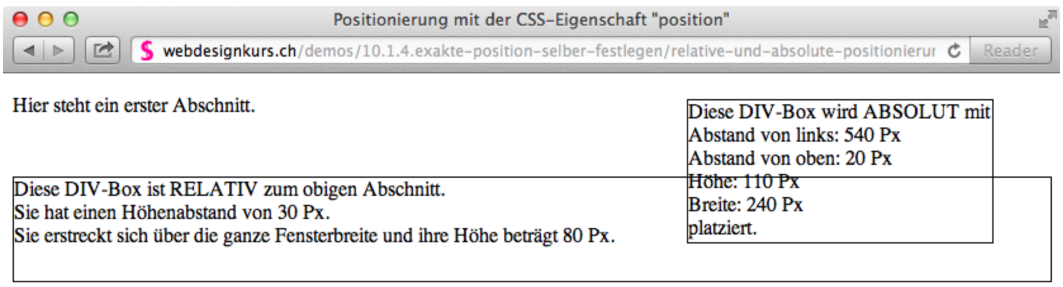

Abbildung 4.5: Die entsprechende Darstellung im Browser. Während die relativ positionierte Box sich innerhalb des Dokumentflusses auf den vorangegangenen Abschnitt bezieht, wird die absolut positionierte Box unabhängig vom Dokumentfluss platziert.

nimmst, kannst du die Position des fraglichen Elements in Relation zu seiner ursprünglichen Position im Dokumentfluss positionieren (Im Beispiel in Abbildung 4.5 wird ein `DIV`-Container (container_1) in der Höhe relativ zum vorangehenden Element, einem Abschnitt, positioniert). Der eigentliche Dokumentfluss wird dadurch jedoch nicht verändert, die nachfolgenden Elemente werden an die «eigentliche» Position des Elements angefügt.

Wenn man ein Element mit `position:absolute;` und den dazugehörigen Abstands-Eigenschaften innerhalb des Browserfensters platziert, wird es aus dem Dokumentenfluss heraus gelöst. Daher wird es mit `top` oder `left` in Abhängigkeit zum nächsten Elternelement, das selber mit `relative` oder `absolute` positioniert wird, positioniert, oder sonst in Abhängigkeit zum `BODY` . Das kann dazu führen, dass ein solches Element andere Elemente innerhalb des Dokumentflusses grafisch überlagert (vgl. dazu den zweiten `DIV`-Container (container_2) in Abbildung 4.5). Ausserdem musst du wissen, dass das Element dadurch zu einem Inline-Block-Element wird, dass nicht mehr automatisch die gesamte Breite des Dokuments einnimmt.

Beachte: Du musst zwischen absoluter und relativer Positionierung einerseits und zwischen absoluten und relativen Wertangaben andererseits unterscheiden. Darum kann es durchaus eine absolute Positionierung anhand von relativen Werten wie auch eine relative Positionierung anhand von absoluten Werten oder umgekehrt geben. Also z.B.:

```
DIV {
      position: absolute;
      left: 10%;}
```

oder

```
DIV {
      position: relative;
      left: 10px;}
```

Mit `position:fixed;` lässt sich eine Box innerhalb des Browserfensters fixieren, selbst wenn die BesucherIn den Rest der Seite scrollt. Eine fixierte Box befindet sich wie eine absolut positionierte ausserhalb des Dokumentflusses. Ihre Referenz ist jedoch der «Viewport», also der im Browserfenster gerade sichtbare Bereich. Beachte, dass du beim Gebrauch dieses Werts die unterschiedlichen Bildschirmgrössen deiner SeitenbesucherInnen berücksichtigen solltest.

4.9.2 Die Eigenschaft «float» und clearing

Mit der CSS-Eigenschaft `float` kann man ein Element rechts- oder linksbündig innerhalb seines Containers bzw. Elternelements platzieren, so dass der nachfolgende Inhalt dieses Containers das Element «umfliesst», d.h. nicht vertikal darunter, sondern horizontal auf der selben Höhe, also «daneben» dargestellt wird (s. Abbildung 4.7). Die Eigenschaft `float` alleine macht das Element zunächst zu einem Inline-Blockelement, dem man eine Breite zuweisen muss, wenn es breiter als sein Inhalt werden soll.

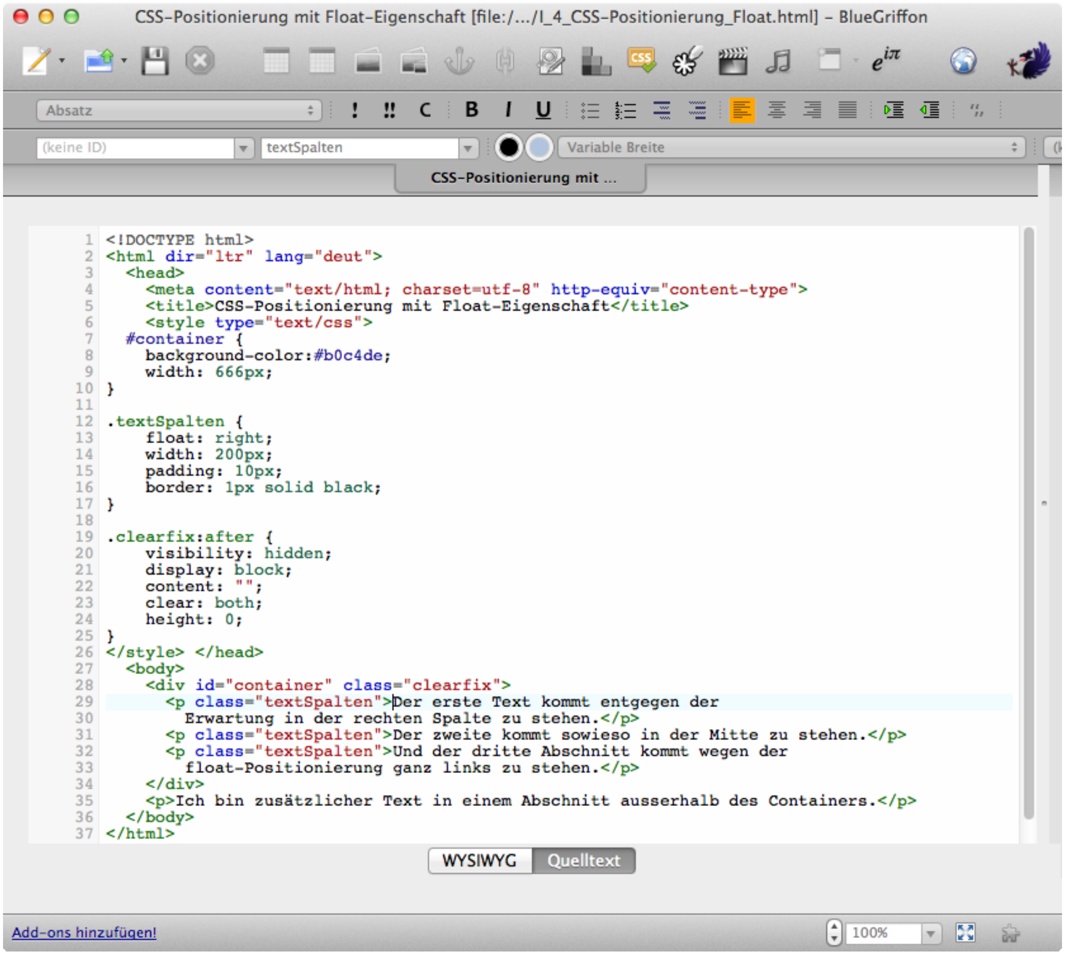

Abbildung 4.6: Der Quellcode einer Seite, die die Eigenschaft float einsetzt und daher auch den clearfix-Hack benötigt.

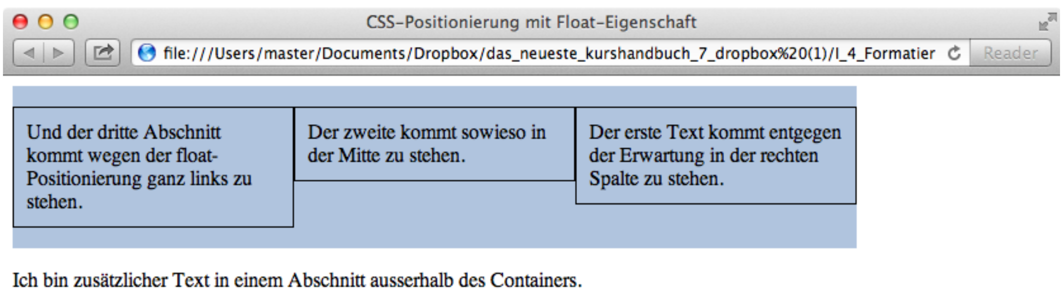

Abbildung 4.7: Die entsprechende Darstellung im Browser. Beachte, dass die Reihenfolge der Spalten gegenläufig zu jener des Dokumentenflusses ist

Ausserdem wird mit der Eigenschaft `float` ein Element aus dem normalen Dokumentfluss heraus genommen, weshalb sich die übrigen Elemente so verhalten, als wären die gefloateten Elemente nicht vorhanden. Das führt dazu, dass die Container-Box eines bzw. mehrerer gefloateter Elemente zu kleine Dimensionen erhält. Damit die Grösse der Container-Box die enthaltenen Elemente tatsächlich umfasst und die gefloateten Elemente nicht die nachfolgenden überlagern, musst du die Container-Box anhand des "Clearfix Hacks" mit einem virtuellen und unsichtbaren Element versehen, das unten an die gefloateten Elemente angefügt wird, ohne selber gefloatet zu sein (`clear:both`). Dadurch erhält der Inhalt der Container-Box die benötigte Höhe. (Siehe Abbildungen 4.6 und 4.7) Alternativ lässt sich die Eigenschaft `clear:both` auch einem auf die gefloateten Elemente nachfolgenden Element zuweisen.

Der Vorteil eines Layouts mit `float` besteht vor allem darin, dass man damit dynamische Layouts gestalten kann, die sich der aktuellen Breite des Ausgabefensters anpassen können.

10 Ausblick, Tipps & Tricks Webdesignkurshandbuch :: :: :: :: :: :: :: :: :: :: :: :: :: :: 3kanal :: digitAll :: :: :: :: :: :: :: :: :: :: :: :: :: ::