



STRATO E-Book

Websites erstellen mit HTML und CSS

Schritt-für-Schritt-Anleitung, um visuell ansprechende und performante Websites nach eigenen Vorstellungen aufzubauen.

An abstract graphic consisting of several overlapping, curved orange lines that create a sense of movement and depth, starting from the top left and extending towards the right side of the page.

STRATO AG

Otto-Ostrowski-Straße 7
10249 Berlin
www.strato.de

Autor

Vladimir Simović

Redaktion

Vladimir Simović
Michael Poguntke

Satz

Sarah Orant

1. Auflage © 2023 STRATO AG, Berlin

Inhalt

1. Statische Websites: robust, performant und sicher	4
1.1 Gründe für statische Websites im Jahr 2022 und darüber hinaus	4
1.1.1 Niedrigere Anforderungen an den Server	5
1.1.2 Höhere Sicherheit	5
1.1.3 Niedrigere Anforderungen an das Know-how	5
2. Was benötigst du für die Reise?	6
3. Was ist HTML und was nicht?	7
3.1 Die Anatomie eines HTML-Elements	7
3.2 Das HTML-Grundgerüst	8
3.2.1 Welcher Dokumenttyp bin ich?	9
3.2.2 Die Landessprache bestimmen	10
3.2.3 Die unsichtbaren Angaben aus der Kopfzeile	10
3.2.4 Jetzt kommt der sichtbare Bereich	10
3.2.5 Häufig genutzte HTML-Elemente	11
4. Was ist CSS und wofür ist es gut?	12
4.1 Das CSS einbinden	12
4.2 Der Aufbau von CSS: Regeln, Deklarationen, Eigenschaften & Werte	13
4.2.1 Die CSS-Selektoren	14
4.4 Die Kaskade	15
4.4.1 Kaskade 1: Die Herkunft der CSS-Regel	17
4.4.2 Kaskade 2: Position vs. Spezifität vs. Selektor-Wert	18
4.5 Das Boxmodell	19
4.5.1 Block- und Inline-Elemente	22
5. Jetzt geht's los! Die Beispielwebsite	23
5.1 Der HTML-Code der Beispielseite	23
5.2 Der HTML-Code erklärt	27
5.2.1 Wichtige Angaben für die Suchmaschinen innerhalb von <head>	27
5.2.2 Der Inhalt der Beispielseite	28
5.3 Das Schriftbild mit CSS anpassen	34
5.3.1 Anmerkung zur Schreibweise in CSS bei Schriften	34
5.3.2 Was ist ein CSS-Font-Stack?	35
5.4 Abstände und Zentrierung für einen leserlichen Inhalt	40
5.4.1 Exkurs: Kurzschreibweise von CSS-Angaben	42
5.5 Bunt und flexibel	43
5.5.1 Exkurs: Flexbox vs. Grids	44
5.6 Anpassung für mobile Geräte und Media Queries	47
5.7 Mehr als nur Schmuckwerk: Kommentare in HTML und CSS	49
5.8 Die Website hochladen	49
6. Wie geht's weiter?	50
7. Über den Autor	50

Das Web, wie wir es heute kennen, wäre ohne die beiden wichtigen Sprachen HTML und CSS nicht möglich gewesen. Diesen beiden Sprachen haben wir somit viel zu verdanken und wer sie beherrscht, dem stehen viele Türen offen.

Das muss nicht zwingend eine hauptberufliche Beschäftigung in einer Webagentur sein. Die Beherrschung von HTML und CSS kann dir auch bei deiner jetzigen Tätigkeit, bei deinem Hobby oder bei der Arbeit im Verein helfen.

Obwohl diverse Redaktionssysteme bzw. Content-Management-Systeme (CMS) wie WordPress den Markt kräftig aufmischen, wird nach wie vor [etwa ein Drittel](#)¹ der Websites ohne ein CMS betrieben.

Aber auch innerhalb eines Content-Management-Systems gibt es verschiedene Szenarien, bei denen dich HTML- und CSS-Wissen weiterbringen kann. Eine kleine [Anpassung am WordPress-Theme](#)² hier, ein Bildeffekt da oder eine zusätzlich gestylte Hinweisbox dort: All dies sind Situationen, in denen dich Kenntnisse dieser beiden Sprachen weiterbringen werden.

1. Statische Websites: robust, performant und sicher

Da wir jetzt geklärt haben, dass HTML und CSS wichtig sind, könntest du die berechtigte Frage stellen, warum man noch im Jahr 2022 auf statische Websites setzen sollte.

Wer eine einfache Website erstellen will, der muss dazu nicht sofort ein anspruchsvolles Content-Management-System installieren, mit Skripten oder Datenbanken hantieren oder teure Software kaufen. In einem beliebigen Text- und Code-Editor erstellte Dateien reichen auch heute völlig aus, um attraktive Websites umzusetzen. Das hat darüber hinaus zahlreiche Vorteile, wie eine erhöhte Sicherheit oder weitaus niedrigere Anforderungen an den Server und an das technische Know-how der Ersteller.

1.1 Gründe für statische Websites im Jahr 2022 und darüber hinaus

Websites, die auf statischem HTML & CSS basieren, sind keine Spinnereien von freiberuflichen Webdesign-Nerds, sondern finden positive Resonanz auch in weiten Teilen ihrer Kundschaft. Dass das so ist, hat mehrere handfeste Ursachen.

¹ https://w3techs.com/technologies/overview/content_management

² <https://www.strato.de/blog/schnell-einfach-css-anpassungen-in-wordpress/>

1.1.1 Niedrigere Anforderungen an den Server

Statische Websites stellen deutlich niedrigere Anforderungen an die Webserver. Das PHP-Modul und die Datenbank sind bei solchen Seiten nicht notwendig. Somit kann der gleiche Webserver deutlich schneller die Seiten ausliefern oder du kannst direkt ein kleineres und somit günstigeres Webhosting-Paket buchen.

Die niedrigeren Anforderungen an den Server helfen nicht nur die Website schneller zu laden, sondern entlasten den Server. Dadurch haben statische Websites einen deutlich grüneren CO₂-Fußabdruck als dynamische Websites, die von einem üblichen CMS ausgeliefert werden.

1.1.2 Höhere Sicherheit

Die statischen Websites haben auch deutlich niedrigere Anforderungen an die Sicherheit. Bei dynamischen Websites kommen Javascript, PHP, MySQL & Co. zum Einsatz, was die Angriffsfläche für Cyberkriminelle deutlich vergrößert.

1.1.3 Niedrigere Anforderungen an das Know-how

Statische Projekte haben einen wichtigen Nebeneffekt: Sie stellen auch deutlich niedrigere Anforderungen an denjenigen, der vorhat, eine Website zu erstellen. Dennoch ist es heute möglich, nur mit HTML und CSS schöne und ansprechende Websites zu erstellen. Und das mehr denn je, da sich HTML und vor allem CSS in den vergangenen Jahren weiterentwickelt haben, sodass diese beiden Sprachen deutlich leistungsfähiger sind, als sie es vor zehn oder 15 Jahren waren.

Heute musst du nicht mit verschachtelten Tabellen-Layouts jonglieren, um ein komplexes Web-Layout umzusetzen. Für diese Zwecke verfügt CSS mittlerweile über eigene Module mit breiter Unterstützung in gängigen Browsern. Viele Funktionen und Effekte, die früher nur mithilfe von Photoshop oder mit Javascript möglich waren, sind heute fester Bestandteil von HTML oder CSS.

Dem gegenüber stehen viele angehende Webentwickler, die sich bestens in den neuesten JavaScript-Frameworks auskennen, aber das Fundament – bestehend aus HTML und CSS – nicht wirklich beherrschen. Das führt dazu, dass viele neue Projekte aufgebläht sind, eine bescheidene Code-Qualität aufweisen und massive Probleme mit der Barrierefreiheit haben.

2. Was benötigst du für die Reise?

Um mit diesem E-Book durchzuarbeiten, benötigst du nicht viel. Am Anfang bist du mit leichtem Gepäck unterwegs: Es reicht ein bisschen Platz auf der Festplatte, ein Text- oder Code-Editor, der Browser und eine solide Portion Neugier.

Bei den Code-Editoren hast du viel Auswahl. Viele meiner Kollegen schwören auf den kostenlosen Editor von Microsoft: [Visual Studio Code](https://code.visualstudio.com)³. Dieser Editor ist allerdings mächtig und lässt viele Konfigurationsmöglichkeiten offen. Das kann am Anfang der Reise etwas einschüchternd wirken und daher greifen viele auf Notepad++ und ähnliche Editoren zurück. [Notepad++](https://notepad-plus-plus.org)⁴ bietet zwar in der Gesamtheit weniger Funktionen als Visual Studio Code. Dafür sind allerdings die notwendigen Grundlagen schon in der Standard-Konfiguration dabei – was dir den Einstieg erleichtert.

Die Wahl des Browsers bleibt dir ebenfalls überlassen. Ich bevorzuge [Mozilla Firefox](https://www.mozilla.org/de/firefox/new)⁵. Zum einen, weil ich damit gut zurechtkomme und zum anderen, weil ich die starke Dominanz von Google mit [Chrome](https://www.google.com/intl/de_de/chrome/)⁶ als äußerst schädlich für das Web empfinde.

Erst, wenn du das Projekt abgeschlossen hast und es im Web verfügbar machen möchtest, wirst ein Webhosting-Paket benötigen, wie ein [STRATO Hosting-Paket](https://www.strato.de/hosting/)⁷, und ein FTP-Programm, um die Dateien auf den Server hochzuladen.

³ <https://code.visualstudio.com>

⁴ <https://notepad-plus-plus.org>

⁵ <https://www.mozilla.org/de/firefox/new>

⁶ https://www.google.com/intl/de_de/chrome/

⁷ <https://www.strato.de/hosting/>

3. Was ist HTML und was nicht?

Du hast gelernt, warum statische Websites jetzt und auch in Zukunft eine wichtige Rolle spielen werden. Jetzt wird es Zeit, dass wir uns zuerst den Grundlagen von HTML und im nächsten Schritt auch den Grundlagen von CSS widmen.

Das Abklären der beiden Fragen aus der Überschrift ist wichtig und wird dir nachher bei der Arbeit mit HTML und CSS helfen, da du dadurch besser zwischen den Einsatzfeldern von HTML und CSS unterscheiden kannst.

Im Begriff HTML (**H**yper**T**ext **M**arkup **L**anguage) sind zwei Begriffe eingebettet, die bereits die Funktion dieser Sprache andeuten. Hypertext können wir als nicht-lineare Organisation von Texten beschreiben. Die einzelnen Texte oder Informationen werden als Knoten bezeichnet und ein Verweis beziehungsweise Link stellt die Verbindung zwischen solchen Knoten her.

Der Begriff Markup bezieht sich auf die Textauszeichnung. Wenn du HTML schreibst, dann zeichnest du Abschnitte eines Inhaltes aus: Das ist eine Überschrift erster Ordnung, das ist ein Absatz, das ist eine nummerierte Liste, das ist ein Zitat. Durch diese Auszeichnung bekommen die jeweiligen Abschnitte eine **Bedeutung** → Semantik.

Damit können wir bei HTML anderes ausschließen. HTML ist wie CSS **keine** Programmiersprache. Mit HTML wirst du weder ein Spiel programmieren, noch eine Drohne steuern. HTML ist aber auch keine Seitenbeschreibungssprache. Bei Seitenbeschreibungssprachen, wie PDF oder PostScript, wird von einer fixen Seite wie einem DIN-A4-Blatt ausgegangen und die einzelnen Elemente werden dann innerhalb dieser Fläche angeordnet. HTML dagegen ist das Aussehen und die exakte Position auf der Fläche **egal**. Es geht darum, dem Inhalt eine Bedeutung und dem Dokument eine logische Struktur zuzuweisen.

3.1 Die Anatomie eines HTML-Elements

```

194     <div class="container content headline">
195         <div class="row mb-md-5">
196             <div class="col-12 col-md-6">
197                 <h1 class="entry-title">Noch schneller: WordPress auf PH
198                 <div class="post-meta">
199                     <p>
200                         <span><i class="far fa-calendar-alt" title=""></
201                     </div>
202                 </div>
203             <div class="col-12 col-md-6 postimg">
204                 
206             </div>
207         </div>

```

Abbildung: Der Blick in den Quelltext (HTML-Code) einer Website, kann beim ersten Mal einschüchternd wirken. Damit es nicht so bleibt, werden wir uns gleich im Anschluss dem Aufbau von HTML-Elementen widmen.

Damit du die einzelnen Teile HTML-Elemente richtig benennen kannst, schauen wir uns kurz den Aufbau von HTML-Elementen an:

```
<Start-Tag Attribut="Attributwert">Der Inhalt  
des HTML-Elements</End-Tag>
```

Es fängt immer mit einem öffnenden Tag an. Innerhalb des öffnenden Tags befinden sich auch die Attribute und die passenden Attributwerte. Daraufhin folgt der sichtbare Inhalt und das Ganze wird beendet mit einem schließenden Tag. An einem konkreten Beispiel schaut das so aus:

```
<p title="Weiterführende Informationen zum Text">Hier befindet sich  
der Inhalt eines Textabsatzes.</p>
```

Die meisten HTML-Elemente verfügen über ein öffnendes und ein schließendes Tag. Es gibt allerdings Elemente, wie Bilder, Trennlinien und einige mehr, wo der öffnende gleichzeitig auch der schließende Tag ist. Hier ein Beispiel für ein Bild:

```

```

Das Beispiel zeigt nicht nur den Aufbau eines HTML-Elements, welches keinen schließenden Tag besitzt, sondern zeigt dir gleichzeitig, wie es sich verhält, wenn ein Element mehrere Attribute besitzt.

3.2 Das HTML-Grundgerüst

Im Folgenden Code-Beispiel wirst du ein HTML-Grundgerüst sehen. Mit Grundgerüst meint man das Minimum an sinnvollen Angaben, die ein HTML-Dokument benötigt. Um die folgenden Zeilen einzugeben, öffne einfach einen Texteditor auf deinem Rechner, füge die folgenden Code-Zeilen ein und speicher die Datei mit der Endung .html ab. So kannst du sie hinterher mit einem Doppelklick im Browser aufmachen.

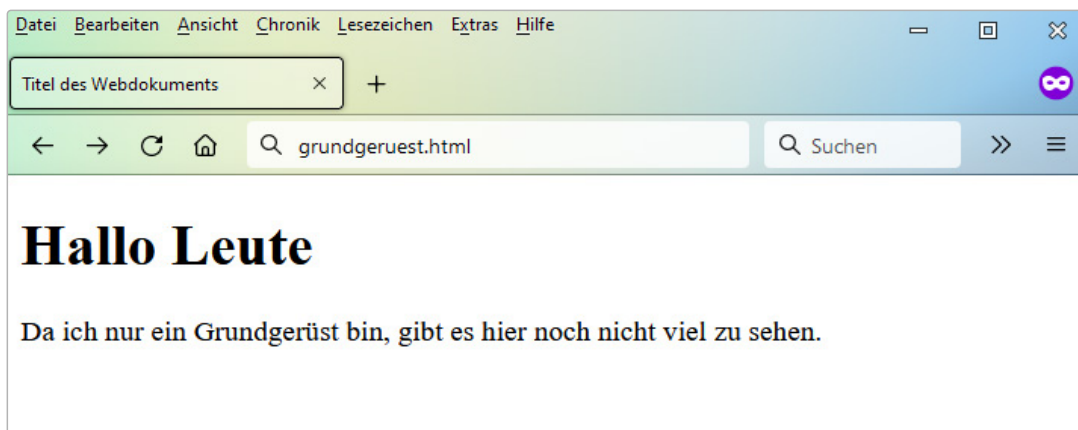
Öffne eine neue Datei in deinem Editor und füge folgende Code-Zeilen ein:


```

<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Titel des Webdokuments</title>
  </head>
  <body>
    <h1>Hallo Leute</h1>
    <p>Da ich nur ein Grundgerüst bin, gibt es hier noch
nicht viel zu sehen.</p>
  </body>
</html>

```

Speichere die Datei unter grundgeruest.html und öffne sie im Browser und Folgendes solltest du im Browser sehen:



Screenshot: Wie du siehst, siehst du nichts. 😊 Lediglich der Inhalt des `title`-Elements aus `<head>` wird als Überschrift im Browsertab eingeblendet und im Inhaltsbereich siehst du die Überschrift und den Absatz. Was mit den restlichen Zeichen passiert ist, wird in den folgenden Absätzen erklärt.

3.2.1 Welcher Dokumenttyp bin ich?

Schauen wir uns die einzelnen Zeilen im Grundgerüst genauer an. Mit der Angabe `<!DOCTYPE html>` signalisierst du allen Browsern »Achtung, es handelt sich hier um ein HTML-Dokument«. Damit wissen die Browser dann, wie sie mit diesem Dokument umzugehen haben. Dieses Dokument wird daher vom Browser interpretiert und nicht als Klartext angezeigt. Mit der Kurzvariante `<!DOCTYPE html>` weiß der Browser zudem, dass hier sowohl das aktuelle HTML 5 als auch die heute weniger üblichen Dialekte XHTML und HTML 4 vorkommen könnten. Wir werden in diesem E-Book die aktuelle HTML-Variante verwenden.

3.2.2 Die Landessprache bestimmen

Mit der Angabe `<html lang="de">` hast du dem gesamten Dokument das erste Element zugeteilt. Das `html`-Element umschließt das ganze HTML-Dokument und bildet die erste Ebene der Stammbaumstruktur. Das Attribut `lang="de"` darin signalisiert dem Browser, dass der Inhalt in deutscher Sprache ist.

3.2.3 Die unsichtbaren Angaben aus der Kopfzeile

Innerhalb von `<head>...</head>` siehst du diverse Meta-Angaben und Informationen über das Dokument. Die meisten dieser Angaben werden im Frontend⁸ nicht angezeigt.

Die erste Angabe aus dem `<head>...</head>` ist die Angabe zum verwendeten Zeichensatz. Mit `<meta charset="utf-8">` signalisiert du dem Browser, dass in diesem Dokument der [UTF-8-Zeichensatz](https://de.wikipedia.org/wiki/UTF-8)⁹ verwendet wird.

Dieser Zeichensatz beherrscht viel mehr Zeichen, als der früher oft eingesetzte ISO 8859-1. Somit musst du Zeichen wie Š, Ć, ©, ® oder ™ nicht mehr maskieren. Um etwa das © in ein HTML-Dokument einzutragen, musstest du dies früher maskieren und als `©` im HTML-Dokument notieren. Dank UTF-8 ist dieser Schritt nicht mehr notwendig.

Das `<meta name="viewport" content="width=device-width, initial-scale=1.0">` ist die zweite `meta`-Angabe in unserem Dokument. Das `meta`-Element beherrscht viele Attribute und wenn du dich weiterhin mit HTML beschäftigst, wirst du noch weitere kennenlernen.

Mit der `viewport`-Angabe signalisiert du dem Browser, dass die Breite so definiert ist, dass sie sich an die Breite des Bildschirms des jeweiligen Gerätes anpasst (`width=device-width`). Der zweite Wert `initial-scale` steht für den anfänglichen Zoomfaktor und sagt dem Browser, dass dieses HTML-Dokument im Verhältnis 1:1 auf dem Mobilgerät angezeigt werden soll.

3.2.4 Jetzt kommt der sichtbare Bereich

Bis jetzt hast du schon mehrere Angaben gemacht und diese waren im Frontend entweder gar nicht zu sehen oder wirkten sich nur indirekt aus. Mit `<body>...</body>` fängt der interessante Abschnitt an. Alles, was du innerhalb von den beiden `body`-Tags notierst, wird im Frontend angezeigt und ist somit für jede Person sichtbar, die das Dokument aufruft.

```
<h1>Hallo Leute</h1>
<p>Da ich nur ein Grundgerüst bin, gibt es hier noch nicht viel zu
sehen.</p>
```

⁸ Mit Frontend wird in der Regel der Bereich bezeichnet, den die Besucher auf deiner Website zu sehen bekommen.

⁹ <https://de.wikipedia.org/wiki/UTF-8>

Bei `<h1>` handelt es sich um eine Überschrift der ersten Ordnung. Diese sollte idealerweise nur einmal pro Seite erscheinen. Für Zwischenüberschriften bleiben dir dann fünf weitere Ordnungen von `<h2>` bis `<h6>`. Das `p`-Element ist ein Absatz und das wahrscheinlich am häufigsten genutzte Element in HTML-Dokumenten.

3.2.5 Häufig genutzte HTML-Elemente

In der folgenden Tabelle findest du eine Auflistung von häufig benötigten HTML-Elementen und einer Beschreibung, wie und wann sie eingesetzt werden:

Name (de + en)	Beispiel	Beschreibung
Überschriften (headline)	<code><h1>Ich bin die Hauptüberschrift!</h1></code> <code><h2>Überschrift 2. Ordnung</h2></code>	Es gibt sechs Überschriften (h1 bis h6). Die h1 sollte sinnvollerweise nur einmal pro Unterseite vorkommen. Die Überschriften sollten hierarchisch angeordnet sein. Nach h1 sollte ein h2 kommen und nicht h3 oder h4.
Absatz (paragraph)	<code><p>Ich bin ein Absatz</p></code>	Hier kommt der Fließtext. Absatz zählt zu den am häufigsten eingesetzten Elementen.
Auflistung (unordered list)	<code></code> <code>Butter</code> <code>Milch</code> <code>Obst</code> <code></code>	Eine Auflistung, also eine unsortierte Liste. Die einzelnen Einträge werden mit dem <code>li</code> -Element umgeben.
Aufzählung (ordered list)	<code></code> <code>Gold</code> <code>Silber</code> <code>Bronze</code> <code></code>	Eine Aufzählung, also eine nummerierte oder sortierte Liste.
Bilder (image)	<code></code>	Bild-Element. Du solltest immer darauf achten, dass die Bilder einen alt-Text haben. Das ist wichtig für Vorlesesoftware und Suchmaschinen.
Verweise (anchor)	<code>STRATO</code>	Verweise bzw. Links sind mit das Wichtigste in HTML-Dokumenten und machen das Wesen des WWW aus.
Fette Textformatierung	<code>Ich bin besonders wichtig!</code>	Für Wörter oder Wortgruppen, die für den Leser besonders wichtig sind.
Kursive Textformatierung (emphasis)	<code>Ich werde betont</code>	Möchtest du ein Wort betonen, es aber nicht fetten, kannst du es mit dem <code>em</code> -Element kursivieren.

Im späteren Verlauf des E-Books werde ich dir anhand eines konkreten Beispiels mehr über HTML erklären, in der Zwischenzeit schauen wir uns CSS an.

4. Was ist CSS und wofür ist es gut?

CSS (Cascading Style Sheets) ist eine Beschreibungssprache und dient als Ergänzung zu HTML. Bei einer gelungenen Umsetzung herrscht zwischen HTML und CSS eine strikte Arbeitsteilung. HTML kümmert sich darum, dem Inhalt eine **Bedeutung** und dem Dokument eine **logische Struktur** zuzuweisen. Das CSS ist für Formatierung und Darstellung der einzelnen HTML-Elemente und des gesamten Webdokuments im Browser zuständig.

In der Vergangenheit beschränkten sich die Fähigkeiten von CSS auf die Anpassung von wenigen Eigenschaften wie die Schriftgröße, Farben oder Positionierung von Elementen. Glücklicherweise wurde CSS kontinuierlich weiterentwickelt, sodass CSS mittlerweile richtige Raster-Layouts beherrscht und viele Funktionen bietet, die früher nur mit Photoshop & Co. oder durch umständliche Tricks möglich waren. Darunter fallen CSS-Verläufe, Schlagschatten, abgerundete Ecken, Animationen und vieles mehr. Einiges davon wirst du in diesem E-Book kennenlernen.

4.1 Das CSS einbinden

Um CSS einzubinden, stehen dir drei Wege zur Verfügung: inline, eingebettet und als externe Datei. Die ersten beiden Möglichkeiten solltest du allerdings selten anwenden, da es mit diesen beiden Varianten schnell unübersichtlich wird. Es ist vorteilhaft, alle CSS-Angaben zentral auszulagern.

Bei der Inline-Einbindung werden die CSS-Angaben innerhalb des jeweiligen HTML-Elements eingetragen, und dies als Attribut im öffnenden Tag:

```
<p style="color: navy;">Dieser Text wird in einem dunklen Blau  
angezeigt.</p>
```

Das eingebettete CSS wird innerhalb von `<head></head>` untergebracht, und zwar nach dem folgenden Muster:

```
<style type="text/css">  
.inhalt {color: maroon;}  
</style>
```

Die Variante, die am meisten angewendet wird, ist die Einbindung einer externen CSS-Datei, in der alle notwendigen CSS-Angaben **zentral** gelagert werden. Auch den Verweis zu der externen CSS-Datei bindet man innerhalb von `<head></head>` ein:

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

Damit unsere Beispiele im E-Book so einfach wie möglich bleiben, habe ich die Pfade vereinfacht und werde alle Dateien in den gleichen Ordner legen. Bei einem Projekt in der Praxis würden die Bilder etwa in einem `img`-Ordner und die CSS-Dateien in einem `css`-Ordner liegen. In so einem Fall würde das letzte Code-Fragment so ausschauen, wenn die Datei den Namen "layout.css" hat:

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

4.2 Der Aufbau von CSS: Regeln, Deklarationen, Eigenschaften & Werte

Das Vokabular einer Sprache zu beherrschen und die einzelnen Komponenten richtig zu benennen, ist wichtig. Es verhindert Missverständnisse und hilft dir bei der späteren Arbeit und bei der Recherche, wenn du an der ein oder anderen Stelle nicht vorwärtskommst.

Schauen wir uns die CSS-Angaben an und benennen die einzelnen Bestandteile. Wenn du einen Bereich formatieren oder stylen möchtest, notierst du die Angaben in einer **CSS-Regel**. Hier der schematische Aufbau einer CSS-Regel:

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

Die CSS-Regel besteht aus mehreren Bestandteilen. Am Anfang steht der Selektor. Man könnte ihn auch den Empfänger der Anweisung nennen. Innerhalb der geschweiften Klammern notierst du die **Eigenschaft**, die angepasst werden soll und dahinter den **Wert**. Die Zeile, die aus der Eigenschaft und dem dazugehörigen Wert besteht, bezeichnet man als **CSS-Deklaration**. Oben wäre das die Zeile 2.

Nehmen wir mal an, du möchtest einem Abschnitt die Breite ("width") von 600 Pixeln zuweisen, dann würde die entsprechende CSS-Regel so ausschauen:

```
div {  
  width: 600px;  
}
```

Bei dem Beispiel habe ich mich für einen Element-Selektor entschieden. Damit werden alle `div`-Elemente innerhalb des Dokuments angesprochen. Die CSS-Deklaration besteht aus der Eigenschaft `width` und dem Wert 600 Pixel.

Der Element-Selektor ist nicht der einzige. In CSS kannst du aus einer Reihe von Selektoren wählen, um so zielgerichtet wie möglich deine Anweisungen unterzubringen.

4.2.1 Die CSS-Selektoren

CSS kennt mehrere Dutzend Selektoren. Auf alle einzugehen, wäre zum einen nicht zielführend, da viele nur selten verwendet werden und würde zum anderen den Rahmen des E-Books sprengen. Am Anfang wirst du nur ein paar wenige Selektoren benötigen. Daher werde ich mich hier auf die Selektoren konzentrieren, die erfahrungsgemäß häufig verwendet werden.

Selektor	Beispiel	Beschreibung
Element-Selektor	<code>p {...}</code>	Diese Regel gilt für alle Vorkommen des jeweiligen Elements.
Klassen-Selektor	HTML: <code><p class="hinweis">...</p></code> CSS: <code>.hinweis {...}</code> <code>p.hinweis {...}</code>	Diese Regel nutzt das Klassen-Attribut im HTML-Element und greift auf alle Elemente zu, die den entsprechenden Attributwert haben. Erfahrungsgemäß wird dieser Selektor am häufigsten eingesetzt.
ID-Selektor	<code><nav id="navi">...</nav></code> <code>#navi {...}</code>	Das ID-Attribut eines Elements kannst du auch als Selektor einsetzen: Diesen Selektor solltest du sparsam einsetzen, da er eine hohe Spezifität hat und ein paar Eigenheiten aufweist. Auf beides werde ich später eingehen.
Nachfahre-Selektor	<code>.inhalt p {...}</code> <code>.abschnitt .inhalt p {...}</code>	Der Nachfahre-Selektor spricht alle p-Elemente an, die sich innerhalb des Elternelements mit dem Klassen-Attribut <code>inhalt</code> befinden.
Kind-Selektor	<code>.inhalt > p</code>	Der Kind-Selektor wirkt ähnlich dem Nachfahre-Selektor, besitzt allerdings etwas mehr Feintuning, er spricht nur die p-Elemente an, die unmittelbar innerhalb des Elternelements mit der entsprechenden Klasse vorkommen. p-Elemente, die tiefer verschachtelt folgen, werden ignoriert.
Pseudoklassen	<code>:hover</code> <code>:visited</code> <code>:focus</code> <code>:first-child</code>	Mit Pseudoklassen kannst du ein HTML-Element adressieren, wenn es eine bestimmte Eigenschaft besitzt. Mit der Pseudoklasse <code>:hover</code> sprichst du Elemente an, die von dem Mauscursor berührt werden und mit <code>:first-child</code> sprichst du das erste Kind-Element eines Abschnitts an.
Pseudoelemente	<code>:before</code> <code>:after</code>	Selektoren sind üblicherweise auf bestehende HTML-Elemente im Dokumentenbaum beschränkt. Eine Ausnahme bilden die Pseudoelemente. Sie erzeugen neue Elemente, die nicht vorhanden sind. Bekannteste Vertreter dieser Gattung sind <code>:before</code> und <code>:after</code> . Damit kannst du in CSS formatierbare Elemente einfügen, die vor/nach einem tatsächlich existierenden Element angezeigt werden.
Universal-Selektor	<code>* {...}</code>	Mit dem Universal-Selektor weist du allen Elementen eine Eigenschaft zu.

Die [Auflistung aller Selektoren](#)¹⁰ findest du unter anderem in der deutschsprachigen Dokumentation »SelfHTML«.

4.4 Die Kaskade

Wenn du ein HTML-Dokument erstellst, das keine CSS-Anweisungen beinhaltet, wirst du feststellen, dass diverse Abschnitte und Teile der Website dennoch formatiert ausgegeben werden: die Überschriften unterscheiden sich in der Größe, Absätze weisen Abstände auf, Auflistungen verfügen über Listenzeichen, Verweise sind blau und so weiter.

Der Grund: Die Browser bringen ein eigenes Basis-CSS mit, in dem alle Elemente grundlegende Formatierungen enthalten. Dass du mit diesen Browser-Formatierungen keinen Schönheitspreis gewinnen wirst, dürfte klar sein. Allerdings sind die Inhalte leserlich und barrierefrei.

Das Wort Kaskade kommt auch in der Bezeichnung der Sprache vor: Cascading Style Sheets. Kaskade kommt aus dem Französischen (cascade) und bedeutet: ein Wasserfall, der über mehrere Stufen geht. Der Begriff »Kaskadierung« bezeichnet in der Elektrotechnik das Hintereinanderschalten mehrerer Module, wobei sich dadurch deren Wirkung [verstärkt](#)¹¹.

Im Zusammenhang mit CSS ist dies von **doppelter** Bedeutung. Zum einen können HTML-Dokumente durch mehrere CSS-Dateien formatiert werden, wobei diese aufeinander aufbauen können. Zum anderen können die Selektoren »hintereinander geschaltet« werden und somit an Gewicht gewinnen.

Ein Bereich deines HTML-Dokumentes könnte zum Beispiel so aussehen:

```
<div id="abschnitt">
<div class="hinweis">
  <p class="absatz">Ein Beispieltext mit
  <strong>hervorgehobenem</strong> Text.</p>
</div>
</div>
```

¹⁰ <https://wiki.selfhtml.org/wiki/CSS/Selektoren>

¹¹ <https://de.wikipedia.org/wiki/Kaskadierung>

Du hast jetzt dank CSS mehrere Möglichkeiten, den Absatz innerhalb des oberen HTML-Konstrukts anzusprechen:

1. `p {color: green;}`
2. `p.absatz {color: grey;}`
3. `.hinweis .absatz {color: navy;}`
4. `.abschnitt .hinweis .absatz {color: maroon;}`

Alle vier CSS-Regeln sprechen das gleiche Ziel an. Die erste Regel sagt: »Dieser Text soll grüne Schriftfarbe haben«. Die zweite Regel möchte, dass der Text in Grau ausgegeben wird. Dritte Regel wünscht sich eine dunkelblaue Farbe für den Text und die letzte Regel möchte, dass die Textfarbe im Absatz dunkelrot (maroon) wird. Und was meinst du? Welche Farbe wird der Text im Absatz haben?

Richtig. Es wird **dunkelrot** sein, da die Angabe viel **spezifischer** und somit stärker ist.

Grundsätzlich gilt, je mehr Selektoren, desto mehr Gewicht hat die Anweisung. Der Selektor `p` allein wäre sehr unspezifisch und würde **alle** Absätze in einem HTML-Dokument betreffen. Je spezifischer die Anweisung, desto mehr Gewicht hat sie.

Der Selektor `p.absatz` überschreibt also `p`, wird selbst aber von `.hinweis .absatz` und dieser wiederum von `.abschnitt .hinweis .absatz` überschrieben. Diesen Effekt nennt man **Spezifität**. Je spezifischer ein Selektor ist, umso stärker ist er.

Hinweis:

Bitte achte hierbei auf die richtige Schreibweise. Oben habe ich `p.absatz` als Selektor benutzt. In so einem Fall kommt kein Leerzeichen vor den Punkt. Wo liegt der Unterschied? Das `p.absatz` steht für alle Absätze mit der Klasse bzw. dem Attributwert `absatz`: `<p class="absatz">`.

Ein Leerzeichen in `p .absatz` würde den Browser anweisen, nach allen Vorkommen der Klasse `absatz` innerhalb von Absätzen zu suchen, zum Beispiel: `<p>`. Ein gewaltiger Unterschied. Das Leerzeichen in diesem Zusammenhang bedeutet **immer**, dass es sich um einen Nachfahre-Selektor handelt.

Die Reihenfolge der Kaskade

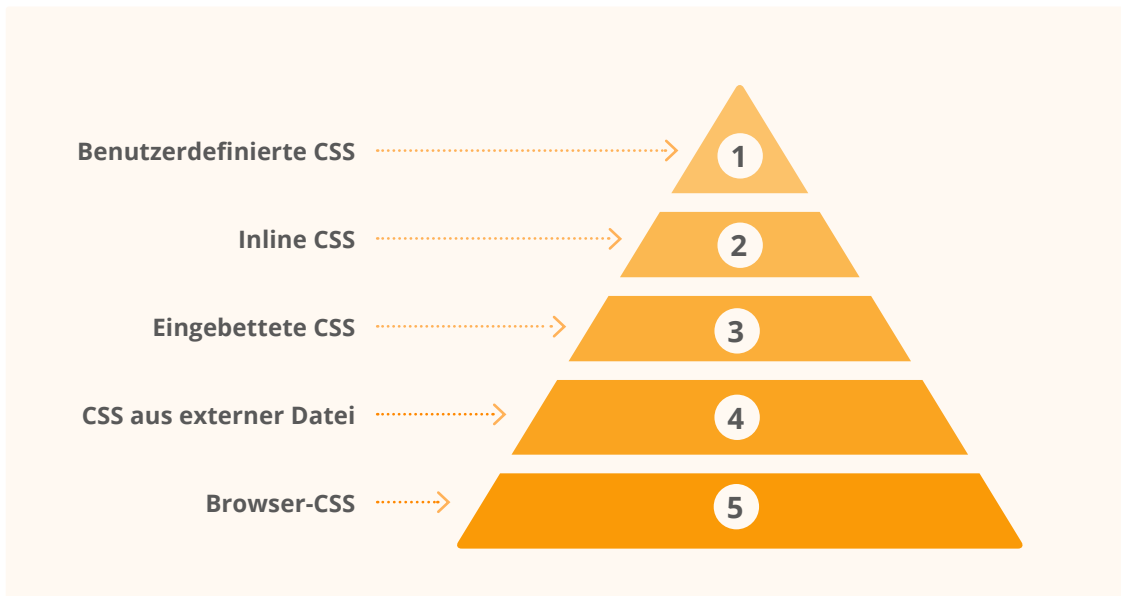


Abbildung: Grobe Aufteilung der CSS-Kaskade

4.4.1 Kaskade 1: Die Herkunft der CSS-Regel

Die Kaskade in CSS wirkt auf mehreren Ebenen. Zum einen gibt es eine grobe Aufteilung, wie du sie auch in der oberen Grafik sehen kannst. Am unteren Ende ist das CSS, welches vom Browser mitgeliefert wird.

Dieses wird von den Angaben aus einer externen CSS-Datei überschrieben. Dies wird dann von den im Dokumentkopf eingebetteten CSS-Regeln überschrieben. Das Inline-CSS überschreibt wiederum das eingebettete und ganz oben kommt das benutzerdefinierte CSS, welches den höchsten Rang hat und daher an der Spitze der Pyramide steht.

Das benutzerdefinierte CSS nutzt du, wenn du etwa mit diversen Browser-Erweiterungen deine eigenen CSS-Regeln eintragen möchtest, die dann bei bestimmten Websites zum Einsatz kommen.

Eine solche Browser-Erweiterung ist »Stylus«, die es für [Firefox](#)¹², [Chrome](#)¹³ und [Opera](#)¹⁴ gibt. Gründe für das Eintragen von benutzerdefinierten Stylesheets gibt es viele. Das kann eine unleserliche Schrift auf einer Website sein, die der Nutzer im eigenen Browser gerne ersetzen möchte oder ein zu schmaler oder breiter Inhaltsbereich. Helfen kann es auch,

um das [WordPress-Backend übersichtlicher zu gestalten](#)¹⁵ oder um die nervigen [Trends aus der Twitter-Sidebar zu entfernen](#)¹⁶.

4.4.2 Kaskade 2: Position vs. Spezifität vs. Selektor-Wert

Auf der zweiten Ebene ist die Auswirkung der Kaskade viel feiner und sie wirkt innerhalb der bereits vorgestellten Blöcke aus der Kaskaden-Pyramide. Eine Anweisung innerhalb einer externen CSS-Datei, die weiter unten folgt, überschreibt die Regel vom Anfang des Dokuments. Hier ein Beispiel. Wenn man die beiden folgenden Regeln so in einer CSS-Datei notiert, stellt sich die Frage, welche Regel zum Einsatz kommen wird?

```
.hinweisbox {color: blue;}
.hinweisbox {color: red;}
```

Hier wird in der Hinweisbox nicht die blaue, sondern die rote Schriftfarbe zum Einsatz kommen. Sind die Regeln von **gleicher** Spezifität, dann ist die Lage im Dokument ausschlaggebend: die CSS-Regeln, die **später** im Dokument auftauchen, werden berücksichtigt. Erweitern wir das letzte Beispiel um eine Kleinigkeit:

```
.inhalt .hinweisbox {color: blue;}
.hinweisbox {color: red;}
```

In diesem Beispiel wird die erste CSS-Regel zum Einsatz kommen und in der Hinweisbox wird es eine blaue Schriftfarbe geben. Ja, die zweite Regel kommt zwar später im Dokument, aber die erste CSS-Regel ist spezifischer und die Spezifität siegt über die Position.

Das ist aber nicht alles. Neben der Position im Dokument und der Anzahl der Ketten im Selektor kommt auch die Stärke einzelner Selektor-Arten zum Tragen. Ein Element-Selektor ist recht schwach und wird von einem Klassen-Selektor überschrieben, da dieser mehr Kraft hat. Dagegen hat der Klassen-Selektor keine Chancen gegen den ID-Selektor, der viel mächtiger ist. Im folgenden Beispiel kannst du das genau beobachten:

```
#inhalt p {color: red;}
.abschnitt .bereich1 .bereich2 .bereich3 p {color: black;}
```

Welche Regel wird zum Einsatz kommen? Die zuletzt im Dokument erscheint, diejenige, die spezifischer ist, oder etwas Anderes? Die erste Regel wird zum Einsatz kommen. Obwohl die zweite Regel die Position im Dokument und fünf anstatt zwei Selektoren auf die Waagschale legen kann, wird sie dennoch gegen den ID-Selektor verlieren.

¹² <https://addons.mozilla.org/de/firefox/addon/styl-us/>

¹³ <https://chrome.google.com/webstore/detail/stylus/clngdbkpkpeebahjckkfobafhncgmne>

¹⁴ <https://addons.opera.com/de/extensions/details/stylus/>

¹⁵ <https://www.perun.net/2020/10/19/wordpress-beitraege-in-der-admin-uebersicht-farblich-hervorheben/>

¹⁶ <https://www.perun.net/2022/09/28/bessere-laune-durch-schlankeres-twitter/>

Der [spezifische Wert](#)¹⁷ des ID-Selektors wird mit 100, der Wert des Klassen-Selektors mit 10 und der Wert des Element-Selektors wird mit 1 angegeben. Somit hat im oberen Beispiel der erste Selektor einen Wert von 101 und der zweite Selektor den Wert von 41. Deswegen würde ich dir empfehlen, den ID-Selektor nur sehr sparsam einzusetzen, weil die Arbeit im CSS sonst komplizierter wird.

Noch extremer ist die Auswirkung bei dem Einsatz von `!important` bei den Deklarationen. Im folgenden Beispiel ...

```
.abschnitt p {color: blue !important;}
#inhalt p {color: red;}
.abschnitt .bereich1 .bereich2 .bereich3 p {color: black;}
```

... würde die erste Regel zum Einsatz kommen. Das `!important` kannst du mit dem 10-kg-Vorschlaghammer auf dem Bau, mit dem Royal Flush im Poker, mit zwei Damen nach Umwandlung im Schach oder mit dem Notfallhammer im Bus vergleichen, je nachdem welchen Vergleich du sympathischer findest. Nutze das `!important` nur in seltenen Ausnahmefällen, da es die Kaskade **komplett** durcheinander bringt und somit zu Problemen bei der Wartung von CSS führen kann.

4.5 Das Boxmodell

Alle Elemente eines HTML-Dokuments, die durch CSS adressiert werden, haben eine Box um sich herum. Meist ist diese unsichtbar, manchmal aber auch mit einem Rahmen oder einer separaten Hintergrundfarbe oder Ähnlichem hervorgehoben.

Diese Box besteht immer aus dem Inhaltsbereich (engl. Content), in dem sich Text, Bild oder andere Inhaltsarten befinden. Um den Inhalt herum liegen drei weitere Bereiche und sie gliedern sich folgendermaßen:

1. Inhalt (Content)
2. Innenabstand (Padding)
3. Rahmen (Border)
4. Außenabstand (Margin)

Damit du später keine Probleme hast, ist es essenziell zu verstehen: Die Maße, die ein Element hat, beinhalten alle diese Bereiche. Gibst du für ein Element eine Höhe oder Breite an, sowie Innen- und Außenabstände und noch einen Rahmen, wird die reale oder dargestellte Größe die angegebene Höhe und Breite übersteigen.

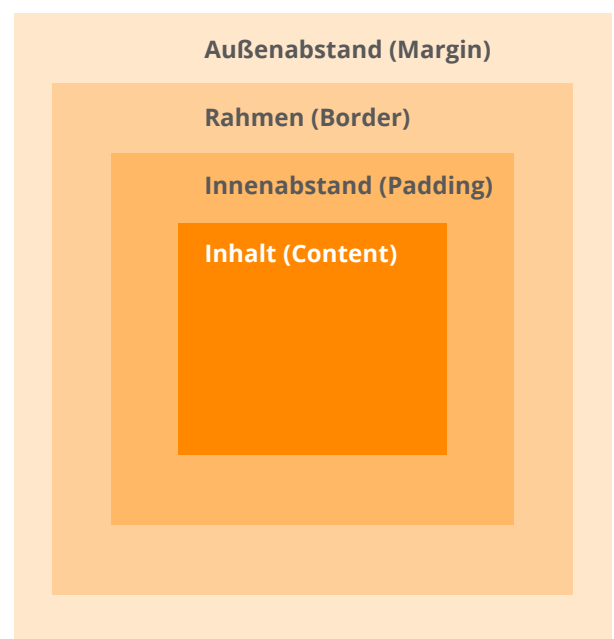


Abbildung: Die einzelnen Bereiche im Box-Modell

¹⁷ <https://www.w3.org/TR/CSS21/cascade.html#specificity>

Alles verstanden? Falls ja, super. Wenn nicht, kein Problem, hier ein Beispiel, das die Auswirkungen verdeutlicht.

```
.box {
width: 450px;
height: 250px;
padding: 25px;
border: 5px solid black;
margin: 15px;
}
```

Schauen wir uns gemeinsam diese CSS-Regel genauer an. Ich habe eine Box definiert und dem eigentlichen Inhaltsbereich habe ich eine Breite (**width**) von 450 und eine Höhe (**height**) von 250 Pixel gegeben. Der innere Abstand (**padding**) hat auf allen vier Seiten 25 Pixel. Der Rahmen (**border**) ist auf allen Seiten fünf Pixel dick und hat eine schwarze Farbe (**solid black**). Der äußere Abstand (**margin**) ist auf allen vier Seiten 15 Pixel groß.

Jetzt stellt sich die Frage: Wie breit ist die Box tatsächlich? Die gleiche Frage stellt sich für die Höhe. Um solche und ähnliche Fragen zu beantworten, ist es immer ratsam, die Developer- oder Entwickler-Tools im Browser zu konsultieren. Diese Tools öffnest du in den gängigen Browsern auf Windows, in dem du die Taste **F12** betätigst. Bei Firefox auf Windows funktioniert noch **Strg + ↑ + i**. Wenn du die Box in den **Web-Entwickler-Tools** untersuchst, würdest du in etwa folgendes Bild sehen:

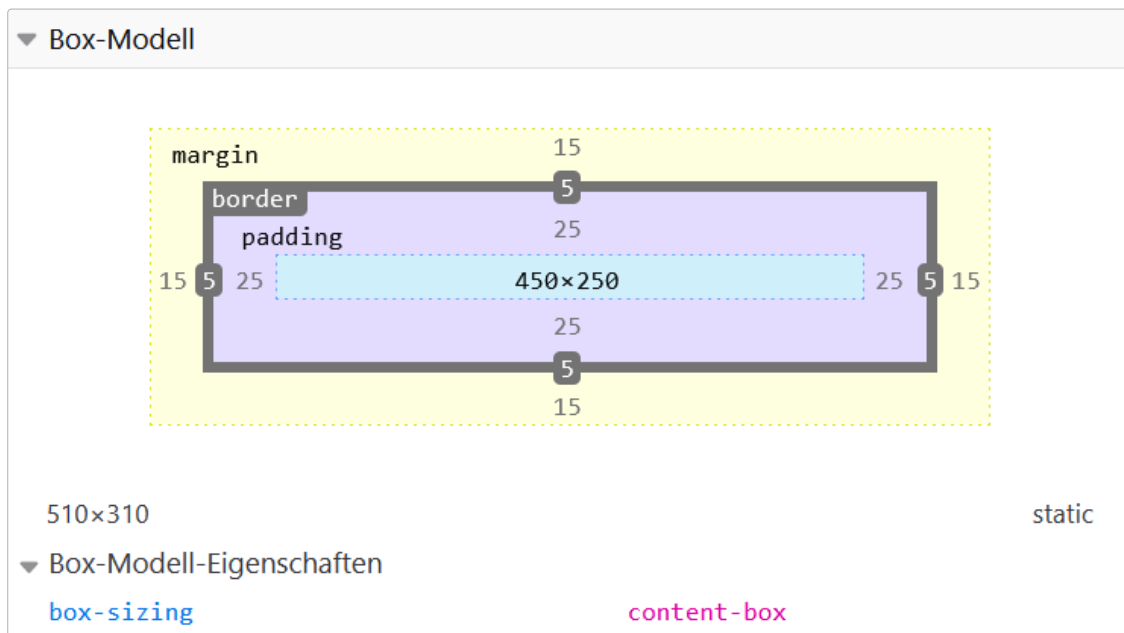


Abbildung: Die Auswirkungen des Standard-Boxmodells.

Die Antwort ist: Die tatsächliche Breite der Box ist nicht 450 Pixel, sondern **510 Pixel**. Wie kann das sein? Die wirkliche Breite der Box setzt sich zusammen aus der Breitenangabe von 450 Pixeln sowie den Angaben für den inneren Abstand und den Rahmen. Der äußere Abstand (**margin**) wird nicht dazu addiert – das erklärt in unserem Beispiel den Wert von 510 Pixel – hat aber Auswirkung auf die umliegenden Elemente und Inhalte. Analog gilt das Gleiche für tatsächliche Höhe. Damit ist die so dargestellte Box immer größer als die tatsächlichen Angaben.

Das Standard-Boxmodell agiert nach dem gleichen Prinzip, wie wir es auch in manchen Fällen aus dem Alltag kennen. Ein Gerät, welches 40 × 20 × 25 Zentimeter groß ist, ist der Inhalt. Wenn du diesen Inhalt per Post verschicken möchtest, dann ist das Paket größer. Es kommt die Dicke bzw. die Stärke des Kartons (entspricht **border**) und der Platz zwischen dem Inhalt und dem Kartonrand (entspricht **padding**), der mit Füllmaterial aufgefüllt wird dazu.

Da das Verhalten im Standard-Boxmodell häufig zu Verwirrungen geführt hat, hat man ein alternatives Boxmodell eingeführt, welches sich an den tatsächlichen Maßen orientiert. Wenn wir unsere CSS-Regel um eine zusätzliche Zeile bzw. Deklaration erweitern, können wir das Problem umgehen:

```
.box {  
  box-sizing: border-box;  
  width: 450px;  
  height: 250px;  
  padding: 25px;  
  border: 5px solid black;  
  margin: 15px;  
}
```

Durch die neue Deklaration in der ersten Zeile des Beispiels haben wir den Browser angewiesen, die wahren Ausmaße der Box nach einer anderen Regel zu interpretieren. Wenn du eine exakte Größe für die Breite und Höhe angibst, dann wird nach dem alternativen Modell, der Browser die entsprechende Box mit diesen Maßen als Maximalgröße darstellen. Um auf unser Beispiel aus dem zurückzukehren: Die Werte im alternativen Boxmodell beziehen sich **nicht** mehr auf den Inhalt des Pakets, **sondern** auf die Gesamtgröße des Pakets.

Das heißt, wenn du 450 Pixel Breite angibst wird die Box mit dem eigentlichen Inhalt + dem inneren Abstand + dem Rahmen zusammen **nicht** die 450 Pixel überschreiten.

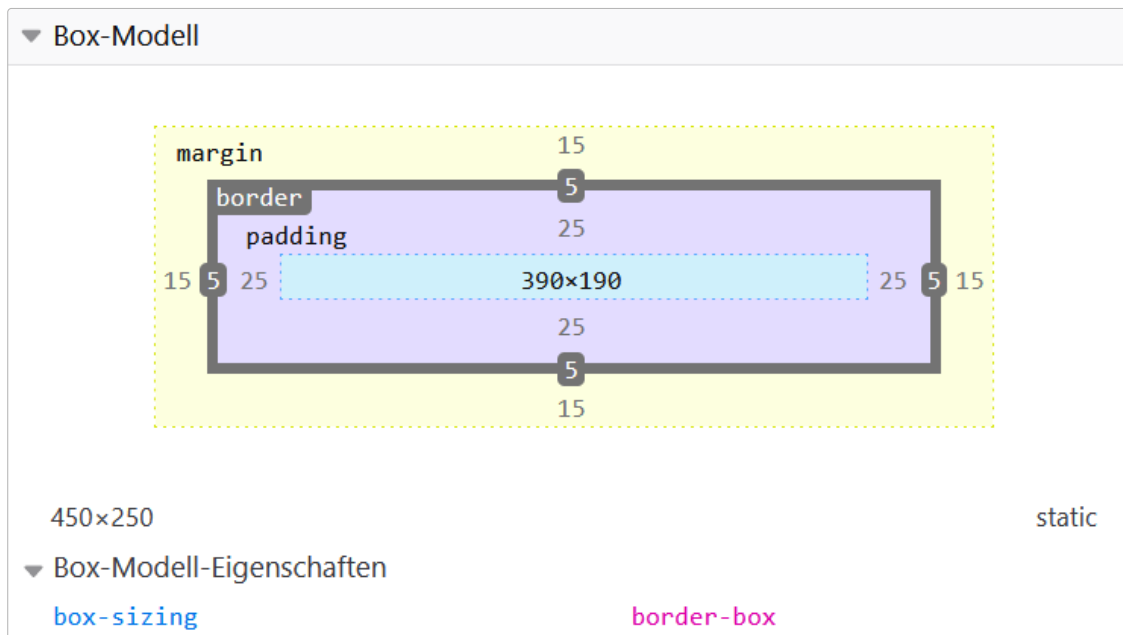


Abbildung: Dank des Wertes `border-box` bleiben 450×250 Pixel auch 450×250 Pixel. Der äußere Abstand zählt, nach wie vor, nicht zu den Angaben der Box, beeinflusst aber den außenstehenden Inhalt.

Damit du diese Deklaration nicht für jede Box angeben musst, kannst du bei deinen Projekten einfach am Anfang der CSS-Datei Folgendes einfügen und dann wird das alternative Box-Modell auf das komplette Projekt angewendet:

```
html { box-sizing: border-box;}
*, *::before, *::after {box-sizing: inherit;}
```

Falls du aus welchen Gründen auch immer, mit dem Standard-Box-Modell besser zurechtkommst, bist du nicht gezwungen umzusteigen und kannst diesen Schritt überspringen.

4.5.1 Block- und Inline-Elemente

Wenn es darum geht, wie sich die HTML-Elemente innerhalb der Fläche ausbreiten, wird nach zwei Gruppen unterschieden. Zum einen gibt es Block-Elemente. Dazu zählen etwa Überschriften, Absätze, Listen oder Zitate. Diese Elemente breiten sich standardmäßig über die komplette Breite aus und erzeugen eine neue Zeile. Die Inline-Elemente dagegen erzeugen keine neue Zeile und konzentrieren sich nur auf die unmittelbare Umgebung. Beispiele für Inline-Elemente sind etwa die ganzen Textformatierungen und die Links.

5. Jetzt geht's los!

Die Beispielwebsite

Glückwunsch, du hast die Theorie überstanden und nun werden wir konkret. Wir haben bis jetzt zwar noch nicht alle Grundlagen besprochen, aber mit einem kleinen Beispielprojekt kannst du zum einen, die bisherigen Kenntnisse besser vertiefen und zum anderen, die folgenden Grundlagen einfacher verstehen.

Stellen wir uns dafür einen freiberuflichen Autor vor, der im Web eine kleine Website einstellen möchte, um für potenzielle Auftraggeber grundlegende Informationen zu präsentieren: Leistungen, Arbeitsproben, Kontaktdaten. Häufig ist eine solche Website auch unter dem Begriff **Web-Visitenkarte** bekannt.

5.1 Der HTML-Code der Beispielseite

Zuerst kommt der Block mit dem gesamten HTML-Code:

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Alexios Barnabas - freier Autor</title>
    <meta name="description" content="Alexios Barnabas ist
freier Autor mit langjähriger Erfahrung.">
    <link rel="stylesheet" type="text/css" href="style.
css">
  </head>
  <body>
    <header>
      <div class="innen">
        <div class="brand">
          <a href="index.html"></a>
          <div class="slogan">
            <h1><a href="index.html">Alexios
Barnabas</a></h1>
            <p>Freier Autor</p>
          </div>
        </div>
      </div>
      <nav class="navi">
        <ul>
          <li><a href="#ueber-mich">Über mich</a></li>
```

```

        <li><a href="#publikationen">Publikationen</
a></li>
        <li><a href="#kontakt">Kontakt</a></li>
    </ul>
</nav>
</div>
</header>
<section id="ueber-mich">
    <div class="innen">
        <h2>Über mich</h2>
        <p>Mein Name ist Alexios Barnabas und ich arbeite seit
25 Jahren als freier Autor. In dieser Zeit habe ich mit namhaften
Verlagen zusammengearbeitet und eine Reihe von Fachbüchern,
Artikeln in Fachzeitschriften und andere Veröffentlichungen
publiziert. Folgende Dienstleistungen biete ich an:</p>
        <ul>
            <li>Bücher und Buchbeiträge</li>
            <li>Artikel für Fachzeitschriften</li>
            <li>Blogartikel und andere Webtexte</li>
            <li>Beratung, Schulungen & Ghostwriting</li>
        </ul>
    </div>
</section>
<section id="publikationen">
    <div class="innen">
        <h2>Aktuelle Publikationen</h2>
        <div class="portfolio">
            <div class="portfolio1">
                
                <h3>WordPress auf C-64</h3>
                <p>Mein neues Buch, in dem ich ausführlich
beschreibe, wie du erfolgreich WordPress auf einem Commodore-Server
betreiben kannst.</p>
            </div>
            <div class="portfolio2">
                
                <h3>Programmieren mit Amiga</h3>
                <p>Ein Artikel für die Fachzeitschrift
"Commodore & Web", in dem ich ausführlich beschreibe, wie du einen
Cloud-Cluster mit einem Amiga 500 betreiben kannst.</p>
            </div>
        </div>
    </div>
</section>

```



```

        <div class="portfolio3">
            

            <h3>Mit BASIC zwitschern</h3>
            <p>Ein Beitrag für die "Cologne Times",
wo ich Schritt für Schritt erkläre, wie du mit BASIC einen Twitter-
Klon programmieren kannst.</p>
        </div>
    </div>
</div>
</section>

<section id="kontakt">
    <div class="innen">
        <h2>Kontakt</h2>

        <address>
            Alexios Barnabas<br>
            Paul-Attreides-Allee 1<br>
            98765 Arrakis<br>
            Dune<br><br>
            0123 456 78 90<br>
            kontakt@spice-post.abc
        </address>
    </div>
</section>

<footer>
    <div class="innen">
        <p>© 2022 Alexios Barnabas</p>
        <p><a href="impressum.html">Impressum</a> und <a
href="datenschutz.html">Datenschutz</a></p>
    </div>
</footer>
</body>
</html>

```

Übertrage den oberen HTML-Code in den Editor und speichere die Datei als index.html ab. Alle Code-Beispiele findest du übrigens auch in diesem [Download-Paket](https://my.hidrive.com/share/c895m0-6qq#$/)¹⁸.

¹⁸ [https://my.hidrive.com/share/c895m0-6qq#\\$/](https://my.hidrive.com/share/c895m0-6qq#$/)

Wenn du die Datei im Browser aufrufst, wirst du in etwa folgendes Bild zu sehen bekommen:

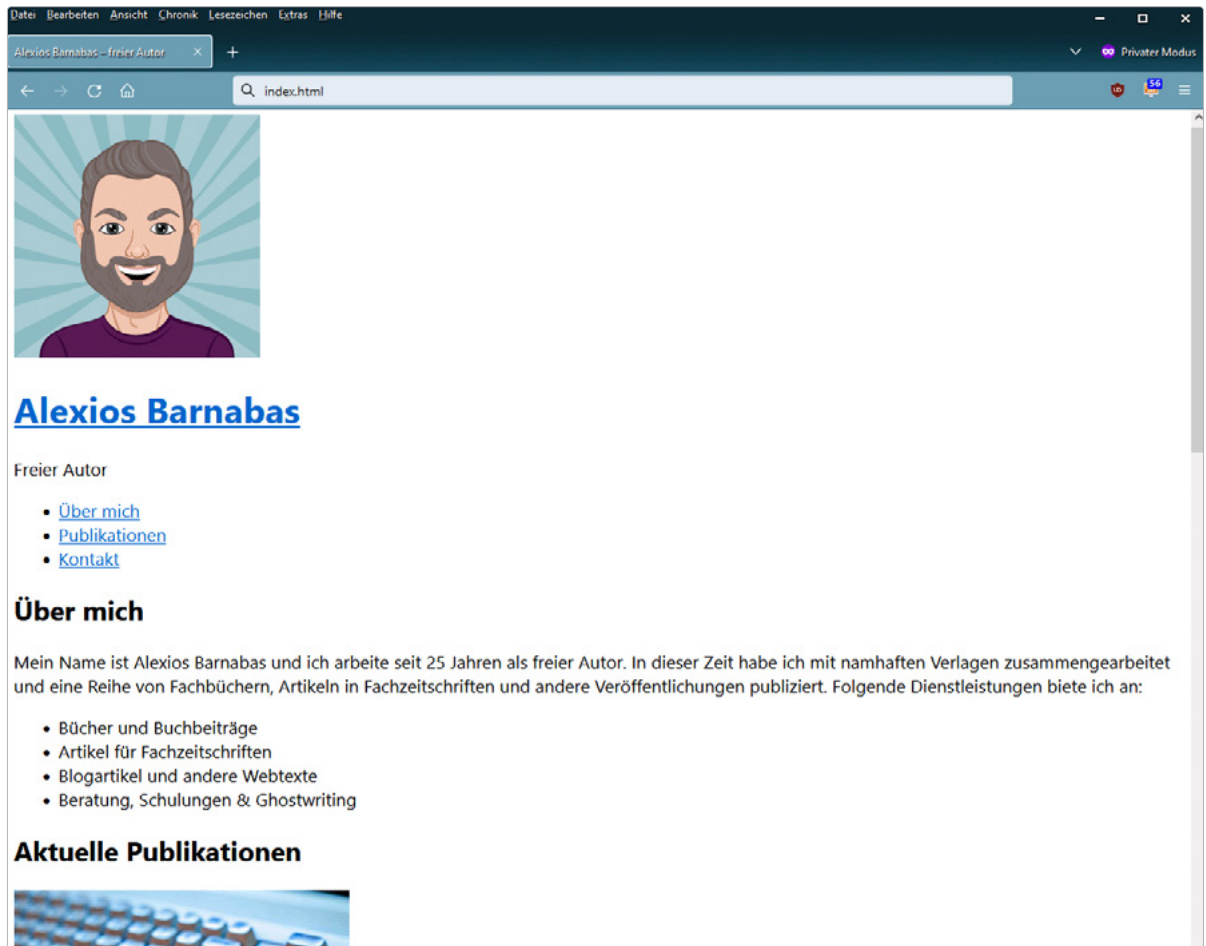


Abbildung: Screenshot der oberen Hälfte der Beispielseite.

Wie du in der Abbildung gut erkennen kannst ... hübsch ist anders. So schauen nun mal Websites ohne Einsatz von CSS aus. Funktional sind sie nicht nur **völlig** in Ordnung, sondern auch noch **barrierefrei**. Aber einen Design-Blumentopf gewinnst du damit nicht.

Dennoch ist dir aufgefallen, dass bestimmte Abschnitte Formatierungen besitzen. Überschriften haben unterschiedliche Schriftgrößen, Links sind blau und unterstrichen und Absätze haben Abstände. Im Abschnitt »[Die Kaskade](#)« habe ich bereits erwähnt, dass die Browser eigene CSS-Angaben mitbringen. Damit ist gewährleistet, dass auch »nackte« HTML-Seiten gut in Browsern darstellbar sind.

5.2 Der HTML-Code erklärt

Jetzt schauen wir uns gemeinsam die einzelnen Abschnitte des HTML-Codes genauer an. Einiges dürfte dir aus dem [Grundgerüst-Abschnitt](#) bereits bekannt vorkommen, daher werde ich die bekannten Angaben nur kurz erwähnen und lediglich neue Angaben ausführlicher erklären.

5.2.1 Wichtige Angaben für die Suchmaschinen innerhalb von <head>

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <title>Alexios Barnabas – freier Autor</title>
  <meta name="description" content="Alexios Barnabas ist
freier Autor mit langjähriger Erfahrung.">
  <link rel="stylesheet" type="text/css" href="style.css">
</head>
```

In der ersten Zeile siehst du die Dokumenttyp-Deklaration (!DOCTYPE), darunter folgt das öffnende `html`-Tag mit der Angabe zur eingesetzten Sprache im Inhalt. Darunter kommt die `meta`-Angabe zum Zeichensatz ("charset") und in der Zeile fünf weisen wir den Browser an, wie er den Inhalt auf verschiedenen Bildschirmgrößen skalieren soll ("viewport").

In der sechsten Zeile siehst du den Seitentitel ("title"). Der Inhalt dieses Elements ist in der Titelleiste des Browsers zu sehen, wenn du dich auf der Seite befindest und ist gleichzeitig auch zu sehen, wenn du die entsprechende Seite in die Lesezeichensammlung des Browsers aufnimmst.

In der siebten Zeile siehst du eine neue `meta`-Angabe. Der Inhalt von `description` wird von den Suchmaschinen als mögliche Quelle für den Beschreibungstext genommen. In der folgenden Abbildung siehst du an einem Suchergebnis, wie der Inhalt von `title` und `description` ausgegeben werden:



Abbildung: Der Aufbau eines Suchergebnisses am Beispiel von Google. Der Inhalt des `title`-Elements wird als Überschrift prominent dargestellt und darunter folgt die `meta-description`.

Bitte achte darauf, dass sowohl der Titel als auch die Beschreibung gut den Inhalt wiedergeben und für die Besucher hilfreich sind. Beides sind Angebote für die Suchmaschinen. In der Regel halten sich diese auch an die Vorgaben, aber speziell Google behält sich vor, sowohl eine alternative Beschreibung als auch einen alternativen Seitentitel einzublenden. Dies geschieht insbesondere, wenn der Algorithmus davon ausgeht, dass es zum Inhalt beziehungsweise zur Suchintention besser passt.

5.2.2 Der Inhalt der Beispielseite

Nun geht es weiter zum sichtbaren Bereich. Hier wirst du einige neue Elemente entdecken:

```
<body>
<header>
  <div class="innen">
    <div class="brand">
      <a href="index.html"></a>
      <div class="slogan">
        <h1><a href="index.html">Alexios
Barnabas</a></h1>
        <p>Freier Autor</p>
      </div>
    </div>

    <nav class="navi">
      <ul>
        <li><a href="#ueber-mich">Über mich</a></li>
        <li><a href="#publikationen">Publikationen</
a></li>
        <li><a href="#kontakt">Kontakt</a></li>
      </ul>
    </nav>
  </div>
</header>
```

In diesem Abschnitt ist dir vielleicht das `header`-Element aufgefallen. Für Leute, die frisch mit HTML anfangen, dürfte dieses HTML-Element wie jedes andere sein. Für diejenigen aber, die sich vor vielen Jahren mit HTML beschäftigt und dann eine Pause gemacht haben, dürfte dieses Element neu sein.

Mit Elementen wie `<header>`, `<footer>`, `<section>`, `<nav>`, `<aside>` und `<article>` wurden zusätzliche Möglichkeiten eingefügt, nicht nur einzelnen Elementen eine Bedeutung (Semantik) zu geben, sondern auch ganzen Bereichen innerhalb des Dokuments.

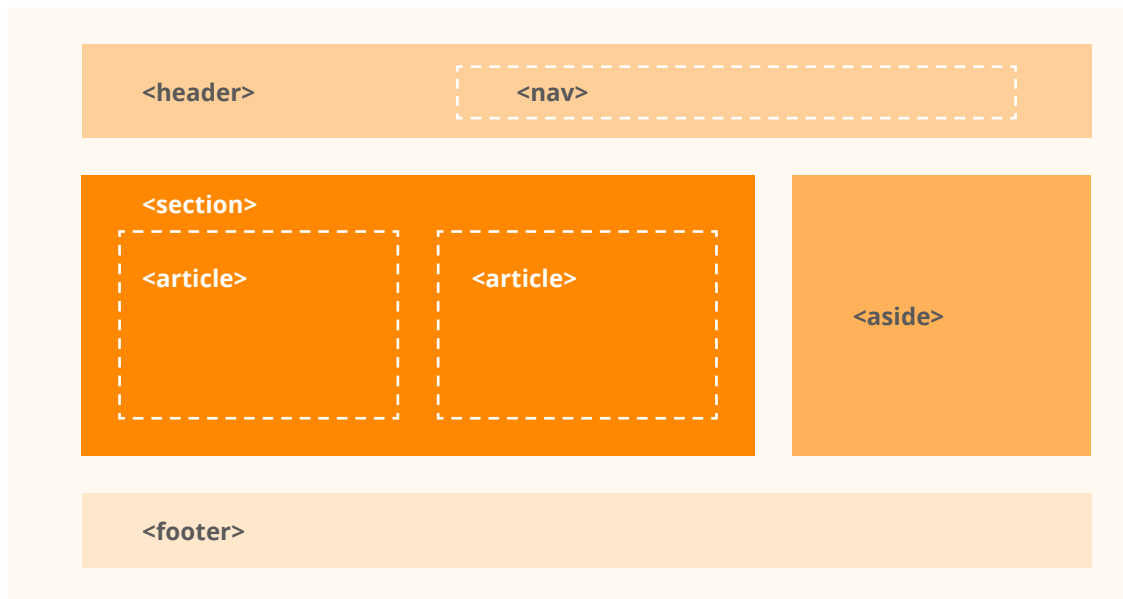


Abbildung: Die Rolle der einzelnen Elemente bei der Strukturierung einer Website.

Innerhalb des `header`-Elements findest du sowohl den Brand-Bereich – Logo inkl. Slogan – als auch die Navigation. Die Navigation befindet sich innerhalb des `nav`-Elements und wird als Aufzählungsliste ausgegeben. Beides ist semantisch richtig, da die Navigation nichts anderes als eine Auflistung zusammenhängender Inhalte ist.

Im nächsten Abschnitt schauen wir uns die erste Sektion (`section`) an. Jede Sektion bekommt ihre eigene Bezeichnung (`section id`), die du entsprechend benennen kannst. Mehr dazu gleich.

```
<section id="ueber-mich">
  <div class="innen">
    <h2>Über mich</h2>

    <p>Mein Name ist Alexios Barnabas und ich arbeite seit
25 Jahren als freier Autor. In dieser Zeit habe ich mit namhaften
Verlagen zusammengearbeitet und eine Reihe von Fachbüchern,
Artikeln in Fachzeitschriften und andere Veröffentlichungen
publiziert. Folgende Dienstleistungen biete ich an:</p>

    <ul>
      <li>Bücher und Buchbeiträge</li>
      <li>Artikel für Fachzeitschriften</li>
      <li>Blogartikel und andere Webtexte</li>
      <li>Beratung, Schulungen & Ghostwriting</li>
    </ul>
  </div>
</section>
```

Dieser ganze Bereich ist innerhalb eines `section`-Elements untergebracht. Die Überschrift dieses Bereichs wird mit einer Überschrift zweiter Ordnung (`<h2>`) abgebildet – schließlich möchten wir die `<h1>` nur einmal im gesamten Dokument vergeben. Bei der Überschriften-Struktur in einem HTML-Dokument solltest du die Hierarchie einhalten. Da der Name der Website im vorherigen Abschnitt mit einer Überschrift erster Ordnung ausgezeichnet wurde, ist es sinnvoll, dass untergeordnete Bereiche auch mit einer untergeordneten Überschrift ausgezeichnet werden. Ebenfalls ist es wichtig, dass nach einer `<h1>` auch eine `<h2>` folgt und nicht `<h3>` oder `<h4>`.

Dabei ist erst einmal egal, wie groß die Überschriften sind. Bei HTML ist die Semantik wichtig. Das Aussehen der Überschriften kannst du später in CSS zielgerichtet anpassen.

Das erste `section`-Element verfügt über ein ID-Attribut. Dadurch kann ich einerseits das Element mit dem ID-Selektor `#ueber-mich` ansprechen, um es mit CSS zu formatieren, aber noch **viel wichtiger** ist die Tatsache, dass ich dadurch andererseits auch einen Anker gesetzt habe. Auf diesen Anker kann ich dann mit einem Link verweisen und somit den speziellen Bereich mit einem Klick erreichen, und zwar mithilfe eines solchen Verweises: `Über mich`.

Nun folgt die Sektion mit den Publikationen, bei der es zwei Dinge gibt, auf die du achten solltest. Innerhalb dieser Sektion folgen wir weiterhin der Hierarchie der Überschriften. Parallel zur vorherigen Sektion ist auch die Überschrift dieser Sektion eine Überschrift zweiter Ordnung und die einzelnen Arbeiten bekommen somit logischerweise die Überschrift dritter Ordnung (`<h3>`) zugewiesen, gefolgt von Texten, die mit dem bereits beschriebenen `<p>` für einen neuen Absatz starten.

```
<section id="publikationen">
  <div class="innen">
    <h2>Aktuelle Publikationen</h2>

    <div class="portfolio">
      <div class="portfolio1">
        

        <h3>WordPress auf C-64</h3>
        <p>Mein neues Buch, in dem ich
ausführlich beschreibe, wie du erfolgreich WordPress auf einem
Commodore-Server betreiben kannst.</p>
      </div>
      <div class="portfolio2">
        

        <h3>Programmieren mit Amiga</h3>
```

```

        <p>Ein Artikel für die Fachzeitschrift
"Commodore & Web", in dem ich ausführlich beschreibe, wie du einen
Cloud-Cluster mit einem Amiga 500 betreiben kannst.</p>
    </div>
    <div class="portfolio3">
        

        <h3>Mit BASIC zwitschern</h3>
        <p>Ein Beitrag für die "Cologne Times",
wo ich Schritt für Schritt erkläre, wie du mit BASIC einen Twitter-
Klon programmieren kannst.</p>
    </div>
</div>
</section>

```

Eine weitere Sache, auf die du achten solltest: Dass du für die Bilder im Inhalt einen aussagekräftigen `alt`-Text vergibst. Der Inhalt des `alt`-Attributes bei Bildern wird angezeigt, wenn aus irgendwelchen Gründen die Bilder nicht geladen werden. Zudem ist er wichtig für die Suchmaschinen und für die Vorlesesoftware, die unter anderem Menschen mit Sehbeeinträchtigung nutzen.

Sollte das Bild nicht zum Inhalt passen, sondern einen dekorativen Charakter haben, nutze dennoch das `alt`-Attribut, lass es aber leer: `alt=""`. Bei Symbolbildern, wie sie häufig am Anfang von Webartikeln zu sehen sind, füge einen beschreibenden Text ein, etwa `alt="Mann mit Bart raucht Pfeife und schaut nachdenklich in die Ferne"`.

Im letzten Abschnitt des Quelltextes befinden sich zum einen die Kontakt-Sektion und zum anderen die Fußzeile, also der Footer-Bereich:

```

<section id="kontakt">
    <div class="innen">
        <h2>Kontakt</h2>

        <address>
            Alexios Barnabas<br>
            Paul-Attreides-Allee 1<br>
            98765 Arrakis<br>
            Dune<br><br>
            0123 456 78 90<br>
            kontakt@spice-post.abc
        </address>
    </div>
</section>

```


```
<footer>
  <div class="innen">
    <p>© 2022 Alexios Barnabas</p>
    <p><a href="impressum.html">Impressum</a> und
  <a href="datenschutz.html">Datenschutz</a></p>
  </div>
</footer>
</body>
</html>
```

Im letzten Segment siehst du ein neues Element, das `address`-Element. Es existiert bereits seit der HTML-Urzeit und ist dafür gedacht, dass man damit Kontaktdaten auszeichnet. Durch den Einsatz des `address`-Elements, zeichnest du die Adresse nicht nur semantisch korrekt aus, sondern du erleichterst auch den Suchmaschinen die Arbeit. Sie können so leichter den entsprechenden Abschnitt als Adresse klassifizieren.

Weiter darunter folgt das neuere `footer`-Element mit dem fast obligatorischen Copyright-Hinweis und mit den Verweisen auf das Impressum und die Datenschutzerklärung.

Da das Thema Impressumspflicht und Datenschutz in Deutschland ein äußerst umfangreiches und heikles Feld ist, gehen wir nicht näher darauf ein. Wer geschäftsmäßig im Web unterwegs ist, ist gut beraten, einen Dienst, wie den [Abmahnschutz von STRATO](#) zu buchen. Dabei hilft dir eine Partnerkanzlei rechtssichere Texte für deine Webpräsenz zu erstellen und haftet sogar dafür.

Nun schauen wir uns an, wie die Beispiel-Seite im Frontend aussieht, oder besser gesagt, wie sie im Browser angezeigt wird:



Alexios Barnabas

Freier Autor


- [Über mich](#)
- [Publikationen](#)
- [Kontakt](#)

Über mich

Mein Name ist Alexios Barnabas und ich arbeite seit 25 Jahren als freier Autor. In dieser Zeit habe ich mit namhaften Verlagen zusammengearbeitet und eine Reihe von Fachbüchern, Artikeln in Fachzeitschriften und andere Veröffentlichungen publiziert. Folgende Dienstleistungen biete ich an:


- Bücher und Buchbeiträge
- Artikel für Fachzeitschriften
- Blogartikel und andere Webtexte
- Beratung, Schulungen & Ghostwriting

Aktuelle Publikationen




WordPress auf C-64

Mein neues Buch, in dem ich ausführlich beschreibe, wie du erfolgreich WordPress auf einem Commodore-Server betreiben kannst.



Programmieren mit Amiga

Ein Artikel für die Fachzeitschrift "Commodore & Web", in dem ich ausführlich beschreibe, wie du einen Cloud-Cluster mit einem Amiga 500 betreiben kannst.



Mit BASIC zwitschern

Ein Beitrag für die "Cologne Times", wo ich Schritt für Schritt erkläre, wie du mit BASIC einen Twitter-Klon programmieren kannst.

Kontaktdaten

Alexios Barnabas
Paul-Attreides-Allee 1
98765 Arrakis
Dune

0123 456 78 90
 kontakt@spice-post.abc

© 2022 Alexios Barnabas
[Impressum](#) und [Datenschutz](#)

Abbildung: 100 Prozent funktional, aber schön ist anders.

Wie bereits weiter oben erwähnt: Eine reine HTML-Seite ist funktional, und das auf jeder Plattform. Momentan lässt das Design zu wünschen übrig, aber das ist nicht die Aufgabe von HTML. HTML hat seine Arbeit getan und wir haben eine funktionierende, logisch aufgebaute und semantisch korrekte Webseite hinbekommen. Um das Aussehen kümmert sich CSS und daher werden wir uns in den kommenden Abschnitten wieder dieser Sprache zuwenden.

5.3 Das Schriftbild mit CSS anpassen

Zu den grundlegenden und den ersten Anpassungen an einem HTML-Dokument gehört in der Regel die Anpassungen der Schrift. Das betrifft zum einen die Schriftgröße, die Schriftart, sowie Hervorhebungen, die entweder durch die Schriftstärke oder die Schriftfarbe erzielt werden.

Um die Schriftart festzulegen, wirst du die CSS-Eigenschaft `font-family` nutzen. Man unterscheidet Schriftfamilien, die mit Namen angegeben werden und generische Schriftfamilien. Die Angabe der generischen Schriftfamilien dient als Reserve, falls eine der namentlich genannten Schriften auf dem System nicht vorhanden sein sollte. Beispiele für bekannte Schriften wären Arial, Segoe UI, Helvetica, Calibri oder Cambria.

5.3.1 Anmerkung zur Schreibweise in CSS bei Schriften

Die Schriften mit Leerzeichen im Namen müssen, entgegen der Annahme von vielen, **nicht zwingend** in Anführungszeichen notiert werden.

Wenn ich Bezug auf Code von Anführungszeichen spreche, dann meine ich nicht die typografisch korrekten Anführungszeichen („...“ oder »...«), sondern die Zollzeichen, welche du in Windows mit der bekannten Tastenkombination `⬆ + 2` bekommst. Die einfachen Anführungszeichen bekommst du in Windows durch die Tastenkombination `⬆ + #`.

CSS ist flexibel und verzeiht vieles. Folgende drei Schreibweisen erreichen das identische Ergebnis:

```
body {font-family: Cambria, "Liberation Serif", Georgia, serif;}
body {font-family: Cambria, Liberation Serif, Georgia, serif;}
body {font-family: cambria, 'liberation serif', georgia, serif;}
```

Du kannst bei Schriftangaben wie `Liberation Serif` auf Anführungszeichen verzichten und den Namen auch kleinschreiben. Du solltest dich aber für eine Variante entscheiden und diese auch beibehalten. Ich bevorzuge diese Variante:

```
body {font-family: cambria, 'liberation serif', georgia, serif;}
```

Warum ich sie bevorzuge? Mehrteilige Schriftnamen, die mit einem Anführungszeichen versehen sind, empfinde ich als übersichtlicher. Die einfachen Anführungszeichen funktionieren, im Gegensatz zu den doppelten, auch wenn ich sie in den Inline-Styles notiere: `<p style="font-family: ,segoe ui";`. Obwohl die Anführungszeichen hier nicht zwingend notwendig sind, so verhindern sie dennoch einige [exotische Bugs²⁰](#) und außergewöhnliche Ausnahmefälle. Die Kleinschreibung empfinde ich als eine kleine Arbeitserleichterung, da ich hierfür nicht noch extra die Hochstelltaste betätigen muss.

5.3.2 Was ist ein CSS-Font-Stack?

Vieles an der Schreibweise ist persönlicher Geschmack und mit der Zeit und zunehmender Erfahrung wirst du die für dich passende Schreibweise finden. Nun zurück zu dieser CSS-Angabe:

```
body {font-family: cambria, 'liberation serif', georgia, serif;}
```

Was passiert hier? Der Browser wird schauen, ob auf dem System die Schrift Cambria existiert. Falls ja, dann wird diese Schrift verwendet. Falls sie nicht existiert, wird der Browser schauen, ob eine der anderen beiden aufgezählten Schriften existiert und sich bis ans Ende der Deklaration durcharbeiten.

Wenn keine der angegebenen Schriftfamilien existiert, greift die letzte Angabe. Bei dem Wert `serif` handelt es sich um eine generische Angabe, die da heißt: »Findest du keine der explizit genannten Schriften, binde bitte irgendeine Serifenschrift ein«.

Serif: Cambria, Constantia, Georgia

Sans-Serif: Segoe UI, Verdana, Calibri

Monospace: Consolas, Courier New

Abbildung: Die drei wichtigen generischen Schriftfamilien.

²⁰ <https://mathiasbynens.be/notes/unquoted-font-family>

CSS kennt mehrere generische Schriftfamilien. Das wären erst einmal die [Serifenschriften](#)²¹ wie Times New Roman, Georgia, Cambria oder Constantia. In den frühen Zeiten der Webentwicklung hat man diese Schriften nicht empfohlen, da sie im Webeinsatz als zu unleserlich galten. Diese Aussage gilt schon länger nicht mehr. Zum einen gibt es eine Reihe von sehr leserlichen Serifenschriften, die speziell für den Webeinsatz optimiert wurden und zum anderen ist die Auflösung der Bildschirme immer besser geworden.

Eine weitere wichtige generische Gruppe sind die Schriften ohne Serifen (**Sans-Serif**) wie **Verdana, Arial, Segoe UI** oder **Open Sans**.

Eine dritte Gruppe ist **Monospace**. Diese wird im Web in den seltensten Fällen im Fließtext verwendet. Durch die gleiche Breite für alle Zeichen erinnert es an das Schriftbild der Schreibmaschinen. Diese Eigenschaft macht sie besonders beliebt für Code-Beispiele. Bekannte Vertreter dieser Gruppe sind **Consolas, DejaVu Sans Mono, Courier New** oder **Courier**.

Für unser Beispiel setzen wir Serifenschriften ein, da sie hervorragend zu einer Website von jemandem passen, der als Autor arbeitet. Ebenso wählen wir eine etwas größere Schriftgröße und vergrößern leicht den Zeilenabstand. Der dazugehörige CSS-Code schaut so aus:

```
body {
  font-family: cambria, 'hoefler text', utopia, 'liberation Serif',
  'nimbus roman no9 l regular', times, 'times new roman', serif;
  line-height: 1.5;
  font-size: 20px;
}
```

Bei der Deklaration für die Schriftfamilie siehst du ziemlich viele Angaben. Wir starten mit der Schrift Cambria, einer von Microsoft entwickelten Schrift, die seit Vista zum Lieferumfang von Windows gehört. Ferner gehört sie zum Lieferumfang von diversen Microsoft-Produkten, daher verfügt Cambria über eine relativ hohe Verbreitung.

Sollte die Schrift auf einem Mac nicht verfügbar sein, kommt Dank der Kaskade **Hoefler Text** zum Einsatz. [Hoefler Text](#)²² wurde für Apple entwickelt und wird mit macOS ab der Version 10.3 mitgeliefert. Analog gilt das Gleiche mit **Liberation Serif** auf Linux-Systemen. Bei [Liberation](#)²³ handelt es sich um ein freies Schriftpaket, welches im Auftrag von Red Hat entwickelt wurde.

21 <https://de.wikipedia.org/wiki/Serife>

22 https://de.wikipedia.org/wiki/Hoefler_Text

23 [https://de.wikipedia.org/wiki/Liberation_\(Schriftart\)](https://de.wikipedia.org/wiki/Liberation_(Schriftart))

Solche langen Schrift-Angaben werden [Font Stacks](#)²⁴ genannt. Die Intention ist, auf so vielen System-Kombinationen wie möglich ein einheitliches Schriftbild zu bekommen. Klar, anstatt der ersten langen Code-Zeile hätte ich auch einfach `font-family: cambria, serif;` schreiben können. Das hätte auch funktioniert ... mehr oder weniger.

Aber die Gefahr wäre recht groß, dass wir auf verschiedenen Systemen ein unterschiedliches Schriftbild bekommen, weil der Browser zwar eine Serifenschrift einbindet, diese sich aber im Schriftschnitt deutlich unterscheidet. Cambria ist zwar eine Serifenschrift, tritt aber dabei dezenter und zurückhaltender als etwa Constantia, oder die [Times](#)²⁵.

In der zweiten Zeile haben wir den Zeilenabstand erhöht, womit der Text leserlicher wird. Hierbei verzichten wir auf die Angabe einer Einheit, da wir somit eine relative Angabe bekommen. Der Wert `1.5` ohne Angabe der Einheit bedeutet, dass der Zeilenabstand ("line height") 1,5-Mal so groß wie die Schriftgröße ist. Da wir in der dritten Zeile die Schriftgröße ("font-sitze") mit 20 Pixel angegeben haben, ist der Zeilenabstand 30 Pixel groß. Bereits mit diesen wenigen Angaben schaut unsere Beispielseite deutlich aufgeräumter aus:

²⁴ <https://css-tricks.com/snippets/css/font-stacks/>

²⁵ <http://www.identifont.com/differences?first=Cambria&second=times&q=Go>



Alexios Barnabas

Freier Autor

- [Über mich](#)
- [Publikationen](#)
- [Kontakt](#)

Über mich

Mein Name ist Alexios Barnabas und ich arbeite seit 25 Jahren als freier Autor. In dieser Zeit habe ich mit namhaften Verlagen zusammengearbeitet und eine Reihe von Fachbüchern, Artikeln in Fachzeitschriften und andere Veröffentlichungen publiziert. Folgende Dienstleistungen biete ich an:

- Bücher und Buchbeiträge
- Artikel für Fachzeitschriften
- Blogartikel und andere Webtexte
- Beratung, Schulungen & Ghostwriting

Aktuelle Publikationen



WordPress auf C-64

Mein neues Buch, in dem ich ausführlich beschreibe, wie du erfolgreich WordPress auf einem Commodore-Server betreiben kannst.



Programmieren mit Amiga

Ein Artikel für die Fachzeitschrift "Commodore & Web", in dem ich ausführlich beschreibe, wie du einen Cloud-Cluster mit einem Amiga 500 betreiben kannst.



Mit BASIC zwitschern

Ein Beitrag für die "Cologne Times", wo ich Schritt für Schritt erkläre, wie du mit BASIC einen Twitter-Klon programmieren kannst.

Kontaktdaten

Alexios Barnabas
Paul-Attreides-Allee 1
98765 Arrakis
Dune

0123 456 78 90
kontakt@spice-post.abc

© 2022 Alexios Barnabas

[Impressum](#) und [Datenschutz](#)

Abbildung: Eine gute Schrift macht viel aus.

Bei der Angabe der Schriftgröße habe ich extra auf die Einheit Pixel gesetzt, weil die für viele angehende Webentwickler eine Angabe ist, die man sich leichter vorstellen kann. Allerdings hat Pixel das Problem, dass es sich um eine absolute Einheit handelt. Es ist empfehlenswerter, eine relative Einheit zu nutzen.

Relative Angaben für Schriftgröße wären entweder Prozentangaben, `em` oder `rem`. Die Angaben in `%` und `em` beziehen sich auf die Schriftgröße des Elternelements, was wegen der Vererbung schnell zu einem Durcheinander führen kann, da sich die Angaben multiplizieren können.

Wenn du etwa einem Element eine **25%** größere Schriftgröße vergeben möchtest – in CSS wären das entweder `125%` oder `1.25em` – dann würde das Ergebnis deutlich größer ausfallen, wenn bereits das Elternelement selbst eine Steigerung der Schriftgröße von 20 % bekommen hat. In diesem Fall wäre die Schrift nicht um 25, sondern um 50 Prozent größer.

Um diese Falle zu umgehen, wurde die relative Einheit `rem` erfunden. Sie funktioniert wie `em`, bezieht sich aber nicht auf das Elternelement, sondern **immer** auf die Schriftgröße des Wurzelements `<html>`. Sofern nichts explizit angegeben wird, wird hierbei auf die Standardschriftgröße der Browser zurückgegriffen, die bei 16 Pixel liegt. Das `r` in `rem` steht für das englische Wort `root` (deutsch: Wurzel).

Würde ich die Schriftgröße von 20 Pixel mit `rem` nachbilden wollen, dann würde unser Beispiel so ausschauen:

```
body {
  font-family: cambria, 'hoefler text', utopia, 'liberation Serif',
  'nimbus roman no9 l regular', times, 'times new roman', serif;
  line-height: 1.5;
  font-size: 1.25rem;
}
```

Die gängigen Browser haben in der Standardeinstellung den Wert für die Basisschriftgröße mit 16 Pixel definiert. 16 Pixel mal 1,25 ergeben 20. Somit haben wir die gleiche Schriftgröße erreicht, aber dafür [mit mehr Flexibilität](https://www.24a11y.com/2019/pixels-vs-relative-units-in-css-why-its-still-a-big-deal/)²⁶.

²⁶ <https://www.24a11y.com/2019/pixels-vs-relative-units-in-css-why-its-still-a-big-deal/>

5.4 Abstände und Zentrierung für einen leserlichen Inhalt

Damit der Inhalt leserlich bleibt, solltest du immer schauen, dass der Inhaltsbereich nicht zu sehr in die Breite läuft. Daher werden wir im nächsten Schritt in CSS die Breite begrenzen und den Inhalt mittig platzieren.

Um das zu erreichen, habe ich innerhalb der einzelnen Sektion zusätzlich noch ein `div`-Element hinzugefügt. Somit bekomme ich einen zusätzlichen Ansatzpunkt, den ich mit CSS ansprechen kann. Die einzelnen Sektionen-Elemente behalten die volle Breite und die einzelnen `<div class="innen">` werden begrenzt:

```
.innen {  
  max-width: 960px;  
  margin: 0 auto;  
}
```

Um flexibler zu bleiben, habe ich die Breite nicht mit der Eigenschaft `width`, sondern mit `max-width` begrenzt. Wie du erraten kannst, steht `max-width` für die maximal mögliche Breite des Browserfensters.

Zentriert habe ich den Inhaltsbereich mittels eines bewährten Tricks. Vergibst du einem Block sowohl links als auch rechts einen Wert `auto` für den äußeren Abstand (`margin`), dann zentriert sich der Block innerhalb des Sichtfensters (Viewports). Hierbei habe ich die Kurzschreibweise benutzt, auf die wir noch eingehen. So schaut das Zwischenergebnis aus:



Alexios Barnabas

Freier Autor

- [Über mich](#)
- [Publikationen](#)
- [Kontakt](#)

Über mich

Mein Name ist Alexios Barnabas und ich arbeite seit 25 Jahren als freier Autor. In dieser Zeit habe ich mit namhaften Verlagen zusammengearbeitet und eine Reihe von Fachbüchern, Artikeln in Fachzeitschriften und andere Veröffentlichungen publiziert. Folgende Dienstleistungen biete ich an:

- Bücher und Buchbeiträge
- Artikel für Fachzeitschriften
- Blogartikel und andere Webtexte
- Beratung, Schulungen & Ghostwriting

Aktuelle Publikationen



WordPress auf C-64

Mein neues Buch, in dem ich ausführlich beschreibe, wie du erfolgreich WordPress auf einem Commodore-Server betreiben kannst.



Programmieren mit Amiga

Ein Artikel für die Fachzeitschrift "Commodore & Web", in dem ich ausführlich beschreibe, wie du einen Cloud-Cluster mit einem Amiga 500 betreiben kannst.



Mit BASIC zwitschern

Ein Beitrag für die "Cologne Times", wo ich Schritt für Schritt erkläre, wie du mit BASIC einen Twitter-Klon programmieren kannst.

Kontaktdaten

Alexios Barnabas
Paul-Attreides-Allee 1
98765 Arrakis
Dune

0123 456 78 90
kontakt@spice-post.abc

Abbildung: Inhaltsbreite begrenzt und Inhalt mittig platziert.

5.4.1 Exkurs: Kurzschreibweise von CSS-Angaben

Die Kurzschreibweise ist in CSS ein gängiges Mittel, um sich Arbeit zu sparen und die CSS-Datei kleinzuhalten, was hauptsächlich der Ladezeit der Website zugutekommt. Die Kurzschreibweise kommt häufig bei dem inneren Abstand (`padding`), dem Rahmen (`border`) und dem äußeren Abstand (`margin`) zum Einsatz.

Bei diesen Eigenschaften werden die Angaben zum Abstand oder der Rahmendicke nach folgendem Muster getätigt:

- Wert für oben
- Wert für rechts
- Wert für unten
- Wert für links

Aus einem `padding: 12px 12px 12px 12px;` kannst du in Kurzschreibweise `padding: 12px;` machen. Hier weiß der Browser, dass alle vier Seiten den gleichen Abstand bekommen. Aus `margin: 10px 5px 10px 5px;` kannst du in der Kurzschreibweise ein `margin: 10px 5px;` notieren. Tauchen zwei Werte auf, weiß der Browser, dass der erste Wert oben und unten und der zweite Wert rechts und links betrifft.

Die Kurzschreibweise wird aber auch gerne bei Schrift- und vor allem bei Farbangaben verwendet. Eine Möglichkeit, Farbwerte anzugeben, ist der HEX-Code. Der Wert startet mit einer `#` und es folgen sechs Zahlen oder Buchstaben, die die Farbe bestimmen. Zahlen und Buchstaben deswegen, weil hier nicht das dezimale, sondern das hexadezimale Zahlensystem zum Einsatz kommt.

Die Farbe Weiß ist demnach `#ffffff`, Schwarz ist `#000000`, Blau ist `#0000ff`, Rot ist `#ff0000` und grün ist `#00ff00`. Wie du schon erkannt hast, ist jeweils ein Zahlenpaar für Rot, eines für Grün und eines für Blau zuständig – da das RGB-Farbmodell zugrunde liegt. Aus der Mischung von diesen drei Farben entstehen in diesem additiven Farbmodell andere Farben. Daher können wir diese Angaben kürzen:

- **Schwarz:** `#000000` → `#000`
- **Rot:** `#ff0000` → `#f00`
- **Grün:** `#00ff00` → `#0f0`
- **Blau:** `#0000ff` → `#00f`
- **Weiß:** `#ffffff` → `#fff`
- **Grau:** `#cccccc` → `#ccc`

Du kannst aber **nur** Farbwerte kürzen, bei denen die beiden Zahlen innerhalb des Paares gleich sind. Es ist möglich `#bbaacc` auf `#bac` oder `#223344` auf `#234` kürzen, aber du kannst nicht `#243245` und auch nicht `#324451` kürzen.

Übrigens: Per HEX-Code lassen sich alle web-fähigen Farben darstellen. Du hast zum Beispiel ein Corporate Design mit einer eigenen Farbe, kennst aber nur den RGB-Farbwert oder hast sogar bloß eine RAL- oder Pantone-Angabe? Im Internet findest du Umrechner, die dir den dazu passenden HEX-Code angeben.

5.5 Bunt und flexibel

Im nächsten Schritt werden wir unserer Beispielseite etwas Farbe spendieren und diverse Elemente nebeneinander anordnen. Insgesamt hat die Beispielseite fünf Sektionen:

1. Header bzw. Kopfbereich
2. Über mich
3. Aktuelle Publikationen
4. Kontaktdaten
5. Footer bzw. Fußzeile

Bei den Bereichen 1, 3 und 5 lasse ich die weiße Hintergrundfarbe stehen und den Bereichen 2 und 4 spendiere ich einen dezenten beigeen Farbton als Hintergrundfarbe. Damit dann die Abstände innerhalb der Sektionen stimmen, habe ich den Selektor `.innen` um eine zusätzliche Zeile erweitert:

```
.innen {  
  max-width: 960px;  
  margin: 0 auto;  
  padding: 1.5rem 0 2rem 0;  
}  
#ueber-mich, #kontakt {background: #f7f4e9;}
```

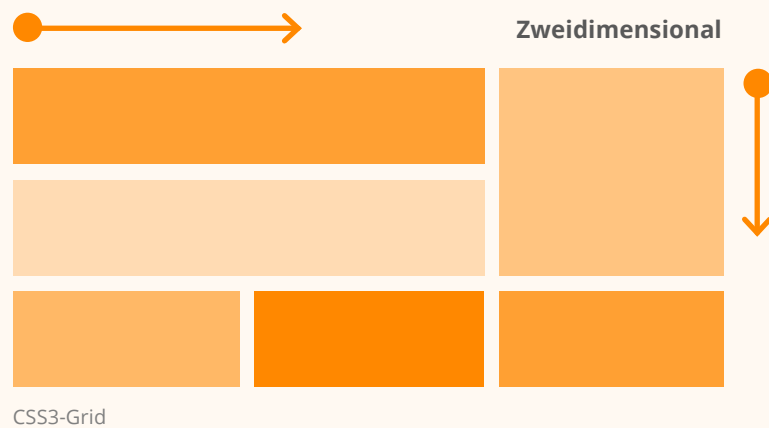
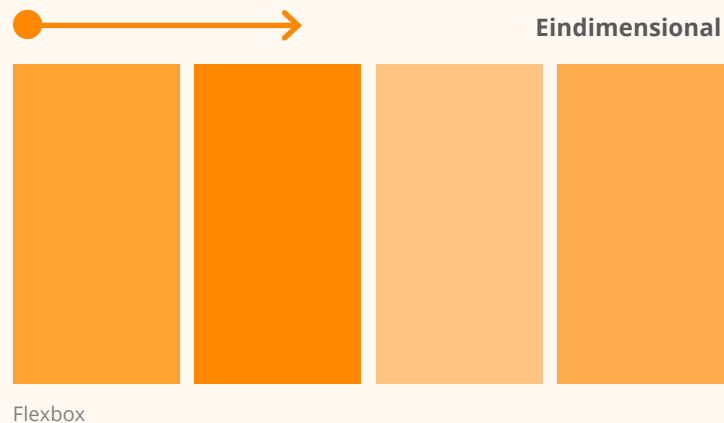
Im nächsten Schritt habe ich mit einem Rundumschlag mehrere Bereiche nebeneinander angeordnet. Das habe ich mithilfe von [Flexbox²⁷](https://wiki.selfhtml.org/wiki/CSS/Tutorials/Flexbox) gelöst:

```
header .innen, .brand, footer .innen, .portfolio {  
  display: flex;  
  justify-content: space-between;  
}
```

Flexbox – Flexible Box Modul – ist ein Layoutmodul, mit dem du die Elemente entweder horizontal oder vertikal anordnen kannst (mehr dazu unten). Wenn du diese Angabe nicht umsetzt, erscheinen die einzelnen Blockelemente untereinander. Dieses Layout habe ich auf den Innenbereich des Headers, auf den Brand-Bereich, auf den inneren Footer-Bereich und auf das Portfolio mit den Publikationen angewendet.

²⁷ <https://wiki.selfhtml.org/wiki/CSS/Tutorials/Flexbox>

5.5.1 Exkurs: Flexbox vs. Grids



Flexbox ist nicht das einzige Layoutmodul, welches CSS beherrscht. Daneben gibt es noch das [CSS Grid Modul](https://www.w3schools.com/css/css_grid.asp)²⁸. Wie bereits erwähnt kann Flexbox gleichzeitig nur eine Richtung bedienen und die Blöcke **entweder** horizontal **oder** vertikal anordnen. CSS Grid ist deutlich umfangreicher und leistungsfähiger und kann **gleichzeitig** die einzelnen Abschnitte horizontal wie vertikal anordnen. CSS Grids eignen sich somit hervorragend für anspruchsvolle Konstrukte und umfangreiche Layouts. Da wir in diesem E-Book einen recht einfachen Aufbau haben, wäre der Einsatz von Grids der sprichwörtliche Einsatz von »Kanonen auf Spatzen«.

²⁸ https://www.w3schools.com/css/css_grid.asp

Der nächste Schritt ist dem Brand-Bereich gewidmet. Dieser besteht aus zwei Unterbereichen: einerseits der Avatar-Grafik und andererseits dem Namen und dem Slogan.

Neben einem inneren Abstand für den Bereich sind die Avatar-Grafik verkleinert, die äußeren Abstände des Namens und des Slogans entfernt und die selbigen mithilfe von Flexbox vertikal angeordnet. Zudem bekamen die Überschriften erster und zweiter Ordnung eine normale Schriftstärke, damit sie nicht mehr so massig wirken. Die fette Formatierung der Überschriften gehört zu den Angaben, die das Browser-CSS mitbringt:

```
.brand {padding: 1.75rem 0 2rem 0;}
.brand img {height: 110px; width: auto; border-radius: 50%;}
.slogan h1, .slogan p {margin: 0; line-height: 1;}

.slogan {
padding-left: 1rem;
display: flex;
flex-direction: column;
justify-content: space-around;
}
h1, h2 {font-weight: normal;}
```

Damit die Avatar-Grafik etwas ansprechender aussieht, habe ich sie zusätzlich abgerundet. Das habe ich durch die Angabe `border-radius: 50%` erreicht. Die Eigenschaft `border-radius` ist in der Lage, die Ecken der Boxen [abzurunden](https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius)²⁹. Wenn du eine quadratische Box oder Grafik kreisrund bekommen möchtest, dann gibst du einfach den Wert von 50 Prozent an.

Als Nächstes ist der kleine Navigationsbereich dran. Auch der Navigationsbereich wurde mithilfe von Flexbox vertikal zentriert, die typischen Abstände der Liste sowie die Listensymbole wurden entfernt und die ganze Liste ist rechts ausgerichtet:

```
.navi {
display: flex;
flex-direction: column;
justify-content: center;
}

.navi ul {
list-style: none; padding: 0; margin: 0;
text-align: right;
}
```

²⁹ <https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius>

Responsive Design: Responsive bedeutet im Webdesign, dass die einzelnen Bereiche oder das komplette Layout der Website flexible Breiten und Höhen haben und sich so viel besser den unterschiedlichen Fenstergrößen von verschiedenen Ausgabegeräten anpassen und in sie einfügen können.

Als Nächstes ist der Portfolio-Bereich dran. Hier sind die einzelnen Publikationen bereits horizontal ausgerichtet. Nun ging es noch darum, den verfügbaren Platz optimal auszunutzen. Zuerst habe ich die (Symbol-) Bilder der Publikationen responsive gemacht, in dem ich ihnen eine maximal mögliche Breite von 100 Prozent und eine variable Höhe („height: auto;“) eingeräumt habe. Die Höhe passt sich automatisch der Breite an, da der Browser bei dieser Angabe das Verhältnis zwischen Breite und Höhe des Bildes berücksichtigt.

Die einzelnen Spalten habe ich auf maximal 31 Prozent Breite begrenzt, womit sie eine gewisse Flexibilität bekommen haben. Bei den Überschriften der einzelnen Arbeitsproben habe ich die Schriftgröße verkleinert:

```
.portfolio img {width: 100%; max-width: 100%; height: auto;}
.portfolio1, .portfolio2, .portfolio3 {max-width: 31%;}
.portfolio h3 {font-size: 1.1rem;}
```

Im vorletzten Schritt habe ich den body-Bereich noch um eine Schriftfarbe erweitert (abgekürzte Angabe), die Verweise gestylt und den Abstand des address-Elements angepasst:


```
a {color: #09a; text-decoration: none;}
a:hover {color: #333;}
address {margin-bottom: 1rem;}
```

Zu guter Letzt habe ich der Seite noch etwas mehr Farbe spendiert und dafür gesorgt, dass die Listenzeichen die blaue Akzentfarbe bekommen:

```
::marker {color: #09a;}
```

Mehr über das Pseudoelement `::marker` erfährst du [in diesem Artikel im STRATO Blog³⁰](#). Das Gesamtergebnis schaut folgendermaßen aus:

³⁰ <https://www.strato.de/blog/listenzeichen-mithilfe-von-css-individuell-gestalten/>



Alexios Barnabas
Freier Autor


[Über mich](#)
[Publikationen](#)
[Kontakt](#)

Über mich

Mein Name ist Alexios Barnabas und ich arbeite seit 25 Jahren als freier Autor. In dieser Zeit habe ich mit namhaften Verlagen zusammengearbeitet und eine Reihe von Fachbüchern, Artikeln in Fachzeitschriften und andere Veröffentlichungen publiziert. Folgende Dienstleistungen biete ich an:


- Bücher und Buchbeiträge
- Artikel für Fachzeitschriften
- Blogartikel und andere Webtexte
- Beratung, Schulungen & Ghostwriting

Aktuelle Publikationen




WordPress auf C-64

Mein neues Buch, in dem ich ausführlich beschreibe, wie du erfolgreich WordPress auf einem Commodore-Server betreiben kannst.



Programmieren mit Amiga

Ein Artikel für die Fachzeitschrift "Commodore & Web", in dem ich ausführlich beschreibe, wie du einen Cloud-Cluster mit einem Amiga 500 betreiben kannst.



Mit BASIC zwitschern

Ein Beitrag für die "Cologne Times", wo ich Schritt für Schritt erkläre, wie du mit BASIC einen Twitter-Klon programmieren kannst.

Kontaktdaten

*Alexios Barnabas
Paul-Attreides-Allee 1
98765 Arrakis
Dune*

*0123 456 78 90
kontakt@spice-post.abc*

© 2022 Alexios Barnabas

[Impressum und Datenschutz](#)

Abbildung: Alexios hat nun eine schöne Webseite.

5.6 Anpassung für mobile Geräte und Media Queries

Die Website im momentanen Zustand macht eine gute Figur auf Desktop-Rechnern und auf Laptops. Bereits in der Tablet-Ansicht fehlen allerdings die äußeren Abstände und der Inhalt »klebt« somit zu nah an den Rändern des Browserfensters. In der Ansicht der gängigen Smartphones stehen noch weniger Platz für die Inhalte zur Verfügung. Da ergibt es keinen Sinn, den Portfolio-Bereich dreispaltig zu belassen. Das sieht zum einen nicht schön aus, zum anderen kannst du bei den kleinen Bildern nichts erkennen.

Aber kein Problem. CSS gibt uns Werkzeuge an die Hand, um auch das zu lösen. Ein Werkzeug nennt man [Media Queries](#)³¹. Damit kannst du in CSS gezielt spezielle Zustände und bestimmte Breiten im Browserfenster ansprechen. Wird ein bestimmter Zustand erreicht, werden die von dir festgelegten CSS-Regeln ausgespielt.

In der Theorie ist es schwer, sich die Wirkungsweise vorzustellen, aber sobald du den Code gesehen hast, wird es dir schnell logisch vorkommen. Hier die erste Erweiterung der CSS-Datei:

```
@media only screen and (max-width: 980px) {  
  .innen {max-width: 96%;}  
}
```

Diese Regel besteht aus zwei Teilen. Im ersten Teil `@media only screen and (max-width: 980px) {...}` wird eine Bedingung aufgestellt, die heißt: »Wenn wir uns auf einem Bildschirm befinden und die Breite des Fensters 980 Pixel oder kleiner ist, dann ...«. Erst wenn diese Bedingung zutrifft, dann wird der innere Code vom Browser berücksichtigt. Die Abfrage, ob wir uns auf dem Bildschirm befinden, mag zuerst befremdlich klingen. Aber CSS-Regeln kannst du auch für die Druckansicht ausgeben lassen und das ist in diesem Fall nicht notwendig.

Somit haben wir mit dieser Anpassung erreicht, dass bei etwas kleineren Browser-Fenstern der Inhalt immer noch ausreichend Abstand zu den äußeren Rändern hat. Jetzt geht es darum, den Inhalt für noch kleinere Auflösungen zu optimieren.

```
@media only screen and (max-width: 699px) {  
  .portfolio1, .portfolio2, .portfolio3 {max-width: 100%;}  
  header .innen, .brand, footer .innen, .portfolio {display: block;}  
  .slogan {padding-left: 0;}  
  .navi ul {text-align: left;}  
}
```

Mit diesem Code-Fragment wird die Webseite für Auflösungen optimiert, die unterhalb von 700 Pixel Breite liegen. Konkret heben wir die Flexbox-Angaben auf und alle Inhalte, die dadurch nebeneinander standen, werden untereinander ausgegeben. So schaut unsere Webseite auf einem Smartphone aus, wenn man alle Inhalte untereinander darstellt:

31 <https://www.strato.de/blog/css-media-queries/>



Abbildung: Die komplette Website in der mobilen Ansicht. Erstellt mit der Screenshot-Funktion in Web-Entwickler-Tools von Firefox.

5.7 Mehr als nur Schmuckwerk: Kommentare in HTML und CSS

Du hast die Möglichkeit, sowohl in HTML als auch in CSS Kommentare einzufügen. Davon solltest du reichlich Gebrauch machen. Diese Kommentare dienen entweder dir oder anderen, die am Projekt mitarbeiten, als Hinweise und Erinnerungen, sind eine Möglichkeit, problematische oder nicht mehr benötigte Abschnitte »unsichtbar« zu machen. Wenn du die Abschnitte wieder benötigst oder die Lösung für das Problem gefunden hast, kannst du die Kommentare entfernen und der Abschnitt ist wieder sichtbar.

So schaut ein HTML-Kommentar aus:

```
<!-- Das hier wird im Browser  
nicht angezeigt. -->
```

Ein CSS-Kommentar wird nach folgendem Muster notiert:

```
/* Das hier wird vom Browser nicht  
berücksichtigt. */
```

Sowohl die CSS- als auch die HTML-Datei habe ich an mehreren Stellen kommentiert, damit der Einstieg so einfach wie möglich wird.

5.8 Die Website hochladen

Um die fertige Website online zu bringen, benötigst du einen Webspaces. Den bekommst du [bei STRATO im ersten Jahr besonders günstig](#)³² – deine eigene Domain und E-Mail inklusive. Wenn du noch nicht weißt, wie lang du online sein möchtest, kannst du das Hosting-Paket auch monatlich buchen.

Deine fertige Website lädst du zum Beispiel über das Protokoll SFTP hoch und nutzt dafür eine Software wie FileZilla. Wie das genau funktioniert, erfährst du in dieser [Anleitung von STRATO](#)³³.

³² <https://www.strato.de/hosting/>

³³ <https://www.strato.de/faq/hosting/so-einfach-veroeffentlichen-sie-ihre-seiten-im-internet-mit-filezilla/>

6. Wie geht's weiter?

Jetzt bist du in der Lage ein einfaches, aber dennoch ansehnliches Layout zu erstellen, mit dem du dich im Web präsentieren kannst. Obwohl du schon viel gelernt hast, hast du gerade einmal an der Oberfläche gekratzt. Du bist in etwa dort, wo sich ein Erstklässler befindet, der nach einem Halbjahr die ersten einfachen Wörter und Sätze schreiben kann.

HTML und vor allem CSS haben noch viel mehr zu bieten, aber lass dich davon nicht abschrecken. Bleibe dran, erstelle ruhig ein zweites Projekt, basierend auf dem Wissen aus dem E-Book und versuche es weiter anzupassen: Spiele mit den Farben, füge weitere Bilder ein und so weiter.

Wenn du dein Wissen vertiefen möchtest, empfehle ich dir zum einen das [SelfHTML-Wiki](#)³⁴. Das ist eine deutschsprachige HTML- und CSS-Dokumentation. Zum anderen ist eine weitere Quelle die MDN Web Docs, welche aus dem Mozilla Developer Network entstanden sind. Dort findest du eine [HTML](#)³⁵- und auch eine [CSS-Dokumentation](#)³⁶. Viele Einträge sind schon ins Deutsche übersetzt, aber noch nicht alle. Wenn du etwas weiter bist, ist das englischsprachige Blog CSS-Tricks eine gute Anlaufstelle.

Ich wünsche dir viel Spaß und Erfolg beim Lernen.

7. Über den Autor

[Vladimir Simović](#)³⁷ arbeitet seit 2000 mit HTML und CSS und seit 2004 mit WordPress. Im Laufe der Jahre hat er diverse Fachbücher und Fachartikel publiziert. Er ist geschäftsführender Gesellschafter der Webagentur [perun.net webwork gmbh](#)³⁸, die umfangreiche Dienstleistungen rund um WordPress und Webdesign anbietet. Seit April 2022 ist er [Senior Online-Redakteur](#)³⁹ bei der STRATO AG.



34 <https://wiki.selfhtml.org/>

35 <https://developer.mozilla.org/en-US/docs/Web/HTML?retiredLocale=de>

36 <https://developer.mozilla.org/en-US/docs/Web/CSS?retiredLocale=de>

37 <https://www.vladimir-simovic.de/>

38 <https://www.perun.net/>

39 <https://www.strato.de/blog/author/vladimir-simovic/>

