

Gestaltung von Webseiten mit HTML5 und CSS3

Formatierung und Layout mit CSS

Das Studienheft und seine Teile sind urheberrechtlich geschützt. Jede Nutzung in anderen als den gesetzlich zugelassenen Fällen ist nicht erlaubt und bedarf der vorherigen schriftlichen Zustimmung des Rechteinhabers. Dies gilt insbesondere für das öffentliche Zugänglichmachen via Internet, Vervielfältigungen und Weitergabe. Zulässig ist das Speichern (und Ausdrucken) des Studienheftes für persönliche Zwecke.

© Fernstudienzentrum Hamburg · Alle Rechte vorbehalten

Falls wir in unseren Studienheften auf Seiten im Internet verweisen/verlinken, haben wir diese nach sorgfältigen Erwägungen ausgewählt. Auf Inhalt und Gestaltung haben wir jedoch keinen Einfluss. Wir distanzieren uns daher ausdrücklich von diesen Seiten, soweit darin rechtswidrige, insbesondere jugendgefährdende oder verfassungsfeindliche Inhalte zutage treten sollten.

MMDE02J

Gestaltung von Webseiten mit HTML5 und CSS3

Formatierung und Layout mit CSS

Autor: Dr. Thomas Wecker

Fachlektorin: Dr. Florence Maurice

Werden Personenbezeichnungen aus Gründen der besseren Lesbarkeit nur in der männlichen oder weiblichen Form verwendet, so schließt dies ausdrücklich alle anderen Geschlechtsidentitäten ein.

Falls wir in unseren Studienheften auf Seiten im Internet verweisen, haben wir diese nach sorgfältigen Erwägungen ausgewählt. Auf die zukünftige Gestaltung und den Inhalt der Seiten haben wir jedoch keinen Einfluss. Wir distanzieren uns daher ausdrücklich von diesen Seiten, soweit darin rechtswidrige, insbesondere jugendgefährdende oder verfassungsfeindliche Inhalte zutage treten sollten.

Gestaltung von Webseiten mit HTML5 und CSS3

Formatierung und Layout mit CSS

Inhaltsverzeichnis

Einleitung	1
1 Struktur und Gestaltung	3
Zusammenfassung	3
2 Ein Beispiel	4
2.1 Das Beispiel im Browser mit und ohne CSS	5
2.2 Der CSS-Code	6
2.3 Browser haben eingebaute Stilvorlagen	7
Zusammenfassung	7
3 Regeln: Die CSS-Syntax	10
3.1 Gruppierte Selektoren	11
3.2 Kontextabhängige Selektoren	11
3.3 CSS-Kommentare	12
3.4 Typografie im Webdesign	12
3.5 CSS-Syntax im Überblick	14
3.6 Flexible Maßeinheiten für Cascading Style Sheets	14
Zusammenfassung	16
4 Farbmodelle und Farbangaben	18
4.1 Das RGB-Farbmodell	18
4.2 Das CMYK-Farbmodell	19
Zusammenfassung	20
5 Klassen	21
5.1 Klassenselektor „pur“	21
5.2 Pseudoklassen	22
Zusammenfassung	23
6 Verknüpfung mit HTML	24
6.1 Externe Stylesheets	24
6.2 Interne Stylesheets	24
6.3 Inline Styles	25
Zusammenfassung	26

7	Vererbung und Hierarchie	27
7.1	Vererbung	27
7.2	Die Kaskade	28
	Zusammenfassung	30
8	Voraussetzungen für CSS-basiertes Layout	31
8.1	Das Box-Modell	31
8.2	HTML-Erweiterung: die Elemente DIV und SPAN	33
8.3	ID-Selektoren	35
8.4	Flexbox	35
8.5	Float	42
	Zusammenfassung	44
9	Ein Beispiel-Layout	46
9.1	Die Beispielseite mit Layoutcontainer	46
9.2	Die Stilvorlagen im Detail erklärt	51
	Zusammenfassung	57
	Schlusswort	59
	Anhang	
A.	Lösungen der Aufgaben zur Selbstüberprüfung	60
B.	Glossar	61
C.	Literaturverzeichnis	63
D.	Abbildungsverzeichnis	64
E.	Sachwortverzeichnis	65
F.	Einsendeaufgabe	67

Einleitung

Liebe Fernschülerin, lieber Fernschüler,

im Studienheft aus dieser Reihe „Gestaltung von Webseiten mit HTML5 und CSS3“ mit dem Titel „Beschreibung der Seitenstruktur mit HTML“ habe ich Ihnen erklärt, wie man mit dieser *Auszeichnungssprache* die Elemente einer Webseite beschreibt: Absätze, Bilder, Listen, Tabellen, Formulare und natürlich die Links, ohne die das ganze World Wide Web relativ nutzlos wäre. Auf gestalterische Maßnahmen wie Formatierungen und Layout habe ich in jenem Heft vollständig verzichten müssen. Sie haben erfahren, dass es im Sprachumfang von HTML dafür gar keine Sprachelemente gibt und ich habe Sie mehrmals auf die CSS-Technologie „vertrösten“ müssen.

Jetzt ist es endlich so weit: In diesem Studienheft erhalten Sie eine Einführung in die Technologie der *Cascading Style Sheets*. Was verbirgt sich eigentlich hinter dieser kompliziert klingenden Technik? Wenn Sie sich schon einmal mit etwas fortgeschrittenen Methoden der Textverarbeitung beschäftigt haben (z.B. mit Microsoft Word, Open Office oder Pages), dann haben Sie bestimmt auch schon *Formatvorlagen* benutzt. In der Textverarbeitung verwendet man sie, um bestimmte Textelemente nicht jedes Mal neu formatieren zu müssen, sondern man weist ihnen immer wieder die gleiche Formatvorlage – z.B. „Überschrift 1“ – zu. Wie dieses Format dann tatsächlich aussieht, kann jederzeit (auch noch nachträglich) festgelegt werden. Sollten Sie auch schon einmal mit einem professionellen Layoutprogramm gearbeitet haben (z.B. mit Adobe InDesign), ist Ihnen dasselbe Konzept in Form der sogenannten Stilvorlagen begegnet. Bei den *Cascading Style Sheets*¹ handelt es sich schlicht und einfach um die Übertragung dieses Konzepts der *Stilvorlagen* bzw. *Formatvorlagen* auf Webseiten. In der aktuellen Spezifikation CSS3 gehen die Möglichkeiten allerdings bei Weitem über das reine Formatieren von Texten hinaus.

Wie auch schon im Studienheft über HTML ist es nicht meine Absicht, Ihnen sämtliche Details zu dem Thema CSS zu präsentieren, sondern ich werde mich auf das Wesentliche konzentrieren. Ich hoffe, es wird Ihnen auch diesmal wieder Spaß machen, diese wichtige Technologie zu erlernen. Also legen wir los!

Dr. Thomas Wecker

1. Es ist auch die Schreibweise in einem Wort möglich: *Stylesheets*.

1 Struktur und Gestaltung

In diesem Kapitel möchte ich Ihnen zunächst noch einmal die Notwendigkeit für die Einführung der CSS-Technologie deutlich machen.

Die Grundidee der Cascading Style Sheets ist – genau wie bei Stil- oder Formatvorlagen – die *Trennung von Struktur und Gestaltung*. HTML wurde als Sprache zur Auszeichnung von Überschriften, Absätzen, Listen, Bildern und anderen Strukturen einer Seite definiert („Markup Language“). Nach und nach wurden in den 90er-Jahren zusätzliche HTML-Tags eingeführt (vor allem das `font`-Tag), um auch das Erscheinungsbild beeinflussen zu können: Schrifttyp, -größe und -farbe, Hintergrundfarbe und -bild. Webdesigner setzten darüber hinaus eine ganze Reihe von Tricks ein, um Gestaltungselemente wie Ränder, Spaltensatz und Leerräume zu implementieren. Die folgende Abbildung zeigt ein typisches Beispiel, bei dem die Befehle zur Formatierung schon 50 Prozent des gesamten Codes ausmachen:

CSS führt zur Trennung von Struktur und Gestaltung

```
<body bgcolor="#FFFF99" text="#000000" background="images/backgrnd.gif">
<font face="Verdana, Arial, Helvetica, sans-serif">Das ist ein Text, der
mit <b><font color="#FF0033">HTML-Tags</font></b> formatiert worden ist.
Man sieht, wie der Code &quot;<i>aufgebläht</i>&quot; wird. Dadurch wird
die <b><font color="#FF0033">Wartung</font></b> erschwert und die <b><font
color="#FF0033">Ladezeit</font></b> erhöht. Der Einsatz von<font
color="#FF0033"> <b>CSS-Stilvorlagen</b> </font>befreit den <font
face="Courier New, Courier, mono">Code</font> von diesem
<i>überflüssigen</i> Ballast.</font>
</body>
```

Abb. 1.1: Beispiel für einen mit Formatierungen durchsetzten HTML-Code

Durch die Einführung von CSS wurden diese Tags und Tricks zum Glück wieder überflüssig. Die Funktion von HTML konnte wieder auf seine ursprüngliche Aufgabe reduziert werden: die Struktur eines Dokuments zu beschreiben. Bei der Anwendung von CSS wird unabhängig vom HTML-Code dem Browser in einer Reihe von Regeln mitgeteilt, wie das *Erscheinungsbild* eines Dokuments sein soll. Cascading Style Sheets definieren Merkmale der Typografie, des Designs und sogar das Layout eines Dokuments unabhängig von seiner Struktur.

Das Konzept der Cascading Style Sheets umfasst die „Sprache“, mit der diese Gestaltungsregeln dem Browser mitgeteilt werden, deren Verknüpfung mit dem HTML-Code sowie die konkreten Auswirkungen der Regeln auf einzelne Strukturelemente. Im nächsten Kapitel werden wir das an einem einfachen Beispiel sehen.

Zusammenfassung

- Mit CSS ist eine konsequente Trennung von Struktur und Gestaltung möglich.
- CSS definiert Typografie, Design und Layout.
- CSS basiert auf Regeln, die in einer eigenen Sprache beschrieben werden.
- Die CSS-Regeln müssen mit dem HTML-Code verknüpft werden.

2 Ein Beispiel

*Das CSS-Prinzip kann man am einfachsten anhand eines Beispiels verstehen.
Das möchte ich Ihnen nun demonstrieren.*

Der folgende Code sollte Ihnen keinerlei Schwierigkeiten machen, denn er enthält nur ganz einfache HTML-Tags, die Sie bereits im ersten Studienheft dieser Reihe kennengelernt haben.

```
<!DOCTYPE HTML>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>Willkommen CSS</title>
    <link href="style1.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <h1>Willkommen in der Welt<br>der <em>Style Sheets!</em></h1>
    <p>Style Sheets zu benutzen ist ganz einfach:</p>
    <ol>
      <li>schreibe die Regeln auf</li>
      <li>sichere sie in einer <em>eigenen</em> Textdatei</li>
      <li>und wende den Stil auf viele Seiten gleichzeitig an!</li>
    </ol>
    <h2>Wie die Regeln geschrieben werden</h2>
    <p>kommt im nächsten Lernschritt dran...</p>
  </body>
</html>
```

Abb. 2.1: Ein einfaches HTML-Dokument

Die einzige Stelle in diesem Code, die Sie natürlich noch nicht verstehen können, ist die Zeile nach dem `title`-Element:

- `<link href="style1.css" rel="stylesheet" type="text/css">`

Aber versuchen Sie doch trotzdem einfach selbst einmal, sich daraus einen Reim zu machen, bevor ich Ihnen alles erkläre, denn manches könnte Ihnen sogar bekannt vorkommen. Also, diese Zeile teilt dem Browser Folgendes mit:

- **link:** „Hier kommt die Verknüpfung zu einem anderen Dokument.“
- **href="style1.css":** „Das ist der *relative* Pfad zu diesem Dokument.“

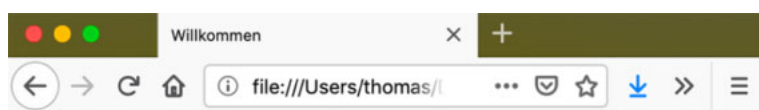
(Das kommt Ihnen doch bestimmt bekannt vor, oder? Stimmt, das ist exakt dasselbe Attribut, das Sie bereits aus dem ersten Studienheft vom `a`-Tag für die Auszeichnung von Links/Verweisen kennen!)

- **rel="stylesheet":** „Das Attribut `rel` legt fest, in welcher Relation die HTML-Datei und die mit ihr verknüpfte Datei `style1.css` zueinander stehen. Der Attributwert `stylesheet` bedeutet, dass `style1.css` die für das HTML-Dokument bevorzugten Stilvorlagen enthält, die automatisch immer angewandt werden sollen.“
- **type="text/css":** „Das ist der sog. MIME-Typ und bedeutet: Das Dokument enthält lesbaren Text in Form von CSS-Angaben“ (seit HTML5 ist diese Angabe nicht mehr zwingend erforderlich).

Wie Sie am Beispielcode in der Abbildung oben auch erkennen können, steht in diesem Beispiel die Zeile `<link ...>` im `head`-Bereich des HTML-Dokuments direkt nach dem `title`-Tag. Sie kann aber auch an anderer Position im HTML-Code stehen – auf jeden Fall aber innerhalb des `head`-Bereichs.

2.1 Das Beispiel im Browser mit und ohne CSS

Die Zeile `<link ...>` sorgt offensichtlich für eine Verknüpfung zwischen der HTML-Datei und dieser Datei `style1.css`. Diese Datei enthält die Regeln für die Stilvorlagen, von denen im vorigen Kapitel bereits die Rede war. Sehen wir uns aber zunächst einmal an, wie unser Beispieldokument **ohne diese Zeile** (also ohne CSS) im Firefox aussehen würde:



Willkommen in der Welt der *Style Sheets*!

Style Sheets zu benutzen ist ganz einfach:

1. schreibe die Regeln auf
2. sichere sie in einer *eigenen* Textdatei
3. und wende den Stil auf viele Seiten gleichzeitig an!

Wie die Regeln geschrieben werden

kommt im nächsten Lernschritt dran...

Abb. 2.2: Das Beispieldokument ohne CSS

Es ist erwartungsgemäß keine besonders interessante HTML-Seite. Jetzt schauen wir uns das Dokument mit dem kompletten Quellcode aus dem ersten Codebeispiel in diesem Kapitel im Firefox an:



Vorher-
Nachher:
Große Wirkung
durch CSS

Abb. 2.3: Das Beispieldokument mit CSS

Diese eine Zeile bewirkt einen ziemlich großen Unterschied, oder? Nicht, dass ich behaupten würde, dass diese Seite nun besonders schön und geschmackvoll gestaltet wäre. Es geht mir einfach darum, durch einige besonders auffallende Merkmale den Unterschied zum Beispiel ohne CSS hervorzuheben.

2.2 Der CSS-Code

Jetzt müssen wir uns aber endlich mit dem Inhalt dieses neuen Dokuments **style1.css** beschäftigen:

```
body {
    font-family: Helvetica, Arial, sans-serif;
    background: yellow;
}

h1 {
    text-align: center;
    font-size: 36px;
    color: blue;
}

h2 {
    font-size: 28px;
    color: green;
}

em { color: fuchsia; }

p {
    text-transform: uppercase;
    font-weight: bold;
    font-size: 12px;
}

li { font-size: 18px; }
```

Abb. 2.4: Das externe Stylesheet **style1.css**

Es handelt sich um ein externes Stylesheet. Wenn es ein externes gibt, dann muss es wohl auch ein internes geben. Stimmt, das werden wir später noch kennenlernen. Aber diese externe Variante ist bei Weitem die Wichtigste. Die Vorteile dieses Verfahrens werden Ihnen in einem späteren Kapitel noch deutlich werden. Jetzt zum Inhalt von **style1.css**: Sie sehen eine Liste von Ihnen bekannten HTML-Tags. Es sind tatsächlich genau die Tags, die in unserer HTML-Beispielseite vorkommen. Nach jedem Tag folgt in geschweiften Klammern eine Gestaltungsregel, die Sie wahrscheinlich zunächst einmal auch ohne große Erklärung verstehen können.

Hinweis:

Sie finden alle Listings zu den in diesem Studienheft gezeigten Beispielen im heft-bezogenen Downloadbereich Ihrer Lernplattform. Da Sie die Beispiele im Rahmen der Aufgaben zur Selbstüberprüfung auch eigenständig erstellen sollen, empfehle ich Ihnen: Benutzen Sie die Musterlösungen erst zur Kontrolle, **nachdem** Sie den Code selbst erstellt haben. Sonst ist der Lerneffekt gering!

2.3 Browser haben eingebaute Stilvorlagen

Sehen wir uns noch mal das „nackte“ HTML-Beispiel ohne CSS im vorigen Abschnitt an: An diese Schriftarten und -größen sind Sie ja noch aus der Arbeit mit dem vorausgegangenen Studienheft gewöhnt. Dabei spielt es keine Rolle, ob Sie sich die Beispiele damals mit Firefox, mit Google Chrome oder einem anderen Browser angesehen haben. Offensichtlich „wissen“ alle Browser, wie sie eine **h1**-Überschrift, eine **h2**-Überschrift, einen Absatz oder eine Liste darstellen müssen. Die Browser haben sozusagen eine „eingebaute Stilvorlage“. Das W3C hat vorgeschlagen, wie die einzelnen Tags von ihren Programmen dargestellt werden sollen, d.h., praktisch alle gängigen Browser basieren auf ähnlichen fest einprogrammierten Darstellungsregeln: Andere für die Darstellung relevanten Einstellungen können die Nutzer in den Voreinstellungen ihres Browsers vornehmen. Dort ist als Standardschrift meist die Times New Roman mit einer Standard-Schriftgröße von **16 Punkt** bei normalen Absätzen voreingestellt.

Was passiert, wenn uns als Webdesigner diese Darstellung nicht gefällt? Dann müssen wir den Browsern eben in geeigneter Form mitteilen, wie wir die **h1**-Überschrift, die **h2**-Überschrift, die Absätze, die Listen usw. dargestellt haben möchten. Und genau das passiert in der Datei **style1.css**: Für jedes HTML-Tag, das im Code der Seite vorkommt, wird in einer Regel festgelegt, wie der Browser dieses Tag darstellen soll, also Schriftart, -größe, -schnitt und -farbe. Für den **body** wird auch noch eine Hintergrundfarbe definiert. Auch wenn die Regeln in **style1.css** praktisch selbsterklärend sind, werde ich Ihnen deren genaue Syntax gleich im nächsten Kapitel erklären.

Hinweis:

Wir reden hier nicht davon, wie einzelne User die Standardeinstellung für Schriftart und -größe in den Voreinstellungen ihres Browsers nach deren persönlichem Geschmack anpassen können. Es geht vielmehr darum, als Webdesigner/-in die Formatierung und Gestaltung einer Seite mithilfe von CSS verbindlich festzulegen.

Zusammenfassung


- Ein externes Stylesheet wird durch den Befehl **link** mit der HTML-Datei verknüpft.
- Das wichtigste *Attribut* dieses Befehls lautet **href** und enthält den relativen Pfad zu dem externen Stylesheet.
- CSS-Dateien tragen die Endung **.css**.
- Browser haben „eingebaute Stilvorlagen“, die durch CSS-Regeln „überschrieben“ werden.
- Ein HTML-Dokument kann durch ein Stylesheet ein vollkommen anderes Erscheinungsbild im Browser erhalten.


Aufgabe zur Selbstüberprüfung

Überprüfen Sie nun bitte Ihr neu erworbenes Wissen. Lösen Sie die Aufgaben zunächst selbstständig und vergleichen Sie immer erst anschließend Ihre Lösungen mit den Lösungen, die Sie im heftbezogenen Downloadbereich Ihrer Lernplattform finden.

- 2.1
- Bitte starten Sie Ihren Code-Editor mit einem leeren Dokument (Sie haben sich ja sicher schon im vorausgehenden Studienheft für einen „Lieblingseditor“ entschieden).
 - Schreiben Sie sorgfältig den CSS-Code für das externe Stylesheet aus dem letzten Screenshot in diesem Kapitel in das Dokument.

Achten Sie dabei bitte sehr genau darauf, dass Sie keinen Syntaxfehler machen, d.h., vergessen Sie keine der geschweiften Klammern, kein Semikolon, kein Komma und auch keinen Doppelpunkt.

Sie erhalten die geschweiften Klammern auf Ihrer Tastatur mit  +  bzw.  + .





Auf dem **Mac** erhalten Sie die geschweiften Klammern mit  +  bzw.  + .

- Speichern Sie dieses Dokument unter dem Namen **style1.css** im Ordner **html-uebung** Ihrer Übungswebsite. Achten Sie bitte darauf, dass Sie die richtige Dateiendung **.css** verwenden!
- Verwenden Sie bitte eines Ihrer alten HTML-Dokumente als Vorlage und speichern Sie es unter dem neuen Namen **css-beispiel.html**.
- Löschen Sie den Inhalt des **body**-Bereichs und geben Sie stattdessen den Quellcode für das HTML5-Dokument aus dem ersten Screenshot in diesem Kapitel ein.
- Geben Sie im **head**-Bereich genau wie im Beispiel gezeigt die Zeile `<link ...>` mit der Verknüpfung zum externen Stylesheet ein.

Speichern Sie die Datei und sehen Sie sich das Resultat in Ihrem Browser an. Es sollte jetzt genau wie der entsprechende Screenshot in diesem Kapitel aussehen (also die Seite mit dem gelben Hintergrund).

Hinweise zu den praktischen Übungen

Bei der Eingabe von HTML- und CSS-Code kann es leicht vorkommen, dass sich Syntaxfehler „einschleichen“.

HTML-Syntaxfehler können Sie leicht vermeiden, wenn Sie sich den Quellcode via Firefox ansehen (**Extras>Webentwickler>Seitenquelltext** anzeigen – bzw. Tastaturbefehl  +  – auf dem Mac  + ). Da werden die meisten Syntaxfehler in Rot dargestellt.

CSS-Syntaxfehler können Sie leicht vermeiden, wenn Sie die *Web-Konsole* von Firefox benutzen. Um diese zu öffnen, wählen Sie den Befehl **Web-Konsole** aus dem Untermenü **Web-Entwickler** im Firefox-Menü (wenn Sie die Menüleiste aktiviert haben oder macOS benutzen, **Extras/Web-Entwickler/Web-Konsole**). Wahlweise können Sie auch den Tastaturbefehl **Strg** + **⇧** + **K** benutzen (auf dem Mac **⌘** + **⇧** + **K**).

Die Web-Konsole wird dann im unteren Teil des Browser-Fensters angezeigt (Sie müssen gegebenenfalls noch den CSS-Filter aktivieren, indem Sie zuerst auf das Trichter-Icon klicken und dann auf CSS). Für das folgende Beispiel wurde absichtlich die Eigenschaften **font-family** falsch als **front-family** und **font-weight** falsch als **font-weigt** geschrieben, was in der Web-Konsole in der Rubrik CSS sofort angezeigt wird, wie folgende Abbildung zeigt:



Abb. 2.5: Die Web-Konsole von Firefox zeigt einen CSS-Syntaxfehler an

Wie Sie sehen, werden diese CSS-Syntaxfehler zuverlässig von der Web-Konsole erkannt. Mit einem Klick auf die jeweils rechts in der Zeile angegebene Zeilennummer wird sofort der zugehörige Code eingeblendet und der Fehler kann unmittelbar behoben werden (in diesem Fall muss man dann anschließend noch auf **Speichern** klicken!).

3 Regeln: Die CSS-Syntax

In diesem Kapitel erkläre ich Ihnen ausführlich, wie die Regeln eines Stylesheets aufgebaut sind und was Sie außerdem über deren Anwendung wissen müssen.

Sie haben ja eben in der letzten Übung selbst schon einige CSS-Regeln in Ihrem Code-Editor eingegeben. Deshalb wird es Ihnen jetzt nicht schwerfallen, die Syntax zu erlernen. Wie Sie nämlich in dem Beispiel gesehen haben, besteht eine Regel aus zwei Teilen, *Selektor* und *Deklaration*:

```
h1 {font-family: Helvetica, sans-serif; }
```

h1 ist der Selektor, der Text zwischen den geschweiften Klammern ist die Deklaration. Diese besteht ihrerseits aus zwei Teilen, der *Eigenschaft* (hier **font-family**) und dem entsprechenden *Wert* (hier **Helvetica, sans-serif**), der nach dem Doppelpunkt angegeben wird. Dabei ist **sans-serif** eine sog. *generische Schriftart*. Die Schriftart Helvetica ist eine serifenlose Schrift (eine Schrift ohne „Verzierungen“). Ist diese auf dem Rechner des Webseiten-Besuchers gar nicht vorhanden, bietet man mit der generischen Schriftart dem Browser die Möglichkeit, eine Schriftart auszuwählen, die zumindest vom grundlegenden Schrifttyp her der Helvetica entspricht.

Wenn für einen Selektor innerhalb der geschweiften Klammern mehrere Deklarationen angegeben werden, müssen diese durch ein Semikolon voneinander getrennt werden. Bei der letzten Deklaration darf das Semikolon weggelassen werden, aber wir verwenden es konsequent auch dort.

Eine CSS-Regel besteht aus Selektor und Deklaration

Der Selektor ist in den bisherigen Beispielen immer ein HTML-Tag gewesen. Er signalisiert dem Browser, dass für dieses Tag ab sofort nicht mehr die interne Standard-Stilvorlage gilt. Wie der Browser das Tag jetzt darzustellen hat, steht dann in der *Deklaration*. In unserem Beispiel haben Sie schon gesehen, dass man auch mehrere Deklarationen hintereinander schreiben kann.

Diese *müssen* dann durch ein *Semikolon* voneinander getrennt werden:

```
h1 {
  font-family: Helvetica , sans-serif; font-size: 80%;
}
```

Die Deklaration besteht aus Eigenschaft und Wert

Wie schon beim HTML-Code spielen Leerzeichen oder Zeilenumbrüche beim CSS-Code keine Rolle. Das vorige Beispiel könnte demnach auch so geschrieben werden:

```
h1 {
  font-family: Helvetica, sans-serif;
  font-size: 80%;
}
```

Hier möchte ich noch eine Anmerkung machen: Im Rahmen der aktuellen CSS-Spezifikation gibt es natürlich eine Vielzahl von Eigenschaften mit den dazugehörigen Werten. Es ist unmöglich und auch gar nicht notwendig, diese alle auswendig zu kennen.

Wenn Sie die Beispiele, die ich Ihnen im Laufe dieses Studienhefts präsentieren werde, kennen, reicht das zunächst vollkommen aus. Zum Nachschlagen weiterer CSS-Eigenschaften empfehle ich die CSS-Kurzreferenz des Wikis SELFHTML:

- <https://wiki.selfhtml.org/wiki/CSS>

3.1 Gruppierte Selektoren

Selektoren können auch **gruppiert** werden, d.h., eine Deklaration gilt dann für mehrere Selektoren gleichzeitig:

```
h1, h2, h3 { font-style: italic; color: green; }
```

Alle drei Überschriften-Tags werden jetzt in kursiver und grüner Schrift dargestellt. Die Gruppierung ermöglicht eine kompaktere und übersichtlichere Schreibweise von Stylesheets. Beachten Sie bitte, dass bei gruppierten Selektoren zwischen den einzelnen Tags immer ein Komma steht! Dagegen muss zwischen mehreren Deklarationen innerhalb der geschweiften Klammern immer ein Semikolon stehen!

3.2 Kontextabhängige Selektoren

Eine wichtige Rolle bei komplexeren Stylesheets spielt die Möglichkeit, Selektoren **kontextabhängig** zu definieren. Dafür gleich ein kleines Beispiel:

```
h2 em {
  color: blue;
  font-family: Courier, monospace;
}
```

In diesem Fall würde nur unter folgender Bedingung ein Text in blauer Courier-Schrift dargestellt:

- Der Text wird mit dem **em**-Tag hervorgehoben.
- Die **em**-Struktur selbst befindet sich innerhalb einer **h2**-Überschrift.

Die CSS-Regel würde also bei folgendem HTML-Code zur Wirkung kommen:

```
<h2>Das ist <em>blauer Text in Courier</em> und hier geht es
normal weiter</h2>
```

An diesem Beispiel kann ich Ihnen gut die Terminologie erklären, die in der Fachliteratur in solchen Fällen verwendet wird: Die **h2**-Überschrift ist in unserem Beispiel das *Elternelement*. Das **em**-Tag ist als *Kindelement* der unmittelbare *Nachfahre* dieses Elternelements. Deshalb bezeichnet man solche Selektoren auch als „*Nachfahre-Selektoren*“ (engl. „descendant selectors“). Neuerdings werden sie auch als *Nachfahrenkombinatoren* bezeichnet. Sie werden später noch viele Beispiele kennenlernen, bei denen man mithilfe solcher Kombinatoren die CSS-Stilvorlagen sehr „zielgenau“ zur Anwendung bringt.

Kontextabhängige Selektoren werden auch als *Nachfahre-Selektoren* oder *Nachfahrenkombinatoren* bezeichnet und erlauben „zielgenaue“ Anwendung von Stilvorlagen



Wenn in einem anderen Zusammenhang Texte mit **em** hervorgehoben werden, wirkt sich diese Stilvorlage darauf nicht aus:

```
<h1>Das ist
  <em>kein blauer Text in Courier</em></h1>
<p>Das ist
  <em>auch kein blauer Text in Courier</em></p>
```

Hinweis:

Hier muss ich Sie warnen: Der Unterschied bei der Syntax zwischen gruppierten und kontextabhängigen Selektoren (Kombinatoren) ist **lediglich ein Komma** (statt nur ein Leerzeichen)! Vergleichen Sie daraufhin noch mal die CSS-Beispiele der vorausgegangenen Abschnitte.

3.3 CSS-Kommentare

Erinnern Sie sich noch an die Kommentare im HTML-Code? Damals habe ich Ihnen erklärt, wie wichtig es ist, den Quellcode mit Kommentaren zu ergänzen, damit man ihn auch noch nach längerer Zeit versteht. Genauso verhält es sich bei Stylesheets. In diesem Fall werden Kommentare mithilfe des schon aus dem HTML bekannten Slash „/“ zusammen mit einem Stern gebildet:

```
/* Das ist ein CSS-Kommentar */
```

Der Kommentar wird abgeschlossen mit einem Stern und einem Slash (in umgekehrter Reihenfolge). Sie werden im Laufe dieses Studienhefts noch sehen, wie groß und komplex Stylesheet-Dateien werden können. Deshalb sollten Sie auf jeden Fall immer daran denken, Ihren CSS-Code mit Kommentaren zu strukturieren. Wie schon bei HTML können Sie den CSS-Code beliebig durch Zeilenumbrüche und Leerzeichen strukturieren und dadurch übersichtlicher machen.

Kommentare werden während des Design-Prozesses auch gerne benutzt, um bestimmte CSS-Regeln „auszukommentieren“ und damit vorübergehend wirkungslos zu machen.

3.4 Typografie im Webdesign

Typografie ist die Lehre vom richtigen Umgang mit Text und Schriften. Im Printbereich spielt Typografie eine ganz wichtige Rolle, denn das Drucken von Text ist schließlich eine uralte kulturelle Errungenschaft. Grafikdesigner, die im Bereich der Printmedien arbeiten, geben sich in der Regel sehr viel Mühe bei der Auswahl geeigneter Schriften, von denen heutzutage Tausende zur Verfügung stehen. Wie sieht es mit der Typografie im Webdesign aus? Bis vor gar nicht so langer Zeit war das nicht so toll, wie man es sich als anspruchsvoller Grafiker bzw. Typograf wünschen würde. Das lag einfach daran, dass man als Webdesigner nur die Schriften einsetzen durfte, die auch auf dem Rechner des Besuchers der Webseite installiert sind. Das ist der Hintergrund, warum man meistens CSS-Regeln für die Schriftauswahl verwendet hat, die so wie in der letzten Abbildung des vorigen Kapitels („Das externe Stylesheet `style1.css`“) aussehen, nämlich:

```
body {font-family: Helvetica, Arial, sans-serif;}
```

Zum Verständnis dieser Deklaration müssen Sie sich folgende Tatsache vor Augen führen: Sie wissen **nichts** über die Besucher der Webseite, an deren Design Sie gerade arbeiten. Sie wissen insbesondere nicht:

Der User – das „unbekannte Wesen“

1. welches Endgerät die Besucher zum Ansehen der Webseite verwenden (Smartphone, Tablet, Notebook, Desktop-Rechner mit externem Monitor),
2. welches Betriebssystem sie einsetzen (iOS oder Android – bzw. bei stationären Rechnern Windows, macOS oder Linux),
3. welche Schriften auf dem jeweiligen Rechner installiert sind,
4. welchen Browser die Besucher verwenden,
5. welcher Viewport² auf dem Endgerät gerade zur Verfügung steht (z.B. Portrait – oder Landscape-Modus bei Smartphones), bzw. welche Größe des Browserfensters die Besucher bevorzugen (eher schmal oder maximiert, d.h. im Querformat).

Für die Auswahl der Schriften sind vor allem die ersten drei Punkte entscheidend. Ich habe aber trotzdem einmal alles zusammengestellt, was Sie über Ihre Besucher **nicht** wissen. Es ist einfach wichtig, sich diese Einschränkungen immer bewusst zu machen, wenn man sich mit Webdesign befasst, und ich werde Sie im Laufe dieses Studienhefts sicher noch einmal an diese Liste erinnern. Also, was bedeutet nun diese Schriftenliste **Helvetica**, **Arial**, **sans-serif** in dem Beispiel? Es ist eine Anweisung an den Browser des Users, die in wörtlicher Rede so lauten würde:

- „Bitte verwende für den gesamten **body**-Bereich (d.h. also für die ganze Seite) die Schrift **Helvetica**.
- Wenn diese auf ‚deinem‘ Rechner nicht installiert ist, nimm bitte stattdessen die Schrift **Arial**.
- Wenn du die auch nicht finden kannst, nimm bitte **irgendeine andere serifenlose Schrift**.

Der Hintergrund ist folgender: Die Schriftart **Helvetica** ist auf allen Macs installiert und sieht der **Arial** ziemlich ähnlich, die ja bekanntlich auf jedem Windows-PC vorhanden ist. Für alle Fälle wird dann, wie wir bereits weiter oben gesehen haben, auch noch die sogenannte *generische Schriftart* **sans-serif** angegeben, die eben „irgendeine serifenlose Schrift“ meint (*Serifen* sind übrigens – wie bereits erwähnt – diese „Verzierungen“ an den Buchstaben, wie z.B. die Häkchen bei diesem großen T).

Inzwischen hat man als Webdesigner/-in glücklicherweise nahezu unbegrenzte Möglichkeiten: Man kann in eine Webseite externe Schriften einbinden – die sog. *Webfonts*. Diese liegen auf einem Webserver und werden vom Browser heruntergeladen – so wie er das mit dem HTML- und CSS-Code auch macht. In diesem Studienheft werde ich mich auf die CSS-Grundlagen beschränken.

2. Falls Sie sich nicht mehr an diesen wichtigen Begriff erinnern können, hier nochmal die Definition: Als **Viewport** bezeichnet man den Anzeigebereich in einem Browserfenster, der den ohne Scrollen sofort sichtbaren Bereich eines Layouts enthält.

3.5 CSS-Syntax im Überblick

Hier noch mal eine vollständige Zusammenfassung der gesamten CSS-Syntax:

- Deklarationen stehen zwischen geschweiften Klammern.
- Nach der Eigenschaft folgt ein Doppelpunkt und anschließend der zugehörige Wert, gefolgt von einem Semikolon.
- Aufeinanderfolgende Deklarationen müssen durch ein Semikolon getrennt werden.
- Gruppierte Selektoren müssen durch Kommata getrennt werden.
- Kontextabhängige Selektoren (Nachfahrenkombinatoren) müssen durch Leerzeichen voneinander getrennt werden.
- Kommentare im CSS-Code werden mit der Struktur `/* ... */` gebildet und können auch mehrzeilig sein.
- Leerzeichen und Zeilenumbrüche spielen keine Rolle und können zur übersichtlichen Formatierung des Codes verwendet werden.

Dazu noch eine gute Nachricht: Wenn Sie später mit einem modernen Webeditor wie *Adobe Dreamweaver*, *BlueGriffon* oder *Google Web Designer* arbeiten, wird Ihnen die Codierung der CSS-Regeln zuverlässig von diesen Werkzeugen abgenommen. Sie müssen also weder sämtliche existierenden CSS-Eigenschaften noch deren mögliche Werte auswendig lernen. Im Studienheft über HTML habe ich betont, dass Sie **kein** wandelndes HTML-Lexikon zu sein brauchen. Für CSS gilt dies selbstverständlich ebenso. Aber wie schon bei HTML gilt auch hier: Das Prinzip zu verstehen, die Syntax zu kennen und gegebenenfalls auch einmal ein Stück Code manuell schreiben zu können, muss für jeden Webdesigner selbstverständlich sein.

3.6 Flexible Maßeinheiten für Cascading Style Sheets

Bei dem ersten CSS-Beispiel im vorausgegangenen Kapitel hatten wir die Schriftgröße in der Einheit **px** – also in Pixeln – angegeben. Das ist einfach und anschaulich, denn alle können sich etwas unter einem Pixel vorstellen. Heute muss man von Webseiten erwarten, dass sie sich flexibel an alle denkbaren Situationen anpassen. Dieses Design-Prinzip nennt man *Responsive Webdesign*:



Definition 3.1:

Responsive Webdesign (engl. *responsive* „reagierend“) ist ein gestalterisches und technisches Prinzip, mit dem Webseiten so erstellt werden können, dass deren Layout automatisch auf die Eigenschaften des jeweils benutzten Endgeräts (Desktop-Monitor, Smartphone, Tablet) reagieren kann.

Eine absolute Maßeinheit wie **px** ist beim Responsive Webdesign nicht mehr praktikabel. Dabei finden fast ausschließlich **relative Maßeinheiten** wie **%**, **em** oder **rem** Verwendung. Die Einheiten **em** und **rem** orientieren sich an der **Schriftgröße**³. Während die Interpretation der Breite eines HTML-Elements als Prozent der Breite seines jeweiligen Elternelements einfach nachzuvollziehen ist, haben Anfänger mit **em** und **rem** häufig Schwierigkeiten. Deshalb folgt hier eine eindeutige Definition:

3. Genau genommen der Größe des Großbuchstabens „M“.

Definition 3.2:

1em entspricht 100 % der vom jeweiligen **Elternelement** (z.B. vom **body**) geerbten Schriftgröße. Das **body**-Element selbst ist das oberste Element in der Hierarchie. Wurden per CSS keine anderen Regeln festgelegt und hat auch der Benutzer nichts geändert, ist hier die zugrunde gelegte Schriftgröße die Grundeinstellung des Browsers. Sie liegt normalerweise bei **16px**. Es gilt also:

1em=16px

Wird für das **body**-Element die Schriftgröße z.B. mit der Angabe **font-size:0.75em** festgelegt, ergibt sich daraus letztlich eine Schriftgröße von **16px×0.75=12px**.

Da sich die Maßeinheit **em** stets auf einen geerbten Wert bezieht, ist der vom Browser umgerechnete absolute Wert in Pixeln z.B. nicht unmittelbar bekannt. Diese Schriftgröße vererbt sich an alle weiteren Elemente, es sei denn, die anderen Elemente erhalten via CSS eine spezifischere Größenangabe.

Einen bequemen „Umrechner“ von **px** zu **em** und umgekehrt finden Sie hier:

- <https://nekocalc.com/px-to-em-converter>

Daneben gibt es auch die Einheit **rem**, die sich aus der Bezeichnung „Root-em“ („Wurzel-em“) ableitet. Ein **rem** entspricht der Schriftgröße, die für das **Wurzelement** (in HTML das **html**-Element) festgelegt wurde. Derzeit (Stand Juni 2019) lässt sich ein Trend zur Verwendung vom **rem** beobachten.

Der entscheidende Unterschied zwischen **em** und **rem** liegt in der **Vererbung**. Der **rem**-Wert bezieht sich **immer** auf die Schriftgröße des **html**-Elements. Der **em**-Wert orientiert sich dagegen an der Schriftgröße des jeweiligen Elternelements (parent Element). Bei mehrfach verschachtelten Elementen (z.B. bei Listen) muss die Schriftgröße jedes Mal um mit dem Faktor des **em**-Werts neu berechnet werden, der als Schriftgröße des Elternelements festgelegt worden ist (z.B. **0.875em**). Die Schriftgrößen in den Unterlisten werden **immer kleiner** (**0.875em von 0.875em**). Entsprechend vergrößert sich die Schrift, wenn dem Elternelement ein Wert **größer** als **1em** zugeteilt wird.

Bei **rem** ist die Berechnung der Schriftgröße wesentlich einfacher, denn sie bezieht sich immer auf **html**. In den meisten Fällen ist es deswegen bequemer, **rem** zu nutzen, und wir werden dies auch in den folgenden Beispielen anwenden.

Eine ausführliche Behandlung der Einheiten finden Sie bei diesen Adressen:

- <http://www.intensivstation.ch/css3/em-rem>
- <https://die-netzialisten.de/wordpress/em-und-rem-was-ist-der-unterschied/>
- <https://www.elmastudio.de/css-tipp-rem-als-einheit-fur-schriftgroese-nutzen>

Hinweis:

Es ist meistens sinnvoll, auch die Abstände von Elementen in **em** oder **rem** zu definieren: Diese passen sich dann automatisch an, wenn die Schriftgröße geändert wird.

Wo wir schon bei flexiblen Maßeinheiten sind: Es gibt mit **vw** und **vh** noch zwei weitere, denen Sie im Laufe Ihrer Beschäftigung mit Webdesign sicher begegnen werden. Es kann also nicht schaden, wenn Sie die hier schon mal kennenlernen:

**Definition 3.3:**

vw (viewport width): **100vw** bedeutet 100 % der **Breite** des Viewports

vh (viewport height): **100vh** bedeutet 100 % der **Höhe** des Viewports

Zusammenfassung

- Eine CSS-Regel besteht aus einem *Selektor* und einer *Deklaration* mit abschließendem Semikolon.
- Die *Deklaration* enthält eine *Eigenschaft* mit einem *Wert*.
- *Selektoren* können gruppiert werden.
- *Selektoren* können kontextabhängig sein.
- CSS-Code sollte mit *Kommentaren* versehen werden.
- Mithilfe von *Webfonts* lassen sich die typografischen Möglichkeiten im Webdesign nahezu unbegrenzt erweitern.
- Beim *Responsive Webdesign* finden fast ausschließlich **relative Maßeinheiten** wie %, **em** oder **rem** Verwendung.
- **1em** entspricht 100 % der vom jeweiligen **Elternelement** (z.B. vom **body**) geerbten Schriftgröße.
- Bei **rem** bezieht sich die Berechnung der Schriftgröße immer auf **html**.
- **100vw** bedeutet 100 % der **Breite** des Viewports, **100vh** bedeutet 100 % der **Höhe** des Viewports.
- CSS-Kommentare werden in dieser Form geschrieben: **`/* Das ist ein CSS-Kommentar */`**
- CSS-Deklarationen können temporär „auskommentiert“ werden.

Aufgabe zur Selbstüberprüfung

- 3.1
- Öffnen Sie bitte das Dokument **style1.css** aus der vorhergehenden Übung.
 - Versuchen Sie, die in diesem Kapitel erlernten Konzepte in das externe Stylesheet einzubauen: gruppierte und kontextabhängige Selektoren (Nachfahrenkombinatoren).
 - Fügen Sie passende Kommentare ein.
 - Speichern Sie dieses erweiterte Stylesheet unter dem Namen **style1-erweitert.css**
 - Passen Sie den Pfad in der zugehörigen HTML-Datei an den neuen Dateinamen des Stylesheets an und testen Sie das Resultat im Browser.
 - Natürlich dürfen Sie zu Übungszwecken auch weitere sinnvolle Veränderungen im HTML-Code der Datei **css-beispiel.html** vornehmen. Speichern sie diese anschließend unter einem anderen Namen, z.B. **css-beispiel-neu.html**.

4 Farbmodelle und Farbangaben

In diesem Kapitel erfahren Sie, welche Farbmodelle es grundsätzlich zur Definition von Farben gibt und welche für die Farbdarstellung auf Bildschirmen relevant sind.

Der richtige Umgang mit Farbe setzt leider etwas Theorie voraus. Ich werde Ihnen in diesem Kapitel so einfach und kompakt wie möglich die notwendigen Grundlagen erklären. Diese Kenntnisse werden wir dann gleich im nächsten Kapitel anwenden.

4.1 Das RGB-Farbmodell

Monitore
„verstehen“
nur RGB

Computermonitore stellen Farben durch Überlagerung der Grundfarben **Rot**, **Grün** und **Blau** dar. Die drei Grundfarben werden auch als **Primärfarben** bezeichnet. Jede Farbe hat also einen Rotwert, einen Grünwert und einen Blauwert. Im **RGB-Modell** wird die Intensität der einzelnen Komponenten des Modells auf einer 256-stufigen Skala (von 0–255) angegeben. 256 Stufen, weil die Farbinsensitivität in einer Grauwerttabelle von einem Byte Tiefe (1 Byte = 8 Bit = $2^8 = 256$) hinterlegt ist, wobei 0 für die geringste und 255 für die höchste Intensität spricht. Jede der RGB-Grundfarben kann dabei einen Wert zwischen 0 und 255 haben (also insgesamt $2^8 = 256$ verschiedene Werte).

Im RGB-Modell stehen demnach insgesamt $2^8 \times 2^8 \times 2^8 = 16,7$ Millionen verschiedene Farben für jedes Pixel zur Verfügung. Diese Darstellung wird bei PCs auch als **True Color** bezeichnet. Man nennt diesen Farbraum auch **24-Bit-Farben** (3×8 Bit pro Pixel = 24 Bit). Das Wertetripel (255, 255, 255) repräsentiert Weiß, (0, 0, 0) steht für Schwarz. Diese Kombination von Farben wird als additive Farbmischung bezeichnet, das RGB-Farbmodell nennt man deshalb auch **additives Farbmodell**.

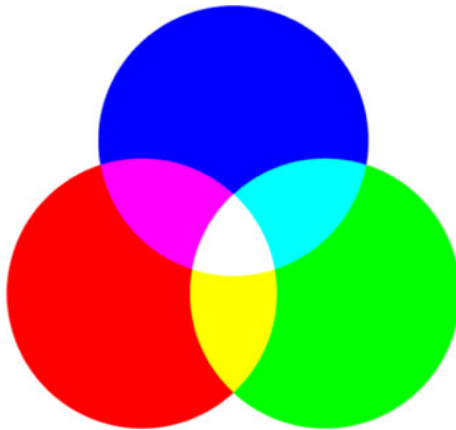


Abb. 4.1: Die additive Farbmischung

Werden die drei Grundfarben in maximaler Intensität miteinander addiert, so entsteht Weiß. Ist die Intensität aller Grundfarben Null, ist das betreffende Pixel natürlich Schwarz.

Farbangaben im Web

Man kann für manche Farben auch englische Farbnamen verwenden:








 Black = "#000000"	 Green = "#008000"
 Silver = "#C0C0C0"	 Lime = "#00FF00"
 Gray = "#808080"	 Olive = "#808000"
 White = "#FFFFFF"	 Yellow = "#FFFF00"
 Maroon = "#800000"	 Navy = "#000080"
 Red = "#FF0000"	 Blue = "#0000FF"
 Purple = "#800080"	 Teal = "#008080"
 Fuchsia = "#FF00FF"	 Aqua = "#00FFFF"

Abb. 4.2: Einige CSS-Farbnamen⁴

In der Abbildung werden auch die zugehörigen Hexadezimalwerte angegeben, die im Webdesign zur Spezifikation von Farben nach wie vor verbreitet sind. Hexadezimalwerte stellen die Umrechnung von dezimalen RGB-Werten in das Hexadezimalsystem dar. Die Zahl 255 entspricht hexadezimal der „Zahl“ `ff`⁵. Die Hexadezimalzahlen sind 6-stellig, wobei die ersten für Rot, die nächsten für Grün und die hinteren für Blau stehen. Ihnen ist die Raute `#` vorangestellt. Die Deklaration `#ffffff` entspricht der Farbe Weiß⁶. In CSS können Farben auch intuitiver mit der `rgb()`-Funktion angegeben werden: Die Angaben `#ffffff` und `rgb(255, 255, 255)` sind identisch.

Mehr zu Farbmodellen und Farbangaben in CSS finden Sie hier:

- <https://wiki.selfhtml.org/wiki/CSS/Wertetypen/Farbangaben>

4.2 Das CMYK-Farbmodell

In einem Lehrgang für Webdesigner wollen wir natürlich nicht tief in das Thema „Farbe im Medium Print“ einsteigen. Aber über ein Grundwissen über das klassische Medium auf bedrucktem Papier sollte auch jeder Webdesigner verfügen: Im Printbereich werden die Farben nach dem **CMYK-Farbmodell**⁷ definiert, das heute alle PC-Benutzer von ihren Desktopdruckern kennen: Beim CMYK-Modell wird mit den vier Druckfarben Cyan, Magenta, Gelb und Schwarz gearbeitet. Nur mit diesen Farben ist der Druck auf einem nicht leuchtenden Medium wie Papier möglich.⁸

4. Eine ausführliche Liste mit Farbnamen finden Sie hier:
https://developer.mozilla.org/de/docs/Web/CSS/color_value

5. Man kann Groß- oder Kleinbuchstaben verwenden.

6. Wenn die Hexadezimalzahl aus identischen Zeichen besteht, kann die Angabe auch verkürzt werden:
`#ffffff` kann als `#fff` geschrieben werden.

7. CMYK steht als Abkürzung für Cyanblau + Magentarot + Yellow + K (= Kontrast = Schwarz).

8. Beim professionellen Offset-Druck können zusätzlich auch noch Sonderfarben eingesetzt werden (z. B. aus dem PANTONE-System).

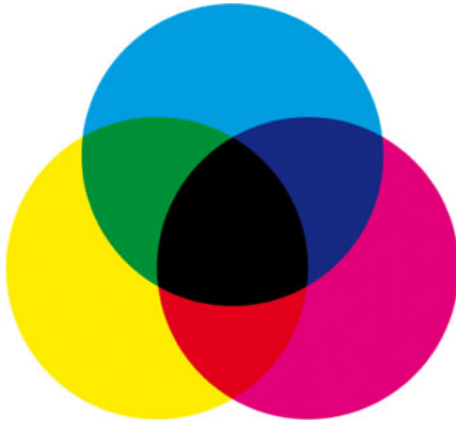


Abb. 4.3: Das subtraktive Farbmodell

Das Prinzip der Farbmischung im CMYK-Farbraum beruht darauf, dass auf die zu bedruckende Fläche Farbpigmente aufgebracht werden, die einfallendes Licht teilweise schlucken und andere Teile auf die reflektierende Fläche durchlassen. Erst durch diese Filterung und Reflexion von Licht entstehen die endgültig sichtbaren Farben. Der Fachbegriff dafür ist „**subtraktive Farbmischung**“. Durch den unterschiedlichen Farbraum sind Bildschirmdarstellung und Druckergebnis in den seltensten Fällen identisch.

Zusammenfassung

- Computermonitore stellen Farben durch Überlagerung der Grundfarben Rot, Grün und Blau dar.
- Jede Farbe hat einen Rotwert, einen Grünwert und einen Blauwert.
- Bei RGB wird die Intensität der einzelnen Komponenten auf einer 256-stufigen Skala (von 0–255) angegeben.
- RGB stellt 16,7 Millionen verschiedene Farben für jedes Pixel zur Verfügung.
- Hexadezimalwerte stellen die Umrechnung von dezimalen RGB-Werten in das Hexadezimalsystem dar.
- Hexadezimale Farbangaben sind 6-stellig, wobei die ersten für Rot, die nächsten für Grün und die hinteren für Blau stehen. Ihnen ist die Raute # vorangestellt (z.B. **#ffffff** für Weiß).
- In CSS können Farben auch mit der **rgb()**-Funktion angegeben werden – die Angaben **#ffffff** und **rgb(255, 255, 255)** sind identisch.

5 Klassen

In diesem Kapitel lernen Sie eine wichtige Anwendung der CSS-Technologie kennen. Mithilfe von Klassen sind Sie in der Lage, in einem HTML-Dokument z.B. mehrere Absätze unterschiedlich zu formatieren.

Bleiben wir gleich bei diesem Fall: Mit der bisher zur Verfügung stehenden Technik können wir für Absätze, d.h. für den Selektor `p`, genau **eine** CSS-Regel schreiben:

```
p {font-size:12px; color:red;}
```

Alle Absätze in dem zugehörigen HTML-Dokument werden dann vom Browser in roter Schrift mit der Schriftgröße 12 Pixel⁹ dargestellt. Was ist denn aber, wenn Sie in demselben Dokument ab und zu auch mal einen Absatz in einer kleineren Schrift benötigen? Sehen Sie sich dieses Studienheft an: Ich verwende hier gerade normalen „Fließtext“, schreibe aber gelegentlich am rechten Rand auch einen Hinweis (Marginalie) in kleinerer Schrift. Wie können wir so etwas auf einer Webseite realisieren?

Dieser Marginaltext ist auch ein Absatz!

Wenn wir mehrere Absätze mit **unterschiedlichen** Stilen in unserem Dokument verwenden möchten, können Sie für **dasselbe** HTML-Element unterschiedliche Klassen definieren. Klassen – in der präzisen Terminologie heißen sie Klassenselektoren – werden ganz einfach gebildet, indem man hinter dem Typselektor nach einem Punkt einen beliebigen Namen angibt. Hier gleich ein Beispiel für zwei verschiedene Absatztypen mit den willkürlichen Klassennamen „`typ1`“ und „`typ2`“:

Mit Klassen kann auch auf einer Webseite ein Marginaltext wie dieser hier formatiert werden.

```
p.typ1 {
  color:blue;
  font-family:Courier, monospace;
}
p.typ2 {
  color:red;
  font-family:Arial, sans-serif;
}
```

Im HTML-Code würde dann die Referenz auf diese Klassen folgendermaßen lauten:

```
<p class="typ1">Courier blau</P>
<p class="typ2">Arial rot</P>
```

Wie Sie sehen, wird die Klasse wie ein ganz normales HTML-Attribut (denken Sie an `width`, `src`, `href` usw.) verwendet. Das Attribut heißt in diesem Fall `class` und als Wert erscheint der Klassenname. Ist doch ganz einfach, oder? Damit haben wir genau das erreicht, was wir haben wollten!

5.1 Klassenselektor „pur“

Das Klassenkonzept kann sogar noch flexibel erweitert werden: Wenn wir einen Klassennamen ohne Typselektor definieren, kann dieser bei **beliebigen** HTML-Elementen als Attribut eingesetzt werden:

```
.blautext {color:blue;}
```

9. Für unsere simplen Beispiele können wir ruhig weiter `px` als absolutes Maß für die Schriftgröße benutzen.

Hier zwei Beispiele, wie die Referenz auf diese Klasse im HTML-Code lauten könnte:

```
<h3 class="blautext">Hier ist eine blaue Überschrift</h3>
<p class="blautext">Und jetzt kommt noch ein blauer Absatz</p>
```

Tatsächlich wird diese Form der Klassen sogar überwiegend eingesetzt: Wenn Sie später mit Dreamweaver, BlueGriffon oder Google Web Designer arbeiten, werden Sie feststellen, dass diese Webeditoren grundsätzlich Klassennamen ohne Typselektor erzeugen.

Hinweis:

In der aktuellen Lernphase ist für Sie der Klassenname `.blautext` einfach nachzuvollziehen. Bei großen Projekten vermeidet man Klassennamen wie `.blautext` eher, die sich auf die konkrete Formatierung beziehen. Wenn sich im Laufe der Entwicklung die Farbe ändert, bleibt ein unpassender Klassenname übrig, der Verwirrung stiftet.

5.2 Pseudoklassen

Mithilfe von Pseudoklassen ist es möglich, auch für solche HTML-Komponenten CSS-Stilvorlagen zu definieren, die sich **nicht** durch ein eindeutiges HTML-Element ausdrücken lassen, wie z.B. ein „noch nicht besuchter Link“. Zum Definieren solcher Pseudoklassen gibt es eine spezielle Syntax. Ich möchte mich hier auf die Pseudoklassen beschränken, die es für die einzelnen Zustände von Links gibt.

Mit Pseudoklassen werden die Links „lebendig“

Üblicherweise notiert man bei den Pseudoklassen für die Linksformatierungen zuerst das `a`-Element. Dahinter folgen ein Doppelpunkt und anschließend die Angabe eines erlaubten Schlüsselworts. Die Pseudoklassen für Links lauten im Einzelnen:

- **`a:link`**
für Verweise zu noch nicht besuchten Seiten.
- **`a:visited`**
für Verweise zu bereits besuchten Seiten.
- **`a:focus`**
für Verweise, die den Fokus erhalten (z.B. durch „Durchsteppen“ mit der Tabulator-Taste bei Benutzer(inne)n, die ohne Maus arbeiten).
- **`a:hover`**
für Verweise, die der Anwender gerade mit der Maus „berührt“.
- **`a:active`**
für gerade angeklickte Verweise.

Beachten Sie bitte, dass bei der Formatierung mit Pseudoklassen **die Reihenfolge** von Bedeutung ist. Damit alles so funktioniert, wie Sie sich das vorstellen, müssen Sie die Reihenfolge wie in der obigen Liste einhalten! Nicht benötigte Pseudoklassen können natürlich ausgelassen werden.

Dazu gleich noch ein konkretes Beispiel: Angenommen, Sie möchten, dass Ihre noch nicht besuchten Links rot und ohne die sonst übliche Unterstreichung dargestellt werden. Berührt man sie mit der Maus, werden sie dann blau und unterstrichen.

Die dazu notwendigen CSS-Regeln müssten lauten:

- `a:link {color:red; text-decoration:none;}`
- `a:hover {color:blue; text-decoration:underline;}`

In vielen Websites wird von der Gestaltung der Links durch diese Pseudoklassen Gebrauch gemacht. Dazu kann noch die grafische Gestaltung von Links mithilfe von CSS kommen (Rahmen, Kästen, Schatten, Verläufe usw.).

Weitere Informationen zu den Pseudoklassen finden Sie in diesen deutschsprachigen Beiträgen im Netz:

- <https://wiki.selfhtml.org/wiki/CSS/Selektoren/Pseudoklasse>
- <https://h5c3.de/inhalte/alle-css3-selektoren-im-detail-pseudoklassen/>

Zusammenfassung

- Klassenselektoren ermöglichen die unterschiedliche Formatierung von HTML-Elementen gleichen Typs.
- Mit Klassen können z.B. verschiedene Absatztypen in einem HTML-Dokument gestaltet werden.
- Die Referenz auf CSS-Klassen erfolgt im HTML-Code mithilfe des Attributs `class`.
- Es sind auch Klassenname ohne Typselektor möglich, die noch größere Flexibilität bieten.
- Pseudoklassen ermöglichen die Formatierung verschiedener Zustände von Links.

Aufgabe zur Selbstüberprüfung

- 5.1
- Erweitern Sie bitte das Beispiel aus dem vorletzten Kapitel: Definieren Sie in Ihrem externen Stylesheet **style1-erweitert.css** zwei Klassenname ohne Typselektor.
 - Wenden Sie diese Klassen durch Angabe des entsprechenden Klassennamens im Attribut `class` bei **zwei** Elementen in der HTML-Seite an und sehen Sie sich das Ergebnis im Browser an.

6 Verknüpfung mit HTML

Jetzt werden Sie erfahren, in welcher Weise CSS-Stilvorlagen mit dem HTML-Code verknüpft werden können.

Es gibt drei unterschiedliche Wege, wie man die CSS-Regeln mit dem HTML-Code verknüpfen kann. Die mit Abstand effizienteste haben wir schon kennengelernt. Gerade weil das die wichtigste Technik ist, möchte ich sie hier noch einmal in Zusammenhang mit den anderen beiden Möglichkeiten wiederholen.

6.1 Externe Stylesheets

Bei unserem ersten CSS-Beispiel zu Beginn dieses Studienhefts haben wir ja bereits ein **externes** Stylesheet kennengelernt. Dazu müssen folgende Voraussetzungen erfüllt sein:

- Die Stylesheet-Datei muss als Textdatei mit der Extension **.css** abgespeichert werden.
- Die Verknüpfung muss im Head-Bereich der HTML-Datei definiert werden:

```
<link rel="stylesheet" href="pfad/name.css">
```

Vielleicht ist Ihnen die Tragweite dieser Methode beim Lesen des ersten Beispiels zu Beginn dieses Studienhefts noch gar nicht so richtig bewusst geworden, weil Sie da mit so vielen neuen Konzepten konfrontiert worden sind: Eine externe Stylesheet-Datei ist deshalb die optimale Lösung, weil man mit einem **einzigen** Satz von Regeln (d. h. einer einzigen CSS-Datei) **beliebig viele** HTML-Dokumente **gleichzeitig** formatieren kann! Es muss lediglich in **allen** HTML-Dateien die Zeile `<link ... >` mit dem richtigen relativen Pfad zur CSS-Datei eingegeben werden – und schon „tanzen alle Webseiten nach einer Pfeife“. Eine Änderung von Formatierungsdetails ist damit – selbst für Hunderte von HTML-Seiten – an einer zentralen Stelle in wenigen Augenblicken erledigt!

6.2 Interne Stylesheets

Eine bestimmte HTML-Seite kann mit CSS formatiert werden, indem man die CSS-Regeln im **head**-Bereich unterbringt. Dafür existiert der spezielle HTML-Befehl **style**. Diese Form der Verknüpfung zwischen CSS und HTML wird auch als *eingebettetes Stylesheet* (engl. „embedded Stylesheet“) bezeichnet.

Alle Seiten
„tanzen nach
einer Pfeife“!

```

<!DOCTYPE HTML>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Embedded Style Sheet</title>
  <style>
    body {
      font-family: Verdana, Geneva, sans-serif;
    }
    h1 {
      color: #090;
      text-align: center;
    }
  </style>
</head>

<body>
  <h1>Eingebettete Style Sheets</h1>
  <p>Lorem Ipsum sit dolor.</p>
</body>
</html>

```

Abb. 6.1: Ein eingebettetes Stylesheet

Wie Sie in der Abbildung oben sehen können, werden die CSS-Regeln bei einem eingebetteten Stylesheet mithilfe des `style`-Tags in einer eigenen „Sandwich-Struktur“ untergebracht.

6.3 Inline Styles

Inline Styles erlauben die lokale Anwendung von Regeln mithilfe des Attributs `style` direkt im betreffenden HTML-Tag, wie das folgende Code-Beispiel zeigt:

```

<!DOCTYPE HTML>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Inline Styles</title>
</head>

<body>
  <h1 style="font-family:Verdana, Geneva, sans-serif; color:#F03;">Inline Styles</h1>
  <p style="font-family: Arial, Helvetica, sans-serif; font-size:16px; font-weight:bold;">
    Lorem Ipsum sit dolor.
  </p>
</body>
</html>

```


Abb. 6.2: Inline Styles

Diese Form ist jedoch am wenigsten flexibel und sollte möglichst vermieden werden: *Inline Styles* wirken wirklich nur genau an **einem einzigen** HTML-Element. Würde man auf diese Weise eine ganze Seite formatieren, wäre der Aufwand für eine gegebenenfalls nötige Änderung erheblich. Verwenden Sie also Inline Styles nur, um mal kurz ein Format auszuprobieren. Sobald Sie mit dem Resultat zufrieden sind, sollten Sie den Inhalt des Inline Styles in ein internes – oder noch besser in ein externes Stylesheet verlagern!

Zusammenfassung

- Es gibt externe Stylesheets, interne Stylesheets und Inline Styles.
- Die größte Effizienz erreicht man mit externen Stylesheets, weil damit eine große Zahl von HTML-Dokumenten gleichzeitig gestaltet werden kann.
- Interne Stylesheets werden auch als eingebettete Styles (embedded styles) bezeichnet.
- Inline Styles werden mithilfe des Attributs **style** direkt beim HTML-Befehl angegeben.

Aufgabe zur Selbstüberprüfung

- 6.1
- Erweitern Sie bitte Ihre HTML-Übungsdatei **css-beispiel.html** (bzw. **css-beispiel-neu.html**) aus den vorangegangenen Kapiteln: Definieren Sie dazu bitte im **head**-Bereich der Datei ein **eingebettetes Stylesheet**. Orientieren Sie sich dabei an dem Code-Beispiel aus diesem Kapitel.
 - Verlagern Sie bitte die beiden Klassen aus der Übung des vorausgegangenen Kapitels „Klassen“ in dieses interne Stylesheet. Entfernen Sie dazu den entsprechenden CSS-Code mit Ausschneiden (**Strg** + **X**) aus dem externen Stylesheet und fügen Sie diesen an der richtigen Stelle im internen Stylesheet wieder ein (**Strg** + **V**). Mac-User wissen natürlich, dass sie statt der Strg-Taste die „Propeller-Taste“  verwenden müssen.
 - Sehen Sie sich das Ergebnis im Browser an: Es sollte sich nicht verändert haben. Falls die Seite nicht so aussieht, wie Sie erwarten: Haben Sie auch daran gedacht, die Dateien nach den Änderungen zu speichern? Haben Sie die Seite in Ihrem Browser neu geladen bzw. aktualisiert?
 - Probieren Sie bitte noch an einem beliebigen Element in Ihrem HTML-Code die Anwendung eines **Inline Styles** aus. Orientieren Sie sich dabei an dem Code-Beispiel aus diesem Kapitel.

7 Vererbung und Hierarchie

Jetzt wird es langsam Zeit, dass ich Ihnen endlich erkläre, warum diese Stilvorlagen eigentlich „Cascading“ Stylesheets heißen.

7.1 Vererbung

Eine besonders wichtige Eigenschaft von CSS-Regeln besteht darin, dass sie *vererbt* (engl. „inherited“) werden können. Betrachten Sie dazu folgendes Stylesheet und das zugehörige HTML-Beispiel:

```
h1 {color: green;}
```

Der HTML-Code, für den diese Regel wirksam ist, lautet:

```
<h1>Stylesheets sind <em>toll</em></h1>
```

Obwohl für den Selektor „**em**“ keine explizite Deklaration angegeben wurde, wird das Wort „toll“ vom Browser ebenfalls in Grün dargestellt, weil es innerhalb der **h1**-Tags steht und den Stil dieses Selektors „erbt“. Die **h1**-Überschrift ist in unserem Beispiel das *Elternelement* – das **em**-Tag ist das *Kindelement*. Die CSS-Regeln werden also immer von Elternelementen auf Kindelemente vererbt (sofern es sich um eine vererbare Eigenschaft handelt).¹⁰ Das ist natürlich sehr nützlich, weil dieses Verhalten meistens genau das ist, was man als Webdesigner/-in möchte. Will man aber, dass das Wort „toll“ anders dargestellt wird, muss man dafür explizit sorgen, z. B. mit einer eigenen Regel für das **em**-Tag:

```
h1 em {color: red;}
```

In diesem Beispiel habe ich einen kontextabhängigen Selektor (Kombinator) verwendet.

Hinweis:

Ein Beispiel für eine Eigenschaft, die **nicht** vererbt wird, ist beim **body**-Tag der Rahmen (**border**). Wenn dieser vererbt werden würde, hätten **alle** Elemente einen Rahmen, was man bestimmt in den meisten Fällen nicht haben will¹¹

Zusammenfassungen des Themas Vererbung finden Sie hier:

- <https://little-boxes.de/lb1/11.6-die-vererbung-inheritance.html>
- <http://www.intensivstation.ch/css/inheritance>

Kindelemente erben immer die CSS-Regeln von ihren Elternelementen

10. Diese Vererbung gilt ganz allgemein für Nachfahren, also beispielsweise auch bei `<h1>... </h1>`.

11. Unter https://developer.mozilla.org/de/docs/Web/CSS/CSS_Reference kann man für jede Eigenschaft nachlesen, ob sie vererbt wird, oder nicht.

7.2 Die Kaskade

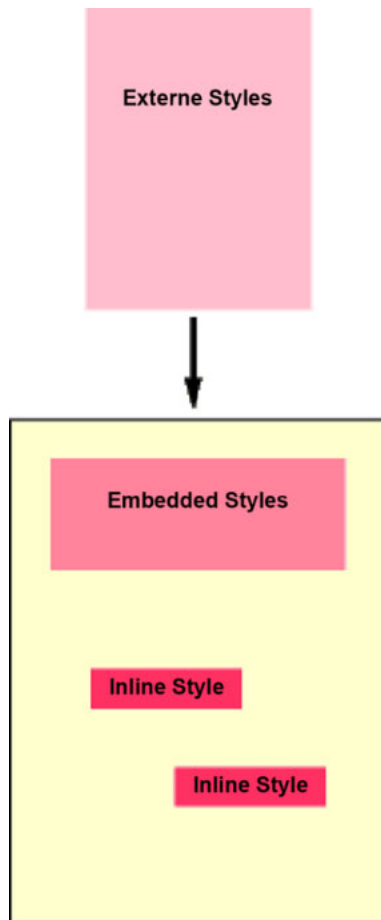


Abb. 7.1: Kaskade

Das Konzept der „Cascading“ Stylesheets erlaubt die Kombination von Stilinformationen aus mehreren Quellen. Ein realistisches Beispiel für die Website einer Firma könnte z. B. so aussehen:

- Der globale Stil der Firma wird in einer externen Stylesheet-Datei festgelegt.
- Der individuelle Stil einer Abteilung wird durch interne (eingebettete) Stilvorlagen definiert, die nur auf den Seiten dieser Abteilung wirken.
- Ein Mitarbeiter wendet auf einzelne HTML-Elemente einen lokalen Stil mithilfe von Inline Styles an.

Der Stil der Abteilung könnte übrigens auch ebenso mit einem zweiten externen Stylesheet definiert werden.

Der Stil, der dem betreffenden Element am nächsten steht, „gewinnt“

Es ist diese *hierarchische Kombination* von Stilvorlagen, die man als **Cascading Style-sheets** bezeichnet. Klar ist, dass es bei einer derartigen Kombination eigentlich zu Konflikten und Kollisionen kommen müsste. Die Stärke des CSS-Konzepts liegt gerade in seiner Fähigkeit, in diesen Fällen für eine festgelegte Hierarchie in der Reihenfolge der Stil-Anwendungen zu sorgen. „Cascading“ bedeutet in diesem Zusammenhang: Wenn mehr als eine Regel auf ein HTML-Element wirkt, regelt das CSS-Konzept ganz präzise, welche Regel gilt.

Eine solche Sammlung mehrerer Regeln für ein HTML-Element lässt sich immer als eine *Kaskade* betrachten, die von den allgemeineren Regeln zu den spezifischeren Regeln führt. Am Ende steht die am meisten spezifische Regel, die dann ausgewählt wird.

Dabei gilt generell:



- Stylesheets, die *später definiert* wurden, überschreiben solche, die *früher definiert* wurden.

Oder anders ausgedrückt:

- Der Stil, der dem betroffenen Element *am nächsten* steht, hat Vorrang.

Betrachten wir einmal die Stilvorlage für eine **h1**-Überschrift in unserem Beispiel aus der Abbildung oben: Nehmen wir an

- im externen Stylesheet stünde die Regel: `h1 {color: green;}`,
- im internen Stylesheet stünde die Regel: `h1 {color: red;}`,
- und schließlich im Inline Style: `<h1 style="color: blue;">`,

dann gilt nach dem zuvor Gesagten, dass die **h1**-Überschrift schließlich blau dargestellt wird!

Leider kommt hier folgende Komplikation hinzu:

Eine CSS-Regel wird **nur dann** von einer weiter unten notierten Regel überschrieben, wenn beide **Selektoren identisch** sind **und** das **gleiche Gewicht** haben. Die unterschiedliche Gewichtung von CSS-Selektoren wird durch die **CSS-Spezifität** beschrieben. Browser berechnen anhand eines **Punktesystems**, welcher Selektor der wichtigste ist.



Nach diesem Punktesystem gilt beispielsweise für:

- Typselektoren (**h1**, **p**, **div**, etc.) = 1 Punkt
- Klassenselektor (**.grosstext**, **.kleintext**) = 10 Punkte
- ID-Selektoren (**#content**) = 100 Punkte¹²

Ein Klassenselektor setzt sich demnach gegenüber einem Elementselektor **immer** durch – selbst dann, wenn der Elementselektor in der Kaskade **später** definiert worden ist. Da IDs im Vergleich zu Klassen ein sehr hohes Gewicht haben (100 zu 10) sollte man IDs allein deshalb schon sparsam einsetzen.

Eine Vertiefung dieses Themas würde den Rahmen dieses Studienhefts sprengen. Eine gute Übersicht zu den Themen Kaskade und Spezifität einschließlich der Erklärung des Punktesystems finden Sie in diesen deutschsprachigen Beiträgen im Netz:

- <https://kulturbanause.de/blog/css-spezifitaet/>
- <https://little-boxes.de/lb1/6.5-spezifitaet-punktesystem-fuer-selektoren.html>

Möchte man erreichen, dass eine bestimmte Eigenschaft entgegen den oben beschriebenen Regeln **unter allen Umständen** wirksam sein soll, kann man dies mit dem Zusatz **!important** in der Deklaration erreichen:

```
h1{ color:black !important; }
```

In diesem Fall werden Regeln für den Selektor **h1** ohne den Zusatz **!important** überschrieben. Diese „Holzhammermethode“ sollte allerdings nur sparsam eingesetzt werden.

12. ID-Selektoren werden Sie in einem späteren Kapitel kennenlernen.

Zusammenfassung

- CSS-Regeln werden von Elternelementen auf Kindelemente (bzw. ganz allgemein auf deren Nachkommen) vererbt.
- Regeln, die später definiert werden, haben Vorrang vor Regeln, die früher definiert werden.
- Der Stil, der dem betroffenen Element am nächsten steht, hat Vorrang.
- Eine CSS-Regel wird **nur dann** von einer weiter unten notierten Regel überschrieben, wenn beide **Selektoren identisch** sind **und** das **gleiche „Gewicht“** haben.
- Die unterschiedliche Gewichtung von CSS-Selektoren wird durch die **CSS-Spezifität** beschrieben. Browser berechnen anhand eines **Punktesystems**, welcher Selektor der wichtigste ist.

Aufgabe zur Selbstüberprüfung

- 7.1 Gibt es in Ihrer Übungsdatei **css-beispiel.html** (bzw. **css-beispiel-neu.html**) aus dem vorangegangenen Kapitel Beispiele für die Vererbung von Eigenschaften? Schauen Sie sich Ihr Dokument einmal unter diesem Gesichtspunkt an.

Testen Sie bitte, wie Sie mithilfe eines **Inline Styles** die bereits im externen Stylesheet festgelegte Formatierung eines beliebigen Elements überschreiben können.

8 Voraussetzungen für CSS-basiertes Layout

Bevor wir damit beginnen können, einer Webseite mithilfe von CSS ein Layout zu geben, müssen wir uns zuerst noch einige wichtige Voraussetzungen erarbeiten: Ich werde Ihnen zuerst das sog. „Box-Modell“ erklären, dann zwei neue HTML-Elemente beibringen und noch eine neue Form der Zuordnung von CSS-Regeln zu HTML-Elementen zeigen. Dann erhalten Sie eine Einführung in Flexbox – eine der beiden wichtigen Techniken zur Gestaltung von Layouts mit CSS. Zuletzt erfahren Sie, was es mit dem Umfließen von HTML-Elementen auf sich hat.

8.1 Das Box-Modell

Zuerst möchte ich Sie auf einen Punkt aufmerksam machen, an den Sie sich ja wahrscheinlich von Ihren HTML-Übungen selbst noch gut erinnern können: Wenn Sie HTML-Befehle wie Überschriften (z.B. **h1**), Absätze (**p**) und Listen (z.B. **ul**) verwendet haben, wurden die entsprechenden Inhalte (also in dem Fall die Texte) vom Browser automatisch jeweils in einer eigenen, neuen Zeile dargestellt. In der Terminologie von CSS heißt das: Diese HTML-Elemente besitzen standardmäßig den Darstellungsstil **block**. Oder anders ausgedrückt: Es gibt eine CSS-Eigenschaft namens **display**, die bei diesen Elementen den Wert **block** hat. Sie haben aber damals auch schon Elemente kennengelernt, bei denen der Wert der Eigenschaft **display** standardmäßig **inline** ist: Das waren die Verweise (**<a...>**), die tatsächlich einfach in den Fließtext eingebettet werden und nicht automatisch in einer eigenen, neuen Zeile stehen.

Übrigens: Man bezeichnet diese beiden Elemente-Typen oft auch nur kurz als **block**-Elemente bzw. **inline**-Elemente. Wie aber werden die Ausmaße der Elemente bestimmt? Das ist im *Boxmodell* festgelegt. Wir werden uns hier ansehen, wie dieses bei Blockelementen wirkt. Beginnen wir mit dem „klassischen“ Box-Modell, das bis zur Einführung von CSS3 ausschließlich gültig war. Hier ergab sich die *Gesamtbreite* eines solchen Elements aus der Addition folgender Einzelwerte (in Klammern steht jeweils der Name der entsprechenden CSS-Eigenschaft):

- *Breite* des Elementinhalts (**width**)
- des *Innenabstands* (**padding**)
- der *Rahmenstärke* (**border-width**)

Wenn der Abstand zum Seitenrand oder zu einem benachbarten **block**-Element relevant ist, müssen dazu auch noch die entsprechenden Werte für den *Außenabstand* (**margin**) berücksichtigt werden.

Die folgende Abbildung veranschaulicht, wie die einzelnen Abstandswerte um den eigentlichen Inhaltsbereich des **block**-Elements (d.h. den Content) herum angeordnet sind:

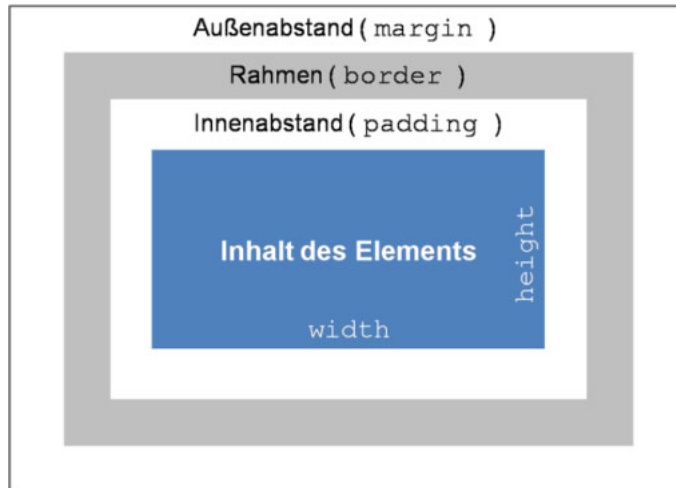


Abb. 8.1: Das Box-Modell

Das Box-Modell kann man sich wie „Zwiebelschalen“ vorstellen

Der Rahmen heißt in der CSS-Sprache **border** und besitzt mehrere einzelne Eigenschaften, von denen im Zusammenhang mit der Breite bzw. Höhe nur die Eigenschaft **border-width** relevant ist. Für alle Breiten- bzw. Höhenwerte kann man in CSS natürlich auch die entsprechenden Einzeleigenschaften für links, rechts, oben und unten definieren (z.B. **padding-left** oder **margin-bottom**).

Hier noch ein konkretes Rechenbeispiel: Wir definieren ein Element mit einer Breite (**width**) von 200 Pixeln, einer Höhe (**height**) von 100 Pixeln und einem Innenabstand (**padding**) und Rahmen (**border**) von je 20 Pixeln Stärke auf allen vier Seiten.

Die tatsächliche Breite beträgt dann 280 Pixel:

- 200 für **width**
- 20 für den Beitrag von **border-width** auf der linken Seite
- 20 für den Beitrag von **padding** auf der linken Seite
- 20 für den Beitrag **padding** auf der rechten Seite
- 20 für den Beitrag von **border-width** auf der rechten Seite¹³

Bisher haben wir uns ja nie weiter um die Breite oder Höhe von Überschriften bzw. Absätzen gekümmert: Die sichtbare Breite hat sich einfach aus der Textmenge ergeben. Die Höhe folgt ebenfalls aus der Textmenge in Verbindung mit der Größe der vom Browser verwendeten Standardschrift. Durch die mit dem Box-Modell eingeführten zusätzlichen CSS-Eigenschaften werden Breite und Höhe zu ziemlich kompliziert zusammengesetzten Größen. Andererseits gewinnt man als Webdesigner dadurch große Flexibilität bei der Gestaltung der Seiten. Die Berechnung der Gesamthöhe eines Elements folgt grundsätzlich denselben Regeln, wie sie bei der Gesamtbreite erklärt worden sind. In der Regel wird jedoch **nur die Breite** eines Elements via CSS definiert. Die Höhe soll sich in den meisten Fällen aus der Menge des Element-Inhalts automatisch ergeben. Wie

13. Im Zusammenhang mit Rahmen ist auch die Eigenschaft **border-style** relevant, mit der man den Rahmentyp festlegen kann (https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/%C3%A4u%C3%9Fere_Gestaltung/Rahmen/border-style).

Sie allerdings im Abschnitt über flexible Maßeinheiten erfahren haben, kann beim Responsive Webdesign mit der Einheit `vh` die Höhe eines Elements auf die aktuelle Höhe des Viewports eingestellt werden.

Das klassische Box-Modell gilt zu recht als wenig intuitiv: Im täglichen Leben würde man die Breite (**width**) einer Box (Kiste) sicher so bestimmen, dass man die Entfernung von einem Außenrand zum anderen inklusive Innenabstand (**padding**) und Rand (**border**) misst. Beim klassischen Box-Modell definiert **width**, wie wir gesehen haben, nur die Breite des Inhaltsbereichs, während die Werte für **padding** oder **border** dazu kommen.

Seit Einführung von CSS3 gibt es eine Alternative zum klassischen Box-Modell. Das wird übrigens als *Content-Box* bezeichnet, weil die Breite nur den **Inhaltsbereich** umfasst. Die Alternative heißt *Border-Box*, weil hier die Breite (**width**) einschließlich **padding** und **border** gemessen wird. Oder anders gesagt: **padding** und **border** wirken bei Border-Box nach innen. Aktiviert wird das alternative Box-Modell ganz einfach, indem dieses z. B. über folgende CSS-Anweisung mit dem Universalselektor `*` auf **alle** Elemente angewendet wird:

```
* { box-sizing: border-box; }
```

Eine gute Übersicht zum Thema Box-Modell finden Sie auf dieser deutschsprachigen Seite im Netz:

- <https://little-boxes.de/lb1/7.2-das-box-modell-in-der-uebersicht.html>

8.2 HTML-Erweiterung: die Elemente DIV und SPAN

Eigentlich ist das hier ja ein Studienheft zum Thema CSS. Trotzdem werde ich Ihnen hier kurz noch zwei neue HTML-Elemente beibringen. Natürlich hätte ich das auch gleich im ersten Heft machen können – aber damals hatten Sie ja schließlich schon genug neuen Stoff zu lernen. Außerdem haben diese beiden neuen Tags ohne den Zusammenhang mit CSS auch gar keinen Sinn und schwierig zu verstehen sind sie sowieso nicht. Also los:

Neben den Elementen zur Strukturierung wie **nav**, **main** etc. gibt es auch das **div**-Element, das man verwendet, wenn keines der spezielleren Elemente passt. Mit dem **div**-Element kann man innerhalb des HTML-Codes einen Behälter definieren. Solche Behälter werden auch oft als *Container* bezeichnet. Hier folgt gleich ein Beispiel:

```

<!DOCTYPE HTML>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>DIV-Container</title>
  </head>
  <body>
    <div>
      <p>Lorem ipsum dolor sit amet.</p>
      <ul>
        <li>Consetetur sadipscing elitr</li>
        <li>Sed diam nonumy</li>
      </ul>
      <p>At vero eos et accusam et justo</p>
    </div>
  </body>
</html>

```

Abb. 8.2: Ein HTML-Container mit dem div-Element

Das **div**-Element schafft also einfach noch eine zusätzliche Ebene in der hierarchischen Sandwich-Struktur von HTML.

Neben dem **div**-Element, das ein generisches *Block*-Element ist, existiert auch ein generisches *Inline*-Element, nämlich **span**..

```

<!DOCTYPE HTML>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <title>SPAN</title>
  </head>
  <body>
    <p>Lorem ipsum dolor sit amet, consetetur sadipscing elitr,
    <span class="blautext">sed diam nonumy</span> eirmod tempor
    invidunt ut labore et dolore magna aliquyam erat,
    sed diam voluptua.</p>
  </body>
</html>

```

Abb. 8.3: Beispiel für die Anwendung des span-Tags

Innerhalb eines normalen Absatzes wird also ein Teil des Textes mit dem **span**-Element in eine zusätzliche Sandwich-Struktur „eingepackt“. Ich habe das Code-Beispiel so formuliert, wie es typischerweise in der Praxis vorkommt: Das **span**-Tag hat nur einen einzigen Sinn – es schafft im HTML-Code einfach nur ein Element, auf das man eine CSS-Klasse anwenden kann. Ich habe hier zur Formatierung wieder den Klassennamen ohne Typselektor **.blautext** aus dem Kapitel über Klassen verwendet:

```
.blautext {color:blue;}
```

Der Zweck dieser zusätzlichen Verschachtelung ist Ihnen natürlich schon längst klar geworden: Man möchte in diesem Fall, dass **nur** der Textteil „sed diam nonumy“ blau dargestellt wird und **nicht** der gesamte Absatz, denn sonst hätte man ja die Klasse **.blautext** dem **p**-Tag als Attribut „angeklebt“. Der Textteil „sed diam nonumy“ läuft also ohne neue Zeile normal im Fließtext des Absatzes mit, ist aber eben blau statt schwarz.

Die „Blauen“
sind mitten
unter den
„Schwarzen“

8.3 ID-Selektoren

Sie können `div`-Elemente zum Formatieren mit einer Klasse versehen.

```
<div class="farbtext"> ... </div>
```

So könnte man sogar mehrere Container mit unterschiedlichen Klassen formatieren. Bei unserem `div`-Container ist so viel Flexibilität aber gar nicht notwendig. Denn Container werden typischerweise nur zu **einem** ganz bestimmten Zweck eingesetzt, den Sie im nächsten Kapitel noch genau kennenlernen werden: Sie sind der Behälter für alle anderen Elemente und bilden so die oberste Hierarchieebene für das Layout einer Webseite! Aus diesem Grund werden diese Layoutcontainer meistens auch als „holder“ oder „wrapper“ (engl. to wrap = einpacken) bezeichnet. Es liegt in der Natur eines solchen Behälters, dass es davon immer nur **einen** pro Seite gibt. Deshalb kann man in diesem Fall die CSS-Zuweisung mit einem ID-Selektor vornehmen. Die `id`-Attribute haben wir ja im vorigen Studienheft bereits im Zusammenhang mit Formularen kennengelernt. Hier brauchen wir sie also wieder; und da kommt auch schon ein Beispiel:

Der ID-Selektor mit der Bezeichnung `dieblauen` wird im CSS-Code so angegeben:

```
#dieblauen {color:blue;}
```

Der zugehörige HTML-Code lautet dann:

```
<div id="dieblauen"> ... </div>
```

Die Zuweisung der Stilvorlage erfolgt in diesem Fall also über diese eindeutige ID. Hier darf es **keinen** weiteren Container mit der ID „`dieblauen`“ geben, weil sonst die Zuordnung nicht mehr eindeutig wäre!

Hinweis:

Derzeit gibt es im Webdesign einen eindeutigen Trend: Man vermeidet IDs und arbeitet ausschließlich mit Klassen. Der Grund ist die bereits erwähnte große Flexibilität bei der Anwendung von Klassen. Deswegen wird im folgenden Beispiel auch nur eine einzige ID genutzt.

Ein Beispiel ist das beliebte Frontend-CSS-Framework *Bootstrap*. Es enthält auf HTML und CSS basierende Gestaltungsvorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-Systeme, Navigations- und andere Oberflächengestaltungselemente und arbeitet ausschließlich mit *Klassen*.

8.4 Flexbox

Flexbox ist das einfachere von zwei Modulen, die die CSS-Technologie für das Layout von Webseiten bereitstellt. Während sich *Flexbox* eher für lineare Strukturen eignet, ermöglicht das weitaus komplexere *CSS Grid* die Implementierung verschachtelter Layouts auf der Grundlage echter Gestaltungsraster. Für unser Beispiel-Layout ist *Flexbox* vollkommen ausreichend. Bevor wir uns also im nächsten Kapitel einem realistischen Layout zuwenden, werde ich Ihnen zuerst eine kurze *Flexbox*-Einführung anhand eines ganz simplen Beispiels geben. Sehen wir uns dazu folgenden Code an, der aus einem eingebetteten Stylesheet und dem zugehörigen HTML-Markup besteht:

```

1  <!DOCTYPE HTML>
2  <html lang="de">
3  <head>
4    <meta charset="utf-8">
5    <title>Meine tolle Firma - Layoutvorstufe</title>
6  </head>
7  <style>
8    body {
9      background: #ADA189;
10   }
11   #holder {
12     max-width: 60rem;
13     background: #FFF;
14     margin: 0 auto;
15     display: flex;
16   }
17   header {
18     background: #DCCBAC;
19   }
20   nav {
21     background: Pink;
22   }
23   main {
24     background: DarkSeaGreen;
25   }
26   footer {
27     background: PaleTurquoise;
28   }
29 </style>
30 </head>
31 <body>
32   <div id="holder">
33     <header>
34       header
35     </header>
36     <nav>
37       nav
38     </nav>
39     <main>
40       main
41     </main>
42     <footer>
43       footer
44     </footer>
45   </div>
46 </body>
47 </html>

```

Abb. 8.4: Flexbox: ein simples Beispiel

In diesem Code gibt es nur eine Besonderheit, die Ihnen auffallen sollte, weil Sie sie bisher noch nicht kennen: **#holder** wurde die Eigenschaft **display: flex;** zugewiesen. Das reicht schon aus, um die Darstellung der Elemente im Markup gravierend zu verändern. Sehen wir uns das im Browser an:¹⁴

14. Damit Sie dabei die einzelnen Elemente besser erkennen und unterscheiden können, wurden ihnen verschiedene Hintergrundfarben zugewiesen.

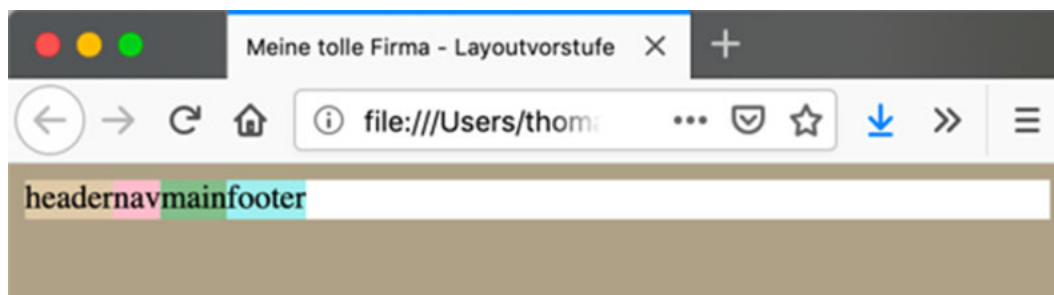


Abb. 8.5: Das Flexbox-Beispiel im Browser

Sie werden sofort bemerken, dass nun die Elemente **nebeneinander** angeordnet sind, statt wie sonst – in der Reihenfolge ihrer Definition im HTML-Markup – **untereinander**. Durch `display: flex;` werden nämlich alle im Layoutcontainer **holder** enthaltenen Elemente spaltenweise positioniert.

Und nun zur Theorie, die ich Ihnen kurz und knapp in folgenden Stichpunkten erklären möchte:

- Bei der Flexbox-Technologie muss man zwei wichtige Begriffe unterscheiden: *Flexcontainer* und *Flexitems*
- Ein Element muss der Flexcontainer sein: das ist das umfassende Element der flexibel angeordneten Elemente. Die Kindelemente des Flexcontainers werden angeordnet und heißen Flexitems. Die Anordnung kann untereinander und nebeneinander (Standard) erfolgen.
- Wir machen bei diesem Beispiel unseren umfassenden Layoutcontainer **holder** durch `display: flex;` zum Flexcontainer. Die darin befindlichen Elemente – hier also **header**, **nav**, **main**, **footer** – werden damit zu Flexitems.

Mit Flexbox können auf unkomplizierte Weise mehrspaltige Layouts erstellt werden

Wir werden nun dieses einfache Beispiel Schritt für Schritt mithilfe weiterer Elemente der Flexbox-Technik verfeinern: Die Elemente **header**, **nav**, **main**, **footer** nehmen im **holder** in der Horizontale gerade so viel Raum ein, wie für den jeweils enthaltenen Text (also „header“, „nav“, „main“, „footer“) nötig ist. Der Rest von **holder** bleibt leer. Das wollen wir nun ändern. Dazu wird dem main-Element die Eigenschaft `flex: 1;` zugewiesen:¹⁵

15. Ab jetzt sehen wir uns nur noch die Änderungen im CSS-Code des eingebetteten Stylesheets an.

```
#holder {
  max-width: 60rem;
  background: #FFF;
  margin: 0 auto;
  display: flex;
}
header {
  background: #DCCBAC;
}
nav {
  background: Pink;
}
main {
  background: DarkSeaGreen;
  flex: 1;
}
footer {
  background: PaleTurquoise;
}
```

Abb. 8.6: Flexbox: die Eigenschaft flex

- Mit `flex: 1;` erhält `main` den gesamten neben `nav` verfügbaren Platz. Wenn mehrere Elemente `flex: 1;` erhalten, wird der verfügbare Platz gleichmäßig verteilt.

Und so sieht das dann im Browser aus:

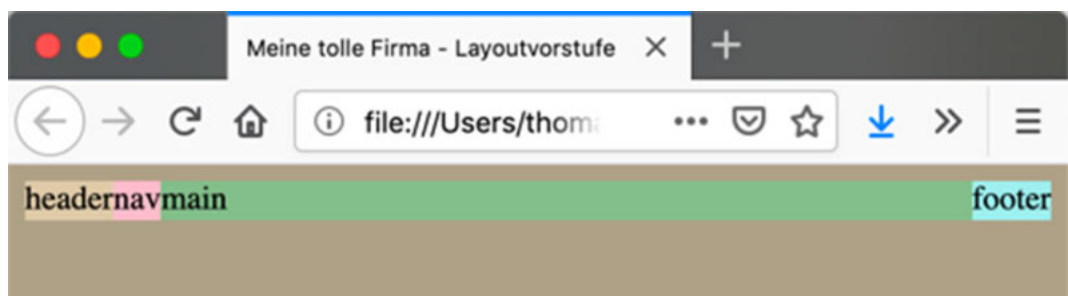


Abb. 8.7: Flexbox: main erhält mehr Platz

```
#holder {
  max-width: 60rem;
  background: #FFF;
  margin: 0 auto;
  display: flex;
}
header {
  background: #DCCBAC;
}
nav {
  background: Pink;
}
main {
  background: DarkSeaGreen;
}
footer {
  order: -1;
  background: PaleTurquoise;
}
```

Abb. 8.8: Flexbox: die Eigenschaft order

- Standardmäßig werden die Flexitems in der Reihenfolge angezeigt, in der sie im HTML-Code notiert wurden.
- Mit der Eigenschaft **order** kann diese Reihenfolge geändert werden, ohne dass der HTML-Code angepasst werden muss.
- Der Standardwert für **order** ist 0. Ein größerer positiver Wert verschiebt das entsprechende Element ganz nach hinten.
- Möchte man, dass ein Element zuerst angezeigt wird und die Reihenfolge aller anderen Artikel unverändert bleibt, kann man diesem **order: -1;** zuweisen. Da dieser Wert niedriger als 0 ist, wird dieses Element immer zuerst angezeigt.

Das haben wir im Code oben beim **footer** einmal ausprobiert. Und so sieht das dann im Browser aus:

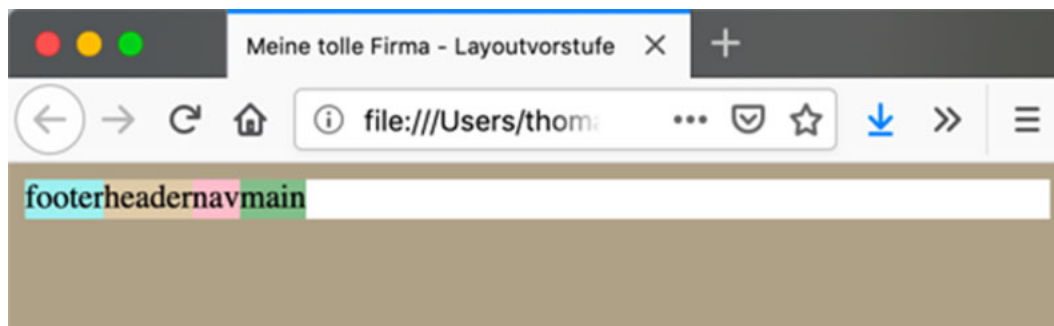


Abb. 8.9: Flexbox: der footer erscheint zuerst

Im nächsten Schritt wollen wir die Breiten unserer Elemente nicht mehr einfach dem darin enthaltenen Inhalt überlassen, sondern ihnen bestimmte Breiten zuweisen. Die Eigenschaft **order: -1;** wird beim **footer** jetzt wieder entfernt:

```
#holder {
  max-width: 60rem;
  background: #FFF;
  margin: 0 auto;
  display: flex;
}
header {
  background: #DCCBAC;
  width: 100%;
}
nav {
  background: Pink;
  width: 30%;
}
main {
  background: DarkSeaGreen;
  flex: 1;
}
footer {
  background: PaleTurquoise;
  width: 100%;
}
```

Abb. 8.10: Flexbox: die Elemente bekommen bestimmte Breiten

Dieser Schritt hat nicht unmittelbar etwas mit der Flexbox-Technik zu tun, wirkt sich aber natürlich auf unser Layout aus, wie der folgende Screenshot der Browserdarstellung zeigt:

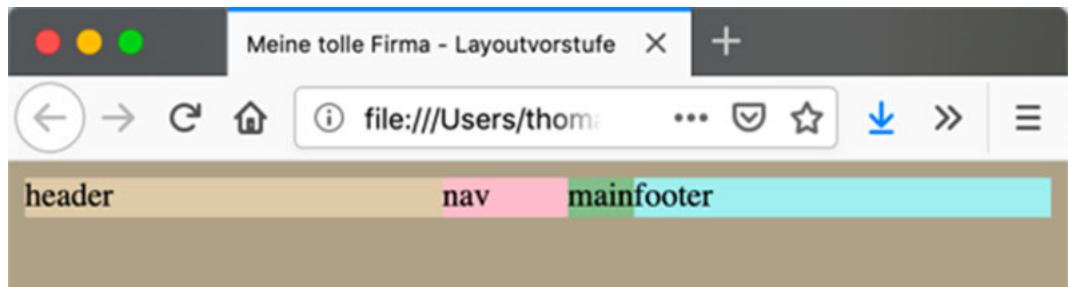


Abb. 8.11: Flexbox: Die Elemente haben definierte Breiten

Nun wird Ihnen sicher auffallen, dass das Layout noch nicht ganz so aussieht, wie wir uns das wünschen: Den Elementen **header** und **footer** haben wir eine Breite von 100 % zugewiesen, d.h., diese beiden sollten eigentlich die gesamte Breite ihres Elternelements **holder** einnehmen. Damit sie das auch tun, müssen wir unserem Layoutcontainer **holder** (er ist ja unser Flexcontainer) neben **display: flex;** eine weitere Eigenschaft zuweisen:

```
#holder {
  max-width: 60rem;
  background: #FFF;
  margin: 0 auto;
  display: flex;
  flex-wrap: wrap;
}
header {
  background: #DCCBAC;
  width: 100%;
}
nav {
  background: Pink;
  width: 30%;
}
main {
  background: DarkSeaGreen;
  flex: 1;
}
footer {
  background: PaleTurquoise;
  width: 100%;
}
```

Abb. 8.12: Flexbox: die Eigenschaft flex-wrap

- Normalerweise werden die Flexitems innerhalb der Flexbox **linear nebeneinander** positioniert.
- Wenn mehrere Items von ihren Abmessung her nicht in den Container passen, kann man mit der **Eigenschaft flex-wrap** für den Layoutcontainer **holder** festlegen, dass diese Flexitems in mehrere Spalten und Zeilen umbrechen.
- Die Deklaration **flex-wrap: wrap;** sorgt für eine mehrzeilige Darstellung innerhalb des Layoutcontainers **holder**.

Im Browser sehen wir sofort, dass dieser Schritt die gewünschte Wirkung hat:

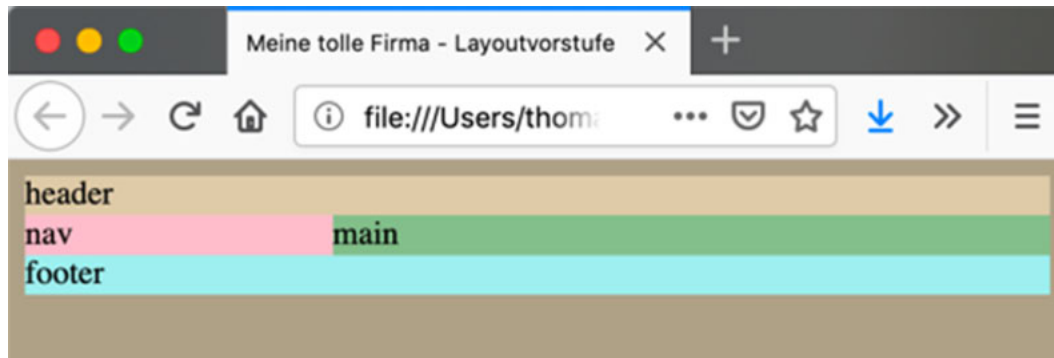


Abb. 8.13: Flexbox: header und footer bekommen eine eigene Zeile

Jetzt können wir unser kleines Flexbox-Beispiel abschließen. Wir entfernen die Hintergrundfarben, die nur der Veranschaulichung dienten und machen noch etwas „Kosmetik“ mit Rahmenlinien links von **main** und oberhalb von **footer**:

```
#holder {
  max-width: 60rem;
  background: #FFF;
  margin: 0 auto;
  display: flex;
  flex-wrap: wrap;
}
header {
  background: #DCCBAC;
  width: 100%;
}
nav {
  width: 30%;
}
main {
  flex: 1;
  border-left: 0.1rem solid #877D6C;
}
footer {
  width: 100%;
  border-top: 0.1rem solid #877D6C;
}
```

Abb. 8.14: Flexbox: CSS-Code für das fertige Beispiel

Und so sieht unser kleines Layout-Beispiel dann im Browser aus:

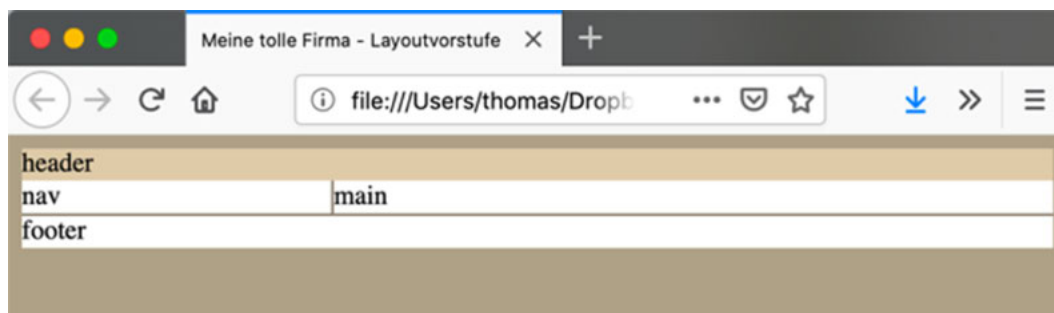


Abb. 8.15: Flexbox: das fertige Beispiel im Browser

Weitere Informationen zu Flexbox finden Sie hier:

- <https://wiki.selfhtml.org/wiki/CSS/Eigenschaften/Flexbox>
- <https://kulturbanause.de/blog/einfuehrung-in-das-flexbox-modell-von-css/>
- https://developer.mozilla.org/de/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
- <https://www.mediaevent.de/css/display-flex.html>

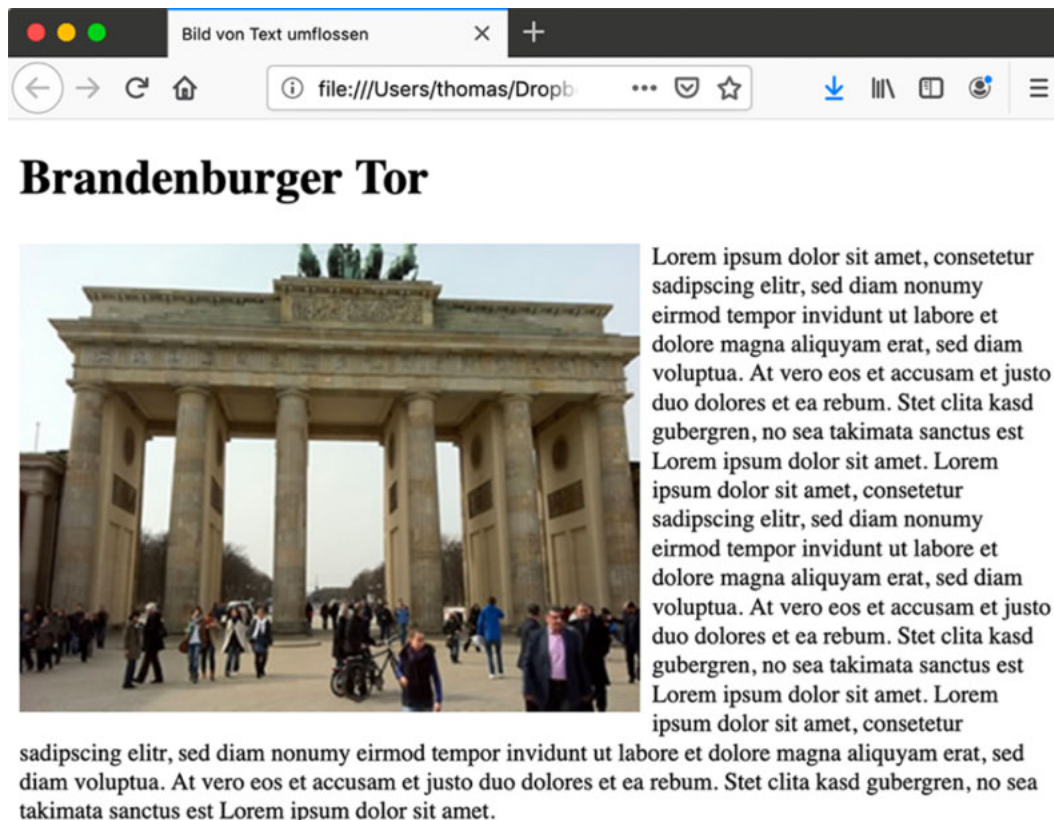
8.5 Float

Es kommt bei anspruchsvolleren Web-Layouts häufig vor, dass man Texte neben und um Bilder herumfließen lassen möchte, so wie das bei Print-Layouts auch gang und gäbe ist. Das kann mit CSS ganz einfach mithilfe der Eigenschaft **float** erreicht werden, wie das folgende Codebeispiel zeigt:

```
<!DOCTYPE HTML>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Bild von Text umflossen</title>
  <style>
    .imgleft {
      float: left;
      margin-right: 0.5rem;
    }
  </style>
</head>
<body>
  <h1>Brandenburger Tor</h1>
  <p>Lorem ipsum dolor sit
amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna
aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet,
consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna
aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet,
consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna
aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita
kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.</p>
</body>
</html>
```

Abb. 8.16: Float: Beispielcode

So sieht dieses Beispiel im Browser aus:



Mit float fließt der Text rechts an dem Bild entlang

Abb. 8.17: Float: der Text umfließt das Bild

- Wie der Name schon sagt, sorgt die Eigenschaft **float** (vom Englischen to float = schwimmen) dafür, dass ein damit gekennzeichnetes Element von allen anderen umflossen wird. Wir müssen lediglich noch den Wert dieser Eigenschaft festlegen, d. h., auf welcher Seite dieses Umfließen passiert. Unser Bild befindet sich ja schon auf der richtigen Seite. Der Rest soll also darum herumfließen, d. h., in unserem Fall lautet die richtige Deklaration **float: left;**
- Das bedeutet im Klartext: Das Element, dem die Klasse **.imgleft** zugewiesen wurde, steht **links** und wird folglich **rechts** umflossen.

Weitere Informationen zu float finden Sie hier:

- https://wiki.selfhtml.org/wiki/CSS/Tutorials/Ausrichtung/float_und_clear#float
- <https://www.mediaevent.de/css/position-float.html>
- <https://kulturbanause.de/blog/die-css-eigenschaft-float-verstehen-und-anwenden/>
- <https://developer.mozilla.org/de/docs/Web/CSS/float>

Zusammenfassung

- Man unterscheidet **block**- und **inline**-Elemente.
- Das Box-Modell definiert, wie bei HTML-Elementen Außenabstand (**margin**), Rahmen (**border**) und Innenabstand (**padding**) angeordnet sind und wie sich deren Werte zur Gesamtbreite bzw. Gesamthöhe des Elements addieren.
- Mit dem **div**-Tag werden Container für andere HTML-Elemente gebildet.
- **div**-Container sind **block**-Elemente.
- Mit dem **span**-Tag werden Bereiche innerhalb eines Fließtextes definiert, denen eine CSS-Klasse zugewiesen werden kann.
- **span**-Bereiche sind **inline**-Elemente.
- ID-Selektoren sind bestimmten **div**-Containern (oder anderen HTML-Elementen) zugeordnet, die im HTML-Dokument nur einmal vorkommen dürfen.
- Der Trend geht im Webdesign zur ausschließlichen Verwendung von Klassen.
- Bei der Flexbox-Technologie unterscheidet man Flexcontainer und Flexitems.
- Der Flexcontainer hat die Eigenschaft **display: flex;** und ist das umfassende Element der flexibel angeordneten Elemente.
- Die Kindelemente des Flexcontainers heißen Flexitems.
- Die Anordnung kann untereinander und nebeneinander (Standard) erfolgen.
- Standardmäßig werden die Flexitems in der Reihenfolge angezeigt, in der sie im HTML-Code notiert wurden.
- Mit der Eigenschaft **order** kann diese Reihenfolge geändert werden, ohne dass der HTML-Code angepasst werden muss.
- Die Deklaration **flex-wrap: wrap;** sorgt für eine mehrzeilige Darstellung innerhalb des Layoutcontainers.
- Mithilfe der Eigenschaft **float** kann man Texte neben und um Bilder herumfließen lassen.
- Das Element mit der Eigenschaft **float: left;** steht links und wird rechts umflossen.

Aufgaben zur Selbstüberprüfung

- 8.1 • Bitte vollziehen Sie die beiden in diesem Kapitel gezeigten Beispiele mit **div** bzw. **span** selbst praktisch nach.
- 8.2 • Legen Sie bitte wie gewohnt durch Kopieren einer Vorlage ein neues HTML-Dokument mit dem Dateinamen **aufgabe-div-id.html** an.
- Fügen Sie darin bitte zwei **div**-Container mit den IDs **box1** und **box2** ein.
 - In jedem dieser Container soll je ein Absatz mit dem Inhalt „Text in box1“ bzw. „Text in box2“ enthalten sein.
 - Formatieren Sie bitte diese beiden Container mithilfe eines internen Stylesheets, in dem folgende ID-Selektoren mit den zugehörigen Regeln stehen:
- ```
#box1 {
 background-color: #6CC;
 width: 300px;
 padding: 20px;
}
#box2 {
 background-color: #F66;
 width: 150px;
 margin: 20px;
 border: 4px solid #06C;
}
```
- Sehen Sie sich dann das Ergebnis im Browser an und vergleichen Sie bitte mit den Stilvorgaben für **#box1** und **#box2**.
  - Um den Unterschied zwischen Content-Box und Border-Box einmal zu sehen modifizieren Sie bitte nun dieses Beispiel in folgender Weise: Verwenden Sie bis auf die Hintergrundfarben **identische** Formatierungen für **#box1** und **#box2**. Bei **#box2** ergänzen Sie bitte zusätzlich die Angabe **box-sizing: border-box;**
  - Sehen Sie sich dann das Ergebnis im Browser an.
- 8.3 • Bitte vollziehen Sie das in diesem Kapitel gezeigte einfache Flexbox-Beispiel selbst praktisch nach. Sie können dazu von der Datei mit der Musterlösung **8\_flexbox-step\_by\_step.html** ausgehen, in der alle Flexbox-Eigenschaften bereits als auskommentierte Zeilen im CSS-Code vorhanden sind. Die Datei befindet sich im Ordner **MMDE2J Lösungen Aufgaben zur Selbstüberprüfung**. Dieser liegt im Ordner **Downloads**, den Sie im heft-bezogenen Downloadbereich Ihrer Lernplattform finden.
- 8.4 • Bitte vollziehen Sie das in diesem Kapitel gezeigte einfache Float-Beispiel selbst praktisch nach. Legen Sie dazu eine neue HTML-Datei mit einem eingebetteten Stylesheet an.

## 9 Ein Beispiel-Layout

*Jetzt wird es endlich „ernst“: Ich werde Ihnen anhand eines Beispiels Schritt für Schritt zeigen, wie man mithilfe von CSS das Layout für eine Webseite erstellt. Das Beispiel ist einerseits noch so simpel, dass ich es im Rahmen dieses Studienhefts ausführlich erklären kann. Es ist aber andererseits auch komplex genug, Ihnen daran viele wichtige CSS-Konzepte demonstrieren zu können.*

Im letzten Kapitel des vorausgegangenen Studienhefts zu HTML hatte ich Ihnen ein kleines, aber realistisches Praxisbeispiel mit dem Titel „Meine tolle Firma“ präsentiert. Erinnern Sie sich noch? Ich hatte Ihnen damals angekündigt, dass ich diese Beispielseite als Grundlage nehmen werde, um Ihnen später eine einfache Variante des CSS-basierten Layouts zu erklären. Jetzt ist es so weit!

### 9.1 Die Beispielseite mit Layoutcontainer

Das Beispiel wird mit einem div-Container erweitert

Haben Sie Lust, das alte HTML-Studienheft wieder zur Hand zu nehmen und sich den Quellcode des Beispiels im letzten Kapitel noch einmal anzusehen? Vergleichen Sie den Quellcode des Praxisbeispiels „Meine tolle Firma“ von damals mit dem Code der folgenden Abbildung. Ein paar kleine Änderungen bzw. Ergänzungen habe ich nämlich zur Vorbereitung der Layoutentwicklung und weiteren Gestaltung an dem Code vorgenommen:

- Es gibt jetzt im **head**-Bereich den Link zum externen Stylesheet **demo.css**.
- Im **body**-Bereich wurde der im vorausgegangenen Kapitel eingeführte Layout-Container als **div**-Element der **ID="holder"** erstellt, in dem nun alle bisherigen strukturellen HTML-Elemente „eingepackt“ sind:
  - der **header** für den Kopfbereich mit der Überschrift (an dieser Stelle könnte auch ein Logo platziert werden)
  - damit die Textmenge etwas realistischer ist, habe ich weitere Absätze mit Blindtext und einige Zwischenüberschriften eingefügt
  - das **nav**-Element für die Hauptnavigation (darin die **ul**-Liste mit den Links<sup>16</sup>)
  - das **main**-Element, das den eigentlichen Inhalt umschließt
  - **footer** für eine Fußzeile mit der Copyright-Angabe
  - innerhalb von **main** befinden sich zwei **article** und ein **aside**-Element
  - das erste **article**-Element wird durch zwei **section**-Elemente unterteilt
  - in der zweiten **section** gibt es nun einen Absatz, in dem ein Bild eingebunden ist

16. Ich möchte an dieser Stelle noch mal daran erinnern, dass ich aus Bequemlichkeit statt echter relativer Pfade mit dem **#**-Zeichen als Platzhalter bei den **href**-Attributen arbeite.

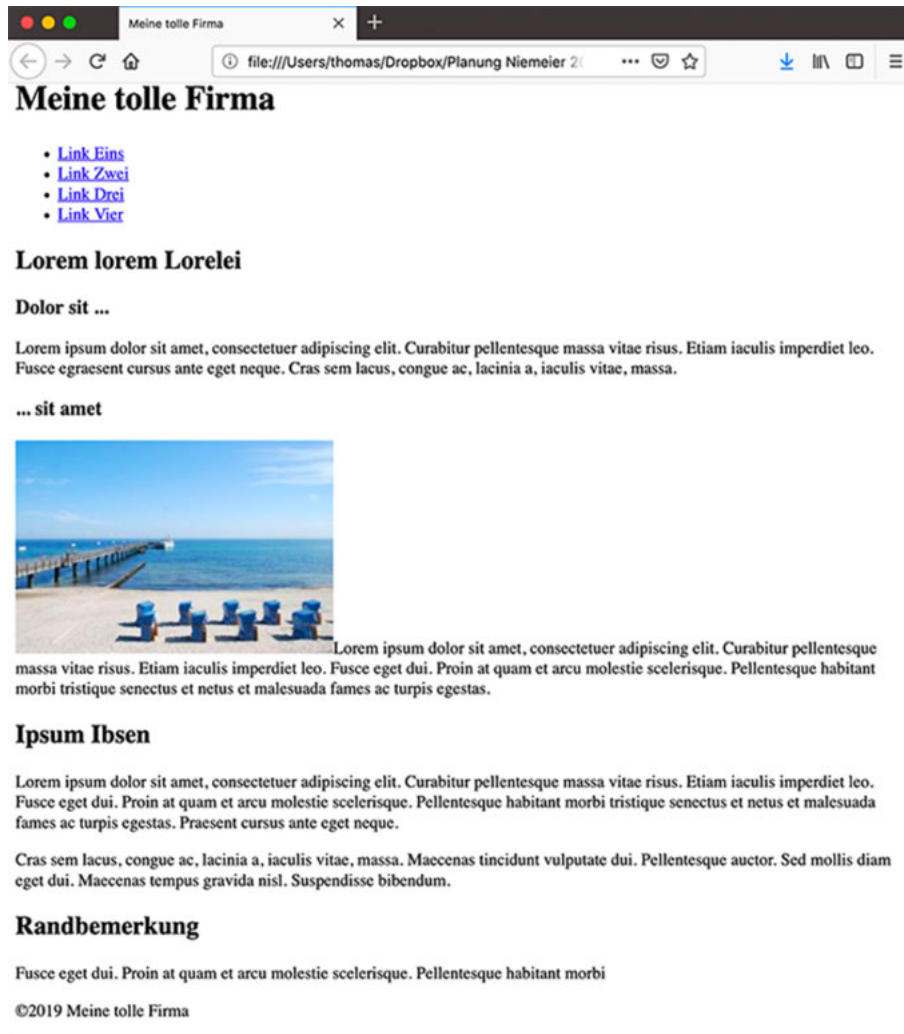
```

1 <!DOCTYPE HTML>
2 <html lang="de">
3 <head>
4 <meta charset="utf-8">
5 <title>Meine tolle Firma</title>
6 <link href="demo.css" rel="stylesheet">
7 </head>
8 <body>
9 <div id="holder">
10 <header>
11 <h1 class="callout">Meine tolle Firma </h1>
12 </header>
13 <nav>
14
15 Link Eins
16 Link Zwei
17 Link Drei
18 Link Vier
19
20 </nav>
21 <main>
22 <article>
23 <h2>Lorem lorem Lorelei</h2>
24 <section>
25 <h3>Dolor sit ...</h3>
26 <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
27 Curabitur pellentesque massa vitae risus. Etiam iaculis
28 imperdiet leo. Fusce egrasent cursus ante eget neque. Cras
29 sem lacus, congue ac, lacinia a, iaculis vitae, massa.
30 </p>
31 </section>
32 <section>
33 <h3>... sit amet</h3>
34 <p>Lorem ipsum
35 dolor sit amet, consectetur adipiscing elit. Curabitur
36 pellentesque massa vitae risus. Etiam iaculis imperdiet leo.
37 Fusce eget dui. Proin at quam et arcu molestie scelerisque.
38 Pellentesque habitant morbi tristique senectus et netus et
39 malesuada fames ac turpis egestas. </p>
40 </section>
41 </article>
42 <article>
43 <h2>Ipsum Ibsen</h2>
44 <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
45 Curabitur pellentesque massa vitae risus. Etiam iaculis
46 imperdiet leo. Fusce eget dui. Proin at quam et arcu molestie
47 scelerisque. Pellentesque habitant morbi tristique senectus et
48 netus et malesuada fames ac turpis egestas. Praesent cursus
49 ante eget neque. </p>
50 <p>Cras sem lacus, congue ac, lacinia a, iaculis vitae, massa.
51 Maecenas tincidunt vulputate dui. Pellentesque auctor. Sed
52 mollis diam eget dui. Maecenas tempus gravida nisl.
53 Suspendisse bibendum.</p>
54 </article>
55 <aside>
56 <h2>Randbemerkung</h2>
57 <p>Fusce eget dui. Proin at quam et arcu molestie scelerisque.
58 Pellentesque habitant morbi</p>
59 </aside>
60 </main>
61 <footer>
62 <p>©2019 Meine tolle Firma</p>
63 </footer>
64 </div>
65 </body>
66 </html>

```

**Abb. 9.1:** Der leicht modifizierte HTML-Code für die Beispielseite

Das externe Stylesheet **demo.css** ist bereits mit der HTML-Datei verknüpft – aber noch ganz leer, sodass diese Seite im Browser trotz des zusätzlich eingefügten Layoutcontainers immer noch sehr ähnlich aussieht wie damals im HTML-Studienheft (natürlich bis auf die zusätzlichen Absätze, Zwischenüberschriften und das Bild):

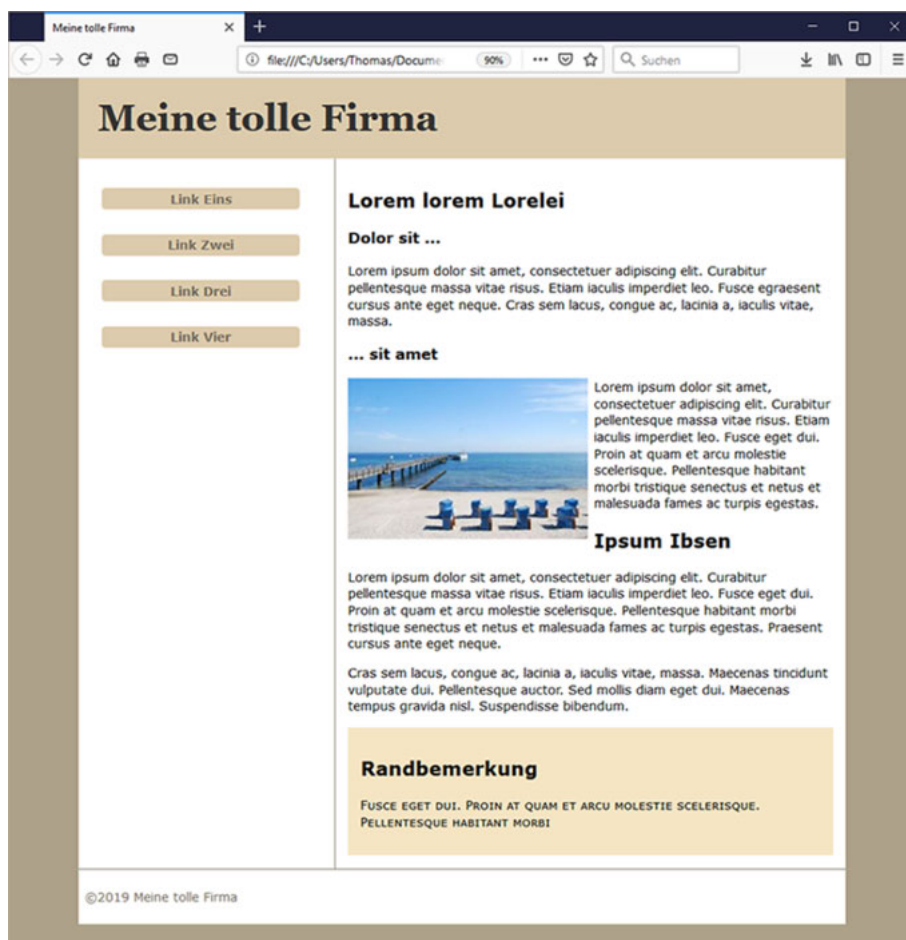


Ohne CSS wirkt sich der Einbau des Layoutcontainers nicht auf das Erscheinungsbild der Seite aus!

**Abb. 9.2:** Die Beispielseite im Browser (mit div-Container, ohne CSS)

Erst wenn die CSS-Regeln mithilfe von Klassen bzw. der automatischen Zuweisung über einen ID-Selektor auf den Layoutcontainer „holder“ und dessen untergeordneten Elemente einwirken, ändert sich das Bild ganz erheblich.

Die folgende Abbildung zeigt unsere Beispielseite, nachdem in das externe Stylesheet **demo.css** eine große Zahl von Stilvorlagen eingefügt worden ist:



**Abb. 9.3:** Das fertige CSS-Layout im Browser

Um zu verstehen, wie dieses Bild zustande kommt, müssen wir jetzt gemeinsam den Inhalt von **demo.css** analysieren:

```

1 body {
2 font: 100%/1.3 Verdana, Arial, Helvetica, sans-serif;
3 color: #000;
4 background: #ADA189;
5 margin: 0;
6 }
7 #holder {
8 max-width: 60rem;
9 background: #FFF;
10 margin: 0 auto;
11 display: flex;
12 flex-wrap: wrap;
13 }
14 header {
15 background: #DCCBAC;
16 width: 100%;
17 }
18 nav {
19 text-align: center;
20 width: 30%;
21 padding: 1rem;
22 }
23
24 main {
25 flex: 1;
26 padding: 1rem;
27 border-left: 0.1rem solid #877D6C;
28 }
29 footer {
30 padding: 0.5rem;
31 border-top: 0.1rem solid #877D6C;
32 width: 100%;
33 color: #736B5E;
34 }
35 .callout {
36 font: bold 290% Georgia, "Times New Roman", Times, serif;
37 color: #333;
38 margin: 0;
39 padding: 1.5rem;
40 }
41 aside {
42 background: #f6e5c3;
43 padding: 1rem;
44 }
45 aside p {
46 font-variant: small-caps;
47 }
48 nav ul {
49 margin: 0;
50 padding: 0.1rem;
51 }
52
53 nav li {
54 margin: 0.5rem 0;
55 font-size: 120%;
56 list-style-type: none;
57 padding: 0.5rem;
58 }
59 nav a:link {
60 display: block;
61 background-color: #DCCBAC;
62 border-radius: 0.3em;
63 width: 90%;
64 font-size: 1rem;
65 margin: 0.2rem;
66 padding: 0.2rem;
67 color: #736B5E;
68 text-decoration: none;
69 font-weight: bold;
70 }
71 nav a:visited, nav a:hover, nav a:active, nav a:focus {
72 color: #333;
73 }
74 .imgleft {
75 float: left;
76 margin-right: 0.5rem;
77 max-width: 100%;
78 height: auto;
79 }

```

Abb. 9.4: Der Code in demo.css

Keine Sorge, ich werde Ihnen diese Stilvorlagen eine nach der anderen erklären! Möglicherweise sind Sie der Meinung, dass das extrem viel Code ist. In diesem Fall muss ich Ihnen sagen: Das ist noch harmlos – komplexe Stylesheets für umfangreiche professionelle Projekte benötigen ein Vielfaches an Code. Aber ich kann Sie trösten: Wenn Sie dieses Beispiel hier wirklich verstanden haben, haben Sie eine solide Grundlage, auch viele der komplizierteren Stylesheets nachzuvollziehen.

#### Hinweis:

Sie finden das komplette Beispiel-Layout im Ordner **MMDE2J Beispiel-Layout**. Dieser liegt im Ordner **Downloads**, den Sie im heftbezogenen Downloadbereich Ihrer Lernplattform finden.

## 9.2 Die Stilvorlagen im Detail erklärt

Jetzt zu den einzelnen Regeln aus dem externen Stylesheet **demo.css**. Ich werde dazu den kompletten Inhalt dieser Datei hier in den Text kopieren und anschließend Schritt für Schritt jede Regel erklären. Bitte werfen Sie zum Vergleich immer wieder mal einen Blick auf den Gesamtzusammenhang in der entsprechende Abbildung des CSS-Codes im vorigen Abschnitt, während Sie die folgenden Erklärungen (grüner Text) durcharbeiten. Ich werde dabei ganz simple Regeln, die Sie bereits aus den vorausgegangenen Kapiteln kennen, oder die vollkommen selbsterklärend sind, nicht mehr erklären:

```
body {
 font: 100%/1.3 Verdana, Arial, Helvetica, sans-serif;
 color: #000;
 background: #ADA189;
 margin: 0;
}
```

Mithilfe der Kurzschreibweise **font** werden Schriftgröße/Zeilenhöhe (Letztere nach dem Schrägstrich als Vielfaches der Schriftgröße) und eine generische Schriftart festgelegt.

Die Schriftfarbe wird für den gesamten **body** auf **#000**; festgelegt (Schwarz)

**body** erhält die Hintergrundfarbe **#ADA189**; (Hellbraun)

Die Außenabstände des **body** werden in einer Kurzschreibweise an allen vier Seiten auf Null gesetzt.

#### Hinweis:

Sie finden am Ende dieses Kapitels eine Übersicht zu den in CSS häufig benutzten Kurzschreibweisen.

```
#holder {
 max-width: 60rem;
 background: #FFF;
 margin: 0 auto;
 display: flex;
 flex-wrap: wrap;
}
```

Mit der Deklaration `max-width: 60rem;` wird ein „flüssiges Layout“ (engl. „liquid layout“) erzeugt. Die Breite des Layoutcontainers kann sich flexibel der zur Verfügung stehenden Viewportbreite anpassen – allerdings nur bis zu einer Maximalbreite von `60rem = 960px`

`#holder` erhält die Hintergrundfarbe `#fff` (Weiß)

Die Außenabstände des Containers werden in einer Kurzschreibweise wie folgt festgelegt: oben=0, rechts=auto, unten=0, links=auto. Durch `auto` beim linken und rechten Außenabstand wird das Element zentriert.

Wir machen bei diesem Beispiel unseren umfassenden Layoutcontainer `#holder` durch `display: flex;` zum Flexcontainer. Die darin befindlichen Elemente – hier also `header`, `nav`, `main`, `footer` – werden damit zu Flexitems.

Die Deklaration `flex-wrap: wrap;` sorgt für eine mehrzeilige Darstellung.

```
header {
 background: #DCCBAC;
 width: 100%;
}
```

`header` erhält die Hintergrundfarbe `#DCCBAC` (ein helles Beige)

`header` erhält eine Breite von 100 %, steht also alleine in einer Zeile.

```
nav {
 text-align: center;
 width: 30%;
 padding: 1rem;
}
```

`nav` erhält immer 30 % der aktuellen Breite des Layoutcontainers `#holder` (diese hängt flexibel von der aktuellen Viewportbreite ab).

```
main {
 flex: 1;
 padding: 1rem;
 border-left: 0.1rem solid #877D6C;
}
```

Mit `flex: 1;` erhält `main` den gesamten neben `nav` verfügbaren Platz.

```
footer {
 padding: 0.5rem;
 border-top: 0.1rem solid #877D6C;
 width: 100%;
 color: #736B5E;
}
```

`footer` erhält eine Breite von 100 %, steht also alleine in einer Zeile.

```
.callout {
 font: bold 290% Georgia, "Times New Roman",
 Times, serif;
 color: #333;
 margin: 0;
 padding: 1.5rem;
}
```

Das ist die Klasse, die auf die **h1**-Überschrift im **header** angewandt wird. Der Klassenname **.callout** ist natürlich frei gewählt („callout“ bedeutet im Englischen „Angabe“ oder „Textzeile“). Hier wird zunächst in einer Kurzschreibweise die Schrift auf fett, auf 290 Prozent der Standardschriftgröße und auf die Schriftart Georgia (bevorzugt) eingestellt.

Mit der Kurzschreibweise **font** wird neben anderen Werten auch die Schriftart definiert. Weil „Times New Roman“ aus mehreren Wörtern mit Leerzeichen dazwischen besteht, muss der gesamte Schriftname in Anführungszeichen geschrieben werden!

Die Schriftfarbe soll **#333** sein.

Mit der Kurzschreibweise **margin:0;** werden alle vier Werte für den Außenabstand der Überschrift auf Null gesetzt, damit kein Browser seine eigene Interpretation von **h1** „ausleben“ darf!

Mit derselben Kurzschreibweise wird der Innenabstand für die Überschrift auf allen vier Seiten auf **1.5 rem** festgelegt.

```
aside {
 background: #f6e5c3;
 padding: 1rem;
}
aside p {
 font-variant: small-caps;
}
nav ul {
 margin: 0;
 padding: 0.1rem;
}
nav li {
 margin: 0.5rem 0;
 font-size: 120%;
 list-style-type: none;
 padding: 0.5rem;
}
```

Durch die Deklaration **list-style-type: none;** werden die Aufzählpunkte („Bullets“) der Listeneinträge entfernt.

```
nav a:link {
 display: block;
 background-color: #DCCBAC;
 border-radius: 0.3em;
 width: 90%;
 font-size: 1rem;
 margin: 0.2rem;
 padding: 0.2rem;
 color: #736B5E;
 text-decoration: none;
 font-weight: bold;
}
```

Wie ich Ihnen bereits in einem früheren Kapitel erklärt habe, sind Verweise normalerweise **inline-Elemente**, d. h., sie beanspruchen **keine** eigene Zeile, sondern „laufen“ einfach mit dem Fließtext mit, in den sie eingebettet sind. Mit **display: block;** werden die Verweise jetzt zu block-Elementen.

Dadurch verhalten sie sich wie Buttons, d. h., man kann sie entlang der gesamten Breite anklicken, die ihnen zur Verfügung gestellt wird und nicht nur da, wo tatsächlich ein Linktext steht.

Durch die Deklaration **width: 90%;** ist das in diesem Fall 90 % der Breite des **nav**-Elements, das die Liste mit den Links enthält.

Durch **border-radius: 0.3em;** erhalten die Buttons mit der Hintergrundfarbe **#DCCBAC;** abgerundete Ecken.

```
nav a:visited, nav a:hover, nav a:active, nav a:focus {
 color: #333;
}
```

Mit float fließt  
der Text rechts  
an dem Bild  
entlang

Für alle Pseudoklassen wird ein Grauton als Schriftfarbe festgelegt.

```
.imgleft {
 float: left;
 margin-right: 0.5rem;
 max-width: 100%;
 height: auto;
}
```

Das Bild, dem die Klasse **.imgleft** zugewiesen wird, steht **links** und wird **rechts** vom Text umflossen.

#### Hinweis:

Dazu noch ein wichtiger Hinweis: In unserem HTML-Code-Beispiel arbeiten wir bekanntlich mit „Dummy-Links“, d. h. mit dem Attribut für das Linkziel **href=#**. Falls man gerade keine realen Unterseiten zum Verlinken hat, ist das eine praktische Methode, damit die Links im Browser das übliche Aussehen und Verhalten aufweisen.

So, jetzt haben Sie es wirklich geschafft. Das war wirklich ein „dicker Brocken“ und Sie haben eine Pause verdient! Ich empfehle Ihnen aber: Wiederholen Sie dieses Kapitel unbedingt. Denn darin habe ich Ihnen ziemlich viel Stoff zugemutet und der muss sich erst mal „setzen“. Ich kann Sie trösten: Bis ich das seinerzeit selber alles begriffen habe, hat das auch eine ganze Weile gedauert!



### CSS-Kurzschreibweisen

Wenn Sie sehr viel CSS-Code schreiben müssen, werden Sie gerne von den Kurzschreibweisen Gebrauch machen, die für zahlreiche CSS-Eigenschaften zur Verfügung stehen. Da diese im Beispiel in mehreren CSS-Deklarationen vorgekommen sind, sollen sie hier noch einmal zusammengefasst werden. Dazu wird immer jeweils im Anschluss an die ausführliche Schreibweise die betreffende Kurzschreibweise beschrieben.

#### 1. font

```
font-style: italic;
font-variant: small-caps;
font-weight: bold;
font-size: 1.4em;
line-height: 150%;
font-family: Georgia, serif;
```

wird abgekürzt zu

```
font: italic small-caps bold 1.4em/1.5 Georgia, serif;
```

Die Kompakteigenschaft **font** verlangt, dass als vorletzte Angabe die Schriftgröße (kombiniert mit der Zeilenhöhe) definiert wird und zuletzt die Schriftart.

CSS-Eigenschaften, die den Standardwert behalten sollen, können in der Reihenfolge einfach ausgelassen werden, z.B.:

```
font: bold 1.4em/1.5 Georgia, serif;
```

#### 2. margin und padding

Anstatt einzelne Werte für **margin-top**, **margin-right**, **margin-bottom** oder **margin-left** (bzw. **padding-...**) anzugeben, können alle Werte mit in der einen CSS-Eigenschaft **margin** (bzw. **padding**) zusammengefasst werden.

```
margin-top: 30px;
margin-right: 10px;
margin-bottom: 40px;
margin-left: 20px;
```

wird also abgekürzt zu

```
margin: 30px 10px 40px 20px;
```

Die Reihenfolge erfolgt immer im Uhrzeigersinn, angefangen wird beim **margin-top** Wert.

Noch kürzer kann man **margin** (oder ganz analog **padding**) zusammenfassen, wenn mehrere Werte gleich sind.

```
margin: 30px 10px 30px 10px;
```

kann man zusammenfassen zu:

```
margin: 30px 10px;
```

In dem Fall werden zuerst die identischen Werte für **margin-top** und **margin-bottom** zusammengefasst. Dann folgen die identischen Werte für **margin-left** und **margin-right**.

Sind alle vier Werte identisch, wie bei:

```
margin: 0px 0px 0px 0px;
```

verkürzt sich die Schreibweise zu:

```
margin: 0px;
```

Hinweis: Bei dem Wert **0** kann die Einheit (hier **px**) auch weggelassen werden, d.h., man kann auch schreiben:

```
margin: 0;
```

Bei Werten ungleich **0** muss die Einheit (also **px**, **%** oder auch **em/rem**) unbedingt **ohne** Leerzeichen direkt neben den Wert geschrieben werden. Sonst wird die Stilangabe als CSS-Syntaxfehler aufgefasst und vom Browser gar nicht berücksichtigt.

### 3. background

Auch die CSS-Eigenschaften **background-color** (Hintergrundfarbe), **background-image** (Hintergrundbild, s. die Links unten), **background-repeat** (Art der Wiederholung des Hintergrundbilds), **background-attachment** (scrollt der Hintergrund mit oder bleibt er fix) und **background-position** (Anfangsposition des Hintergrundbilds) für die Definition von Hintergründen lassen sich in einer viel praktischeren Kurzschreibweise **background** schreiben.

Unter anderem stehen hier folgende Werte zur Verfügung:

```
background: [background-color] [background-image] [background-repeat] [background-attachment] [background-position];
```

d.h. aus

```
background-color:#066;
background-image: url("bg.png");
background-repeat:repeat;
background-attachment: fixed;
background-position:top left;
```

wird

```
background:#066 url("bg.png") repeat fixed top left;
```

Auch bei der Background-Eigenschaft können Sie nicht benötigte Werte weglassen, es wird dann automatisch der Standardwert genutzt (so wird bei unserem Beispiel bei **body** und **#holder** nur der Wert für **background-color** definiert).

In dieser Übersicht geht es um die CSS-Kurzschreibweisen. Dies ist also nicht der richtige Rahmen, das wichtige Thema Hintergrundbilder zu behandeln. Sie finden dazu eine gute Übersicht unter:

<https://wiki.selfhtml.org/wiki/CSS/Tutorials/Hintergrund>

#### 4. border

Platzsparend und übersichtlich ist auch die Kurzschreibweise für **border** (Rahmen). Hier kann man die Eigenschaften **border-width** (Rahmenbreite), **border-style** (Stil des Rahmens, z.B. als Fläche, gestrichelt, gepunktet usw.) und **border-color** (Rahmenfarbe) in einer Deklaration vereinen.

Folgende Werte stehen hier zur Verfügung:

```
border: [border-width] [border-style] [border-color];
```

d.h. aus

```
border-width: 5px;
border-style: solid;
border-color: #666;
```

wird

```
border: 5px solid #666;
```

#### 5. Farbangaben

Wenn zwei aufeinanderfolgende Zeichen in der hexadezimalen Schreibweise gleich sind, reicht es auch, nur eines davon hinzuschreiben.

Statt

```
#ff11dd
```

schreibt man kurz

```
#f1d
```

## Zusammenfassung

- Im Beispiel wurde ein **div**-Element mit ID als Layoutcontainer verwendet.
- Dem Layoutcontainer können mithilfe einer Klasse oder auch eines ID-Selektors zahlreiche Eigenschaften wie Hintergrundfarbe, Breite, Außen- und Innenabstand sowie Rahmenlinien zugewiesen werden.
- Mit der Deklaration **max-width: 60rem;** für den Layoutcontainer wird ein „flüssiges Layout“ erzeugt.
- Bei der Flexbox-Technologie unterscheidet man: Flexcontainer und Flexitems.

- Ein Element muss mit **display: flex;** zum Flexcontainer gemacht werden: das ist das umfassende Element der flexibel angeordneten Elemente.
- Die Kindelemente des Flexcontainers werden angeordnet und heißen Flexitems.
- Mit **display: block;** werden die Verweise zu block-Elementen.
- Dadurch verhalten sie sich wie Buttons, d. h., man kann sie entlang der gesamten Breite anklicken, die ihnen zur Verfügung gestellt wird.
- Um Bilder von Text umfließen zu lassen, verwendet man die Eigenschaft **float**.
- Für CSS-Deklarationen existieren viele nützliche Kurzschreibweisen.

### Aufgabe zur Selbstüberprüfung

- 9.1
- Ich hoffe, Sie haben noch die Beispieldatei aus dem HTML-Studienheft aufbewahrt, so wie ich Ihnen das geraten hatte. Das war die Aufgabe zu dem Kapitel „Ein Praxisbeispiel“. Damals sollten Sie einen Ordner **html-praxis** anlegen und darin die Datei **index.html** mit dem Code, der die Basis für das Beispiel „Meine tolle Firma“ in diesem Kapitel bildet, ablegen.
  - Öffnen Sie dieses Dokument in Ihrem Code-Editor und ergänzen Sie die Modifikationen, so wie sie zu Beginn dieses Kapitels erklärt worden sind.
  - Falls Sie das Dokument aus dem HTML-Studienheft nicht (mehr) haben, müssen Sie jetzt leider den ganzen Code für „Meine tolle Firma“ neu eingeben. Das wäre aber auch nicht so schlimm, denn „Übung macht den Meister“!
  - Ob Sie nun das alte Dokument modifizieren konnten oder ganz neu eingeben mussten – wichtig ist: Es muss sich innerhalb eines Ordners befinden.
  - Öffnen Sie nun bitte ein neues Dokument in Ihrem Code-Editor und geben Sie darin sorgfältig den CSS-Code für die Layout-Vorstufe ein, wie das in diesem Kapitel gezeigt worden ist. Speichern Sie das Dokument unter dem Namen **demo.css** im **selben Ordner wie die HTML-Datei**, damit es auch zum **href**-Attribut der Link-Zeile in **index.html** passt!
  - Vollziehen Sie nun alle in diesem Kapitel beschriebenen Schritte bis zum endgültigen Layout nach. Modifizieren Sie dazu das externe Stylesheet **demo.css** nach und nach bis zum endgültigen Stand.
  - Denken Sie bitte daran, während der Bearbeitung von **demo.css** diese Datei immer wieder zu speichern und anschließend Ihre Browserdarstellung zu aktualisieren.

---

## Schlusswort

Geschafft! Das war sicher kein „Spaziergang“, sich dieses Thema zu erschließen. Wenn Sie noch Anfänger waren und nach dem ersten Durcharbeiten dieses Studienhefts das Gefühl hatten, dass Ihnen noch nicht alles restlos klar geworden ist, kann ich Sie trösten: Das ist vollkommen normal! Ich kenne niemanden, der das alles sofort versteht und dann auch gleich umsetzen kann. Nehmen Sie sich die Zeit und wiederholen Sie die Kapitel, die Sie als am schwierigsten empfunden haben. Vergessen Sie vor allem die Aufgaben zur Selbstüberprüfung nicht: „Übung macht den Meister“.

Ich hoffe jedenfalls, es hat Ihnen auch wieder etwas Spaß gemacht, mit diesem Studienheft zu arbeiten. Cascading Style Sheets sind heute auf jeden Fall entscheidend für Design und Layout anspruchsvoller Webseiten. Sie können sich jetzt sicher besser vorstellen, was für ein Potenzial in dieser Technik steckt. Es ist natürlich unmöglich, im Rahmen eines Studienhefts das Thema CSS vollständig abzuhandeln. Inzwischen ist der Umfang dieses Gebiets wohl weitaus größer als der von HTML selbst.

Deshalb empfehle ich Ihnen, sich zum Thema CSS vielleicht noch ein gutes Buch zu besorgen und sich auch im Netz weiterzubilden. Es gibt dort sehr viele und zum Teil sehr gute Tutorials zu CSS und allen anderen relevanten Themen des Webdesigns. Wir Autoren/Autorinnen geben uns alle Mühe, dass die Studienhefte korrekt und gut geschrieben sind. Trotzdem kann es manchmal hilfreich sein, ein Thema auch einmal von jemand anderem erklärt zu bekommen. Einige Anregungen finden Sie in der Literatur- und Linkliste im Anhang.

Vielen Dank für Ihre Aufmerksamkeit und weiter viel Erfolg!

Dr. Thomas Wecker

Beim Thema  
CSS lernt man  
nie aus!

## A. Lösungen der Aufgaben zur Selbstüberprüfung

Hier finden Sie die Lösungen zu den Aufgaben zur Selbstüberprüfung in den einzelnen Kapiteln. Bei offenen Aufgaben mit freien Formulierungen kommt es nicht auf eine wörtliche Übereinstimmung an, sondern auf den Inhalt. Entsprechen Ihre Ergebnisse nicht den Lösungen, wiederholen Sie bitte das entsprechende Kapitel und bearbeiten Sie die zugehörigen Aufgaben zur Selbstüberprüfung nach einer Pause erneut.

Alle Lösungen zu den Praxisaufgaben zur Selbstüberprüfung finden Sie im heftbezogenen Downloadbereich Ihrer Lernplattform.

### Kapitel 1–6 und 8–9

Für diese Kapitel finden Sie die Lösungen der Aufgaben zur Selbstüberprüfung im Ordner **MMDE2J Lösungen Aufgaben zur Selbstüberprüfung**. Dieser liegt im Ordner **Downloads**, den Sie im heftbezogenen Downloadbereich Ihrer Lernplattform finden.

### Kapitel 7

- 7.1 In unserem Beispiel erben z.B. **alle** Elemente die Deklaration **font-family: Helvetica, Arial, sans-serif;** für die Schrift, die für den **body** definiert wurde, weil **alle** Elemente Nachkommen des **body** sind.

Hier ein Beispiel, wie ein Inline Style die Regel überschreibt, die bereits früher (bzw. „weiter entfernt“) definiert worden ist. In unserem externen Stylesheet ist mit **text-align: center;** festgelegt, dass die **h1**-Überschrift zentriert sein soll.

```
<body>
 <h1 style="text-align: left;">Willkommen in der Welt
der
Style Sheets!</h1>
 <p class="grosstext">Style Sheets zu benutzen ist ganz einfach:</p>

 schreibe die Regeln auf
```

Dies wird durch den oben gezeigten Inline Style überschrieben, wie der folgende Ausschnitt aus der Darstellung im Firefox zeigt:



Willkommen in der Welt  
der *Style Sheets!*  
STYLE SHEETS ZU BENUTZEN IST GANZ EINFACH:

## B. Glossar

Hier finden Sie eine kurze Erklärung zu den im Studienheft eingeführten Begriffen:

Content	Bezeichnet allgemein den Inhalt einer Webseite – unabhängig von deren Layout.
Deklaration	Kombination von Eigenschaft und Wert.
Flüssiges Layout	Layout, das sich automatisch prozentual an die Größe des Viewports anpasst.
Formatvorlagen	Siehe Stilvorlagen.
ID	Eindeutiger Bezeichner eines HTML-Elements.
Klasse	Klassenselektor, der über den Namen mit dem Attribut <b>class</b> einem HTML-Element zugewiesen wird. Klassen ermöglichen die unterschiedliche Formatierung gleichartiger struktureller Elemente. Klassennamen ohne Selektor sind flexibel einsetzbar.
Kommentar	Element der CSS-Sprache, das vom Browser nicht ausgewertet wird. Kommentare gibt es in allen Programmiersprachen.
Kontextabhängiger Selektor	Identifiziert Elemente in einer ganz bestimmten Umgebung (Kombinator).
Layout	Anordnung von Gestaltungselementen auf einer Webseite.
MIME-Typ	„Multipurpose Internet Mail Extensions“ – teilt dem Browser mit, von welchem Typ die Daten sind, die vom Server übertragen werden.
Pseudoklasse	Bezeichnet Elemente, für die es keine eindeutigen HTML-Tags gibt, wie z.B. die verschiedenen Zustände von Verweisen.
Regel	Eine oder mehrere Deklarationen für einen bestimmten Selektor (entspricht der Stil- oder Formatvorlage).
Selektor	Ein Selektor ist ein Ausdruck, um ein Element auszuwählen, für das die anschließend definierte Regel gilt. Bei gruppierten Selektoren können es auch mehrere unterschiedliche Elemente sein.
Serifen	Verzierungen an einem Buchstaben.
Stilvorlagen	Sammlung von Gestaltungsregeln, die auf die strukturellen Elemente in einer Seite angewandt werden.
Typografie	Umgang mit Texten und Schriften auf einer Seite.

Viewport	Bereich des Browserfensters, der für die Darstellung der Inhalte zur Verfügung steht.
W3C	World Wide Web Consortium.

## C. Literaturverzeichnis

- Laborenz, K. (2015). *CSS: Das umfassende Handbuch*. Rheinwerk Computing.
- Maurice, F. (2013). *CSS3. Die neuen Features für fortgeschrittene Webdesigner*. dpunkt.
- Maurice, F. (2019). *HTML & CSS für Dummies*. Wiley-VCH.
- Wenz, C. & Prevezanos, C. (2018). *HTML5 und CSS3 – Start ohne Vorwissen*. Markt + Technik Verlag.
- Wenz, C., Maurice, F. & Hauser, T. (2016). *Das Website Kompendium. Programmierung und Design*. Markt+Technik.
- Wolf, J. (2019). *HTML5 und CSS3. Das umfassende Handbuch*. Rheinwerk Computing.

## Informationsquellen im Internet

Weitere interessante Quellen zum Thema CSS zeigt folgende Liste. Allerdings kann ich natürlich nicht garantieren, dass die URLs für immer so korrekt sind. „Googlen“ Sie also bitte einfach nach Begriffen wie „CSS“ oder „CSS-Layout“ und Sie werden zahllose interessante Seiten zu diesen Themen finden.

- <https://wiki.selfhtml.org/wiki/CSS>
- [https://developer.mozilla.org/de/docs/Learn/Getting\\_started\\_with\\_the\\_web/CSS\\_basics](https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/CSS_basics)
- <https://www.w3schools.com/css/> (das ist keine offizielle Seite des W3C)
- <https://www.w3.org/Style/CSS/Overview.de.html>
- <https://little-boxes.de/little-boxes-teil1-online.html>
- <https://css-tricks.com/>

## D. Abbildungsverzeichnis

Abb. 1.1	Beispiel für einen mit Formatierungen durchsetzten HTML-Code .....	3
Abb. 2.1	Ein einfaches HTML-Dokument .....	4
Abb. 2.2	Das Beispieldokument ohne CSS .....	5
Abb. 2.3	Das Beispieldokument mit CSS .....	5
Abb. 2.4	Das externe Stylesheet style1.css .....	6
Abb. 2.5	Die Web-Konsole von Firefox zeigt einen CSS-Syntaxfehler an .....	9
Abb. 4.1	Die additive Farbmischung .....	18
Abb. 4.2	Einige CSS-Farbnamen .....	19
Abb. 4.3	Das subtraktive Farbmodell .....	20
Abb. 6.1	Ein eingebettetes Stylesheet .....	25
Abb. 6.2	Inline Styles .....	25
Abb. 7.1	Kaskade .....	28
Abb. 8.1	Das Box-Modell .....	32
Abb. 8.2	Ein HTML-Container mit dem div-Element .....	34
Abb. 8.3	Beispiel für die Anwendung des span-Tags .....	34
Abb. 8.4	Flexbox: ein simples Beispiel .....	36
Abb. 8.5	Das Flexbox-Beispiel im Browser .....	37
Abb. 8.6	Flexbox: die Eigenschaft flex .....	38
Abb. 8.7	Flexbox: main erhält mehr Platz .....	38
Abb. 8.8	Flexbox: die Eigenschaft order .....	38
Abb. 8.9	Flexbox: der footer erscheint zuerst .....	39
Abb. 8.10	Flexbox: die Elemente bekommen bestimmte Breiten .....	39
Abb. 8.11	Flexbox: Die Elemente haben definierte Breiten .....	40
Abb. 8.12	Flexbox: die Eigenschaft flex-wrap .....	40
Abb. 8.13	Flexbox: header und footer bekommen eine eigene Zeile .....	41
Abb. 8.14	Flexbox: CSS-Code für das fertige Beispiel .....	41
Abb. 8.15	Flexbox: das fertige Beispiel im Browser .....	41
Abb. 8.16	Float: Beispielcode .....	42
Abb. 8.17	Float: der Text umfließt das Bild .....	43
Abb. 9.1	Der leicht modifizierte HTML-Code für die Beispielseite .....	47
Abb. 9.2	Die Beispielseite im Browser (mit div-Container, ohne CSS) .....	48
Abb. 9.3	Das fertige CSS-Layout im Browser .....	49
Abb. 9.4	Der Code in demo.css .....	50
Abb. F.1	Vorlage für CSS-Einsendaufgabe .....	68

## E. Sachwortverzeichnis

### A

a:active .....	22
a:focus .....	22
a:hover .....	22
a:link .....	22
a:visited .....	22
Attributwert .....	4
Auszeichnungssprache .....	1

### B

Betriebssystem .....	13
block .....	31
border .....	32
Border-Box .....	33
border-width .....	32
Box-Modell .....	31
Breite .....	31
Browserfenster .....	13

### C

CMYK-Modell .....	19
Container .....	33
Content-Box .....	33
CSS Grid .....	35
CSS-basiertes Layout .....	31
CSS-Kommentar .....	12
CSS-Syntax .....	14

### D

Deklaration .....	10
descendant selector .....	11
Design .....	3
display .....	31
DIV .....	33

### E

Eigenschaft .....	10
eingebettetes Stylesheet .....	24
Elternelement .....	11, 14, 15
em .....	14
embedded Stylesheet .....	24
Erscheinungsbild .....	3
externe Stylesheets .....	6, 24

### F

Farbmodell	
additives .....	18
subtraktives .....	19
Flexbox .....	35
Flexcontainer .....	37
Flexitems .....	37
flex-wrap .....	40
Fließtext .....	21
float .....	42
Formatvorlagen .....	1

### G

generische Schriftart .....	10, 13
Gesamtbreite .....	31, 32
Gesamthöhe .....	32
Gestaltung .....	3
Gewicht .....	29
gruppierte Selektoren .....	11

### H

Hierarchie .....	15, 27
HTML .....	1
HTML-Tag .....	6, 10

### I

id-Attribute .....	35
ID-Selektoren .....	35
Inhaltsbereich .....	33
inline .....	31
Inline Styles .....	25
Innenabstand .....	31
interne Stylesheets .....	24

### K

Kaskade .....	28
Kindelement .....	11, 27, 37
Klassen .....	21
Klassennamen .....	21
Klassenselektoren .....	21
Komma .....	11
Kommentar .....	12
kontextabhängiger Selektor .....	11, 27

<b>L</b>		Struktur .....	3
Layout .....	3	Subtraktive Farbmischung .....	20
Layoutcontainer .....	35, 37		
Leerzeichen .....	10	<b>T</b>	
		Times New Roman .....	7
<b>M</b>		True Color .....	18
Markup Language .....	3	Typografie .....	3, 12
MIME-Typ .....	4		
		<b>U</b>	
<b>N</b>		Umfließen .....	43
Nachfahre .....	11		
Nachfahrenkombinatoren .....	11	<b>V</b>	
Nachfahre-Selektor .....	11	Vererbung .....	27
		Verknüpfung .....	4, 24
<b>O</b>			
order .....	39	<b>W</b>	
		Webdesign .....	12
<b>P</b>			
Primärfarben .....	18	<b>Z</b>	
Print .....	19	Zeilenumbrüche .....	10
Pseudoklassen .....	22		
Punktesystem .....	29		
<b>R</b>			
Rahmenstärke .....	31		
Regel .....	3, 7, 10, 51		
relative Maßeinheiten .....	14		
relativer Pfad .....	4, 24		
rem .....	14, 15		
Responsive Webdesign .....	14		
RGB-Modell .....	18		
<b>S</b>			
Schriften .....	13		
Schriftenliste .....	13		
Schriftgröße .....	15		
Selektor .....	10		
SELFHTML .....	11		
Semikolon .....	11		
Serifen .....	13		
SPAN .....	33		
Spezifität .....	29		
Standardschrift .....	7, 32		
Standard-Schriftgröße .....	7		
Stilvorlagen .....	1, 51		

## F. Einsendeaufgabe

### Gestaltung von Webseiten mit HTML5 und CSS3

Code:

**MMDE02J-XX03-K04**

Name:	Vorname:
Postleitzahl und Ort:	Straße:
Studien- bzw. Vertrags-Nr.:	Lehrgangs-Nr.:

Fernlehrer/in:

Datum:

Note:

Unterschrift Fernlehrer/in:

Bitte reichen Sie Ihre Lösungen über die Online-Lernplattform ein oder schicken Sie uns diese per Post. Geben Sie bitte immer den Code zum Studienheft an (siehe oben rechts).

#### Hinweis:

Im **Aufgabenteil A** sollen Sie als **Praxisaufgabe** ein HTML-Dokument erstellen. Erstellen Sie alle erforderlichen Dateien der Praxisaufgabe A innerhalb des Ordners **nachname-mmde02j-aufgabe**. Ersetzen Sie dabei **nachname** durch Ihren persönlichen Nachnamen.

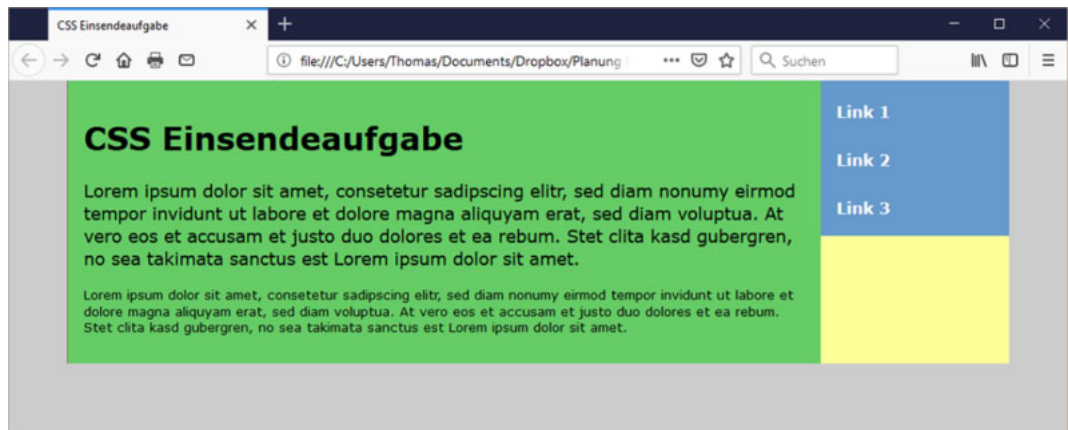
Im **Aufgabenteil B** geht es um **Multiple-Choice-Fragen** zum Thema CSS. Bitte erfassen Sie die Lösungen dazu in digitaler Form. Sie können dazu die Digitale Einsendeaufgabe (das Wordformular) verwenden, die Sie im heftbezogenen Download finden. **Speichern Sie diese Datei** mit Ihren Lösungen **zusätzlich** im oben genannte Ordner **nachname-mmde02j-aufgabe** ab.

**Erstellen Sie anschließend ein ZIP-Archiv des Ordners** mit ihren Lösungen aus Teil A und B.

**Reichen Sie dieses ZIP-Archiv anschließend über Ihre Online-Lernplattform ein.**

### A. Praxisaufgabe

Sehen Sie sich bitte die folgende Abbildung einer Webseite an, wie sie im Firefox dargestellt wird. Entwickeln Sie bitte nach dieser Vorlage ein HTML-Dokument **css-aufgabe.html** mit einem zugehörigen externen Stylesheet **css-aufgabe.css**. Dabei müssen Sie weder dieselben Maße einhalten noch genau dieser Farbgebung folgen. Das Layout sollte aber diesem Muster entsprechen. Dazu hier noch einige Informationen und Anforderungen:



**Abb. F.1:** Vorlage für CSS-Einsendeaufgabe

- Legen Sie dazu als Erstes einen neuen Ordner auf Ihrem Rechner an und nennen Sie ihn **nachname-mmde02j-aufgabe**.
- Speichern Sie bitte darin die beiden Dateien **css-aufgabe.html** und **css-aufgabe.css**.
- Orientieren Sie sich bei Ihrem Entwurf an dem Beispiel aus Kapitel 9. Das Layout der Aufgabe ist eine stark vereinfachte Variante dieses Beispiels, es soll also ein „flüssiges Layout“ erstellt werden.
- Die Navigationsleiste soll **rechts** platziert werden.

#### Hinweis:

Überlegen Sie bitte, welches HTML-Element Sie mit einer Flexbox-Eigenschaft modifizieren müssen, damit die Navigation auf der **rechten Seite** angezeigt wird. Beschaffen Sie sich bei Bedarf hier weitere Informationen: <https://kulturbaunause.de/blog/einfuehrung-in-das-flexbox-modell-von-css/>

- Die **Verweise** im rechten Container sollen **als Liste** ausgezeichnet sein.
- Die Listenelemente sollen **keine Aufzählungspunkte** haben.
- Die Verweise sollen im Normalzustand **keine Unterstreichungen** haben.
- Sie können als freiwillige Zusatzarbeit die Verweise als grafische Buttons gestalten.
- Mithilfe einer Pseudoklasse soll als **Hover-Effekt** für die Links die Unterstreichung realisiert werden. Denken Sie daran, dass Sie als Linkpfad im **href**-Attribut den „Gartenzaun“ verwenden.

- Setzen Sie bitte **ähnliche Hintergrundfarben wie in der Vorlage** um. Überlegen Sie bitte, welchem Element Sie dazu welche Farbe zuweisen müssen.
- Auf der ganzen Seite soll eine **Schriftart mit der Prioritätsreihenfolge Verdana, Geneva, sans-serif** verwendet werden.
- **Nutzen Sie Klassen und kontextabhängige Selektoren** (Kombinatoren). Hier können Sie Ihrer Fantasie freien Lauf lassen!
- **Im linken Container** sollen **nach der Hauptüberschrift zwei Absätze** eingefügt werden, denen die Klassen `.grosstext` und `.kleintext` mit entsprechenden CSS-Regeln zugewiesen sind.

*max. 60 Pkt.*

### B. Multiple Choice

Bitte beantworten Sie die folgenden 10 Fragen durch Ankreuzen der richtigen Antwort. Zu jeder Frage gibt es nur **eine** (!) richtige Antwort. Jede korrekt gelöste Frage wird mit 4 Punkten gewertet.

**Bitte lesen Sie sich die Fragen aufmerksam durch, bevor Sie antworten.**

**Viel Erfolg!**

1. CSS führt zur Trennung von ...
  - ☐ ... Struktur und Inhalt
  - ☐ ... Layout und Design
  - ☐ ... Inhalt und Gestaltung
  - ☐ ... Struktur und Gestaltung
2. Frage: Welche CSS-Codezeile ist korrekt?
  - ☐ `p {font-family:Helvetica, font-weight:bold}`
  - ☐ `p {font-family:Helvetica, sans-serif; font-weight:bold;}`
  - ☐ `p {font-family=Helvetica, sans-serif; font-weight=bold}`
  - ☐ `p {font-family:Helvetica, font-weight:bold};`
3. Welche Aussage zu kontextabhängigen Selektoren ist richtig?
  - ☐ Sie identifizieren Elemente in einem bestimmten HTML-Dokument.
  - ☐ Sie beschreiben die Vererbung der Eigenschaften von Elternelementen auf Kindelemente.
  - ☐ Sie werden auch als „Nachfahre-Selektoren“ bezeichnet.
  - ☐ Damit lässt sich die Vererbung von Eigenschaften kontrollieren.

4. Die Deklaration `{font-family:Helvetica, Arial, sans-serif;}` wendet man an, um ...
  - ☐ ... dem User die Auswahl zwischen verschiedenen Schriften zu ermöglichen.
  - ☐ ... dem Browser die Möglichkeit zu geben, aus verschiedenen Schriften zu wählen.
  - ☐ ... den Browser anzuweisen, nach Möglichkeit die erste Schrift in der Liste zu verwenden.
  - ☐ ... kontextabhängige Schriften zu verwenden.
5. Für die Vererbung von Eigenschaften gilt, ...
  - ☐ ... dass sie der Webdesigner mit entsprechenden Regeln definieren muss.
  - ☐ ... dass sie immer zwischen Elternelementen und Kindelementen stattfindet.
  - ☐ ... dass sie immer zwischen allen gleichartigen Elementen (`p`, `h1`, `h2` usw.) stattfindet.
  - ☐ ... dass sie nur innerhalb eines Containers stattfindet.
6. Welche Aussage zu der Bezeichnung „cascading“ ist richtig?
  - ☐ „Cascading“ bedeutet, dass mehrere Regeln gleichzeitig auf ein Element wirken.
  - ☐ „Cascading“ bedeutet, dass bei gleichzeitiger Wirkung mehrerer Regeln auf ein Element genau festgelegt ist, welche Regel dann tatsächlich angewandt wird.
  - ☐ „Cascading“ bedeutet dasselbe wie kontextabhängig.
  - ☐ „Cascading“ bezeichnet die hierarchische Schreibweise des CSS-Codes.
7. Welche Eigenschaften werden in dem Box-Modell beschrieben?
  - ☐ Hintergrundfarben und Ränder von Containern sowie deren Abstände untereinander.
  - ☐ Die Breite und Höhe eines `block`-Elements, dessen Außen- und Innenabstand sowie sein Rahmen.
  - ☐ Die Gesamtbreite des Contents.
  - ☐ Die hierarchische Struktur mehrerer ineinander verschachtelter `div`-Container.
8. Welche Aussage zur CSS-Spezifität ist korrekt?
  - ☐ Ein Klassenselektor setzt sich immer gegenüber einem ID-Selektor durch.
  - ☐ Inline-Styles tragen mit geringer Punktzahl zur Spezifität bei.
  - ☐ Eine CSS-Regel wird nur dann von einer weiter unten notierten Regel überschrieben, wenn beide Selektoren identisch sind und das gleiche „Gewicht“ haben.
  - ☐ Es gilt grundsätzlich die Regel, die als Erste definiert worden ist.

9. Welche Aussage zur Flexbox-Technologie ist richtig?

- ☐ Das oberste Element in der Hierarchie der HTML-Elemente ist immer der Flexcontainer.
- ☐ Durch die Deklaration **flex-wrap: wrap;** wird der Flexcontainer von den Flexitems umflossen.
- ☐ Das Element mit **flex: 1;** kommt im Layout in die erste Spalte rechts neben dem Flexcontainer.
- ☐ Der Flexcontainer ist das umfassende Element der flexibel angeordneten Flexitems.

10. Pseudoklassen werden benutzt, um ...

- ☐ ... Klassen bei beliebigen HTML-Elementen anwenden zu können
- ☐ ... Absätze mit unterschiedlichen Stilen zu definieren.
- ☐ ... Klassen ohne Typselektor zu definieren.
- ☐ ... CSS-Stilvorlagen für HTML-Elemente zu definieren, die sich nicht durch ein eindeutiges HTML-Element ausdrücken lassen.

*max. 40 Pkt.*

*insgesamt max. 100 Pkt.*

