

Hinweis:



Diese Druckversion der Lerneinheit stellt aufgrund der Beschaffenheit des Mediums eine im Funktionsumfang stark eingeschränkte Variante des Lernmaterials dar. Um alle Funktionen, insbesondere Animationen und Interaktionen, nutzen zu können, benötigen Sie die On- oder Offlineversion. Die Inhalte sind urheberrechtlich geschützt.
©2018 Beuth Hochschule für Technik Berlin

CSS - Graphische Oberflächengestaltung mit CSS



Überblick und Lernziele

Analog zur Darstellung von HTML in der vorangegangenen Lerneinheit werden wir Ihnen im folgenden einen Überblick zu Konzepten und Ausdrucksmitteln geben, die Cascading Style Sheets (CSS) zugrunde liegen. Auch hier gehen wir davon aus, dass Sie bereits Kenntnisse in der Anwendung von CSS mitbringen. Wir werden uns daher zunächst auf die aus unserer Sicht interessanten Aspekte konzentrieren, die CSS als Regelsprache kennzeichnen, und werden Ihnen dann einige aktuelle Ausdrucksmittel vorstellen, die von besonderem Interesse für die Entwicklung von Webanwendungen auf mobilen Geräten und für die Umsetzung der Übungsaufgaben sind.

Die Implementierungsbeispiele zeigen exemplarisch, wie diese Ausdrucksmittel eingesetzt werden können. Für die weitere und detailliertere Beschäftigung mit CSS verweisen wir auf frei verfügbares Online-Material, z. B.  <http://www.w3schools.com/css/>, sowie auf einschlägige Entwicklungshandbücher wie  [Smf11] .



Lernziele

Nachdem Sie die Lerneinheit durchgearbeitet haben, sollten Sie in der Lage sein:

- Den Aufbau und die Funktionsweise der Regelsprache CSS zu erklären und an Beispielen zu verdeutlichen.
- Verschiedene Ausdrucksmittel von CSS zu nennen und anzuwenden
- Einzelne und Mengen von Elementen mittels Selektoren zu identifizieren.
- Den Begriff Media Queries zu erklären und dessen Relevanz bei der Entwicklung mobiler Anwendungen aufzuzeigen.



Gliederung

Gliederung

- Grundlagen von CSS
- Ausdrucksmittel von CSS
- Zusammenfassung
- Wissensüberprüfung
- Übungen






Zeitbedarf

Zeitbedarf und Umfang

Für die Bearbeitung der Lerneinheit benötigen Sie etwa 4 Stunden. Für die Bearbeitung der Übungen etwa 7 Stunden.

1 Grundlagen von CSS

Die erste Version der CSS  Spezifikation wurde bereits 1996 durch das W3C als Recommendation verabschiedet, seither wurden zahlreiche Funktionserweiterungen vorgenommen. Anders als es das Schlagwort „CSS3“ nahelegt (und im Gegensatz zur Vorgängerversion  CSS2.1) – wie auch im Unterschied zu „HTML5“ – existiert jedoch auch beim W3C kein Spezifikationsdokument dieses Namens. Stattdessen arbeitet das W3C unter dem Titel CSS Level 3 an verschiedenen Teilspezifikationen, sogenannten „Modulen“, die die Grundlage für den aktuellen Funktionsumfang von CSS bilden (siehe dazu auch  <http://www.w3.org/Style/CSS/Overview.en.html>).

„CSS3“ ist daher als ein durchaus vermarktungsfähiger Begriff anzusehen, der die Gesamtheit dieser Level 3 Spezifikationen bezeichnet – und für den ebenfalls, wie Sie in folgender Abbildung sehen, ein nicht wirklich bescheidenes Logo vergleichbar dem Logo von HTML5 existiert.



Abb.: CSS3 Logo

1.1 Stileigenschaften

Wichtigste Voraussetzung von CSS ist die Überlegung, dass die visuelle Darstellung eines HTML Elements – z. B. `<section>`, `<figure>`, ``, etc. – durch eine Menge von *Stileigenschaften* – „*Style Properties*“ – beschrieben werden kann. Dazu gehören u. a. die folgenden Eigenschaften, die grundlegend für die Beschreibung zweidimensionaler Ansichten sind, welche Text, Linien und Flächen als Grundelemente beinhalten:

- Höhe
- Breite
- Abstände
- Farbe
- Schriftgröße bzw. Liniendicke
- Schrifttyp bzw. Linienstil

Im Kern besteht CSS denn auch in einer Definition von Bezeichnern für solche Eigenschaften und der Festlegung von Wertebereichen, die entweder offen oder geschlossen sein können, d. h. die ggf. auch nur eine begrenzte Menge möglicher Werte umfassen können. Um einem HTML-Element einen bestimmten Wert bezüglich einer Stileigenschaft zuzuweisen, wird außerdem die folgende Syntax eingeführt:



Quellcode

style Attribut

Syntax für Stileigenschaft

```
001 Property-Name: Property-Wert;
```

Solche Stileigenschaftszuweisungen – oder wahlweise in Denglisch „Style-Property-Setzungen“ – können dabei direkt auf Ebene eines HTML-Dokuments durch Verwendung eines zu diesem Zweck definierten Attributs `style` gesetzt werden, welches für alle HTML-Elemente existiert. Mit CSS werden also sämtliche Darstellungsaspekte bezüglich der Elemente eines HTML Dokuments auf ein einziges allen Elementen gemeinsames Attribut konzentriert und wird auf diese Weise eine Separierung von *Struktur* und *Gestaltung* innerhalb eines HTML Dokuments herbeigeführt.

Von besonderem Interesse – auch aus dem Blickwinkel einer Beschäftigung mit verschiedenen Typen von *Programmiersprachen* – ist CSS jedoch aus einem anderen Grund. So können Stileigenschaften nämlich nicht nur durch explizite Setzung des `style` Attributs zugewiesen werden. Dies wäre auch nicht praktikabel, wenn Sie beispielsweise allen Elementen eines bestimmten Typs, die ggf. in verschiedenen HTML-Dokumenten verwendet werden, jeweils die gleichen Eigenschaften zuweisen wollten. Diese müssten Sie dann ja manuell für alle Vorkommen des betreffenden Elements an Ort und Stelle setzen. Dass aber manuelle Duplikationen und insbesondere auch etwaige Änderungen, die Sie evtl. zu einem späteren Zeitpunkt vornehmen möchten, in hohem Maße fehleranfällig sind und daher bei der Softwareentwicklung wo immer möglich vermieden werden sollten, ist Ihnen bestimmt bewusst.

CSS Stylesheets

In diesem Sinne steht uns auch in CSS ein Mittel zur Verfügung, um die Zuweisung von Stileigenschaften an die Elemente einer Menge von HTML Dokumenten komplett aus diesen Dokumenten auszulagern und in einem separaten Dokument, einem sogenannten Stylesheet zusammenzuführen oder auch über eine Menge von Stylesheets zu verteilen. Diese Stylesheets können dann von den jeweiligen HTML Dokumenten mittels `<link>` Elementen innerhalb des `head` Elements „importiert“ werden, beispielsweise wie folgt:



Quellcode

link Element

```
001 <head>
002   <title>Topicview</title>
003   <link rel="stylesheet" href="css/navigation.css"/>
004   <link rel="stylesheet" href="css/topicview.css"/>
005   <link rel="stylesheet" href="css/editview.css"/>
006   <!-- (...) -->
007 </head>
```

Hier bringen Sie zum Ausdruck, dass der Browser für die Darstellung des betreffenden Dokuments, die Stilzuweisungen verwenden soll, die in den drei an dieser Stelle referenzierten Stylesheets vorliegen. Alternativ kann die Zuweisung von Stileigenschaften aber auch nach wie vor innerhalb eines HTML-Dokuments mittels des textuellen Inhalts eines `<style>` Elements erfolgen.

Die Funktion eines Stylesheets besteht also darin, die Stileigenschaften von HTML Elementen festzulegen. Welche Mittel stehen uns dafür aber zur Verfügung? D. h. in welcher Form bringen wir zum Ausdruck, dass wir einem bestimmten Element oder einer Menge verschiedener Elemente eine bestimmte Stileigenschaft bezüglich seiner Größe, seiner Hintergrundfarbe oder seiner Umrandung zuweisen wollen?

Stilzuweisungsregeln

Die Form, die uns CSS hierfür bereit stellt, ist die Form von Regeln, die sich verallgemeinert wie folgt umschreiben lassen



Hinweis

Stileigenschaften von HTML-Elementen festlegen

„Elementen eines bestimmten Typs / einer bestimmten Position / mit bestimmten Attributen sollen die folgenden Stileigenschaften zugewiesen werden: Hintergrundfarbe: ..., Textfarbe: ..., Abstand vom Seitenrand: ..., etc.“

Beispielsweise können wir die in CSS konkrete Regeln wie die folgenden formulieren:



Beispiel

Formulierungen von CSS-Regeln

Hauptüberschriften werden in schwarzer Fettschrift mit einer Größe von 20 pt wiedergeben.
Randbemerkungen werden kursiv gesetzt.
Fachbegriffe werden in Fettschrift und blau dargestellt.

Bei CSS haben wir es also mit einer Regelsprache zu tun, die es uns erlaubt, Elemente in Dokumenten zu identifizieren und für diese Elemente Stileigenschaften anzugeben. Was aber sind Regelsprachen und was muss bei der Anwendung von Regelsprachen beachtet werden? Dieser Frage werden wir im folgenden Unterabschnitt nachgehen, da sich die allgemeinen Aussagen bezüglich Regelsprachen auch in sehr deutlicher Weise anhand von CSS illustrieren lassen.

1.2 CSS als Regelsprache

Ohne Regeln wäre unser Alltag – weder privat noch beruflich – kaum denkbar. Regeln stellen nämlich die *gemeinsame* Grundlage von Entscheidungen in einzelnen, voneinander *isolierten* Entscheidungssituationen dar und nehmen uns die Aufgabe ab, in der jeweiligen Situation immer wieder von neuem zu überlegen, wie wir handeln wollen. Anstelle dessen reduziert sich unserer Aufgabe darauf, herauszufinden welche Regel „am besten“ auf die betreffende Situation anwendbar ist und dann entsprechend dieser Regel zu handeln. Nachfolgend seien beispielhaft verschiedene solcher Entscheidungssituationen genannt:



Beispiel

Entscheidungssituationen

Ein Schachspieler ist am Zug. Was soll er in der gegebenen *Spielsituation* tun?

Ein Patient wird untersucht. Auf welche Krankheit lassen die *beobachtbaren Symptome* schließen?

Verschiedene Verkehrsteilnehmer treffen sich an einer Kreuzung. Wer darf in der *vorliegenden Konstellation* fahren?

„Verläufe“

Aus mehreren solcher voneinander isolierten Entscheidungen kann dann z. B. ein „Verlauf“ resultieren, der als eine „zusammenhängende Einheit“ betrachtet werden kann, beispielsweise ein *Schachspiel*, die *Behandlungsverlauf eines Patienten*, das *Verkehrsgeschehen in einem Straßenabschnitt* oder das *Verkehrsverhalten eines Teilnehmers*, etc. Aber auch in CSS werden wir sehen, dass aus einzelnen voneinander isolierten Entscheidungen „ein Gesamtbild“ resultiert, das sich zwar nicht wie in den beschriebenen Fällen über eine zeitliche Dimension erstreckt, dafür aber auf eine Menge gleichzeitig dargestellter und wahrnehmbarer Elemente eines HTML Dokuments bezogen ist.

Regelwerke

Regeln werden üblicherweise in Form von *Regelwerken* festgelegt, die mehrere Situationen im jeweiligen Anwendungsgebiet bzw. verschiedene Ausprägungen von Situationen mit gewissen Gemeinsamkeiten und jeweils unterschiedlichen Abweichungen abdecken. Solche Regelwerke bestehen im Grunde immer aus einer Menge von Regeln der folgenden Form:

WENN Vorbedingung DANN Aktion

Zu lesen ist eine solche Regeln wie folgt: „*Wenn Vorbedingung in einer konkreten Entscheidungssituation zutrifft, dann führe AKTION aus*“. Begrifflich spricht man dann mitunter auch vom „Match“ bzw. denglisch vom „Matchen“ einer Regel und meint damit die Tatsache, dass eine Vorbedingung zutrifft. Die Tatsache, dass aus einer Regel eine Aktion, Entscheidung o. ä. resultiert, wird auch mit dem Ausdruck des „Feuerns“ einer Regel bezeichnet.

Idealerweise werden Regeln isoliert voneinander formuliert, d. h. für die Formulierung einer Regel müssen andere Regeln nicht berücksichtigt werden. Beachtet man diese Anforderung, dann lassen sich Regelwerke grundsätzlich leicht erweitern, weil dann auch neue Regeln zu einem Regelwerk hinzugefügt werden können, ohne dass die Gesamtheit der bereits existierenden Regeln notwendigerweise bekannt zu sein braucht.

Regelsprachen

Formuliert werden Regeln der vorstehenden Form in einer *Regelsprache*. Bei einer solchen handelt es sich entweder um eine natürliche Sprache, die dann auch gewisse sprachliche Variationen der Grundform einer Regel erlaubt, oder um eine *formale Regelsprache*, die die Anwendung der Regeln eines Regelwerks durch eine Maschine ermöglicht.

Entscheidungsprobleme

Egal, ob ein Regelwerk aber in einer natürlichen oder einer formalen Sprache verfasst wurde, erscheint es fast unvermeidbar, dass aus ihm mitunter Entscheidungsschwierigkeiten resultieren können. So könnte es ja sein, dass in einer konkreten Situation die Vorbedingungen von mehr als einer Regel zutreffen und dass andererseits die Aktionen, die aus den zutreffenden Vorbedingungen resultieren, sich gegenseitig ausschließen. In einem solchen Fall stellt sich die Frage, auf welcher Grundlage dann eine *widerspruchsfreie Entscheidung* getroffen werden kann. Wir illustrieren dies an einem „klassischen“ Beispiel, bei dem wir die folgenden allgemein nachvollziehbaren Regeln verwenden:



Beispiel

Regelbeispiel

1. Wenn Du einen Vogel siehst, dann nimm an, dass er fliegen kann.
2. Wenn Du eine Ente siehst, dann nimm an, dass Sie schwimmen kann.
3. Wenn Du einen Pinguin siehst, dann nimm an, dass er nicht fliegen kann.

Nehmen wir nun an, dass wir uns in einer Situation befinden, in denen wir eine Ente sehen, dann können wir im Wissen darum, dass Enten Vögel sind, darauf schließen, dass diese Ente sowohl schwimmen kann, als auch aufgrund des Zutreffens von Regel 1 dazu in der Lage sein wird zu fliegen. Wie aber verhält es sich, wenn wir es nicht mit einer Ente, sondern mit einem Pinguin zu tun bekommen? Hier folgern wir aus unserem Wissen über Pinguine, dass das betreffende Tier nicht fliegen kann. Gleichzeitig aber sagt uns unser Wissen aber auch, dass Pinguine Vögel sind, und dass Vögel fliegen können, mithin also auch der Pinguin dazu in der Lage sein sollte. Während im ersteren Fall zwei Regeln sich hinsichtlich ihres Effekts ergänzt haben, liegt im Fall des Pinguinbeispiels also ein wechselseitiger Ausschluss der Effekte vor.

Freilich besteht für uns aus Sicht des „gesunden Menschenverstandes“ ein solcher Widerspruch gar nicht. Wie können wir diesen jedoch auf Ebene eines Regelwerks behandeln?

Eine Möglichkeit besteht darin, eine Ausnahmebedingung explizit zu formulieren und anzunehmen dass „(...) *alle Vögel fliegen können, es sei denn es handelt sich um Pinguine*“. Hier verletzen wir aber gerade die oben formulierte Empfehlung, dass „Regeln nicht voneinander wissen sollten“. Denkbar wäre daher auch eine Regel, die weniger verabsolutierend formuliert ist als die obigen Regeln und annimmt, dass „*Vögel normalerweise fliegen können*“. Wenn wir es also mit einem besonderen Vogel wie einem Pinguin zu tun haben, würde unser Sonderwissen bezüglich Pinguinen uns zu dem Schluss führen, dass dieser Vogel im Gegensatz zum Normalfall nicht fliegen kann, ohne dass wir unsere Normalfallregel verletzen oder ein nicht auflösbarer Widerspruch entstünde. Und zugleich blieben unsere Regeln isoliert voneinander, da die Normalfallregel nicht explizit Vorsorge für Pinguine und etwaige andere nicht flugfähige Vögel zu treffen brauchte.

Formale Regelsprachen

Für formale Regelsprachen stellt die hier geschilderte Entscheidungsproblematik allerdings durchaus eine Herausforderung dar, da Normalfallregeln nicht durch die Ausdrucksmittel klassischer Aussagen- und Prädikatenlogik abgedeckt sind. Es bedarf daher geeigneter Logikerweiterungen, um zu einem generalisierbaren Verfahren zu kommen, mittels dessen die widerspruchsfreie Anwendung von Regeln wie im beschriebenen Fall gewährleistet werden kann. Tatsächlich nimmt ein solcher Vorschlag an (siehe [www http://books.google.de/books?id=ejgMbaqLfNsC](http://books.google.de/books?id=ejgMbaqLfNsC)), dass Regeln als Normalfallregeln formuliert werden können und dass für Fälle von Widersprüchen generische Regeln wie ein sogenanntes „Pinguinprinzip“ greifen, mittels dessen auf Ebene der formalen Logik die intuitive Annahme nachgebaut werden kann, dass Normalfallregeln durch Sonderfälle außer Kraft gesetzt werden können.

CSS als formale Regelsprache

Auch bei CSS haben wir es mit einer formalen Regelsprache zu tun. Die Art der Entscheidungen, die wir mit dieser Sprache treffen können, lässt sich wie folgt beschreiben.



Hinweis

„Hier ist ein HTML-Dokument. Wie soll es dargestellt werden?“

In CSS wird diese Entscheidung, wie oben bereits erwähnt, zum einen auf eine Menge von Einzelentscheidungen der folgenden Form zurückgeführt:



Hinweis

„Hier ist ein HTML Element. Welche Style-Properties sollen ihm zugewiesen werden?“

Darüber hinaus trifft CSS jedoch auch für genau den Fall Vorkehrung, dass mehrere Regeln ähnlich wie unsere Regeln bezüglich Pinguinen, hinsichtlich ihrer Effekte zu widersprüchlichen Resultaten bezüglich des Werts einer Stileigenschaft kommen. Betrachten Sie beispielsweise noch einmal die folgenden bereits vorher genannten Regeln:



Beispiel

Formulierungen von CSS Regeln

Hauptüberschriften werden in schwarzer Fettschrift mit einer Größe von 20pt wiedergegeben.
 Randbemerkungen werden kursiv gesetzt.
 Fachbegriffe werden in Fettschrift und blau dargestellt.

Spezifitätshierarchie

Wie können wir auf Grundlage dieser Regeln entscheiden, wie *Fachbegriffe* in *Hauptüberschriften* bzw. *Fachbegriffe* in Randbemerkungen dargestellt werden? Hierfür gibt uns CSS ein Mittel an die Hand, das sich in seiner Grundidee sehr nahe am oben formulierten Pinguinprinzip bewegt. So legt CSS nämlich eine „Spezifitätshierarchie“ über den Vorbedingungen mehrerer ggf. gleichzeitig matchender Regeln fest und sieht vor, dass bei Widersprüchen sich die Zuweisung einer spezifischeren Regel gegenüber den Zuweisungen etwaiger weniger spezifischer Regeln „durchsetzt“ – in der gleichen Weise, wie wir oben zum Schluss gekommen sind, dass Pinguine im Gegensatz zu normalen Vögeln nicht fliegen können, ohne die Gesetze der von uns verwendeten Logik außer Kraft setzen zu müssen.

Regelsprache und
 Markupsprache

Auf die konkreten Ausdrucksmittel, die uns für die Formulierung von Vorbedingungen in CSS zur Verfügung stehen und auf die Ausgestaltung der Spezifitätshierarchie werden wir in den nachfolgenden Abschnitten weiter eingehen. An dieser Stelle möchten wir Sie jedoch noch darauf hinweisen, dass CSS als Regelsprache mit den noch vorzustellenden Eigenschaften eine sehr passende Ergänzung von HTML als deklarativer Markupsprache darstellt. So trifft ein HTML-Dokument beispielsweise die folgenden Aussagen bezüglich der in ihm verwendeten Textinhalte:

- Das ist eine Hauptüberschrift (`<h1>`)
- Das ist eine Randbemerkung (`<aside>`)
- Das ein Fachbegriff (``)

CSS nimmt dann Bezug auf diese und andere Elemente des Dokuments und formuliert Regeln bezüglich derer Stileigenschaften, aus denen dann beispielsweise hervorgeht, dass Fachbegriffe in Überschriften in blauer Schrift dargestellt werden. Und da CSS Regeln in Stylesheets gruppiert werden können und mehrere Stylesheets in einem HTML-Dokument verwendet werden können, verfügen wir auch über eine Handhabe, um ein in CSS ausgedrücktes Regelwerk „bequem“ auf mehrere Dokumente anzuwenden.

Cascade

Bevor wir uns aber konkreter mit den aus dem Blickwinkel von Regelsprachen besonders interessanten Ausdrucksmittel von CSS – Selektoren und Media Queries beschäftigen, sei noch einmal auf den Namen der hier behandelten Technologie eingegangen. Haben Sie sich diesbezüglich schon einmal gefragt, was der Begriff „Cascading“ in diesem Namen besagt? Damit ist gemeint, dass CSS eine „Kaskadierungsreihenfolge“ für Stylesheets vorsieht, bei der die Defaulteinstellungen eines Browsers durch die Stylesheets einer spezifischen Anwendung „überschrieben“ werden können. Ein Nutzer hat jedoch seinerseits die Möglichkeit, die Zuweisungen aus Stylesheets zu überschreiben, was durch den Stylesheet-Entwickler wiederum durch Markierung einer Zuweisung als `!important` unterbunden werden kann. Oberhalb der Spezifität von Vorbedingungen fließt also auch gewissermaßen die „Autorität“, die Vorbedingungen formuliert, in die zu treffenden Entscheidung bezüglich der Zuweisung von Stilmerkmalen mit ein.

2 Ausdrucksmittel von CSS

Im Sinne des in der vorliegenden Lerneinheit praktizierten „einführenden Überblicks“ werden wir nun mit Selektoren und Media Queries Ausdrucksmittel vorstellen, die CSS zur Formulierung ggf. individueller Stilanforderungen vorsieht. Dabei werden wir jeweils auch darauf eingehen, welche Kriterien bezüglich der Spezifität von Vorbedingungen in beiden Konstrukten jeweils vorgesehen sind. Wir werden schließlich mit Blick auf das dem Übungsprogramm zugrunde liegende Spezifikationsdokument ausgewählte Ausdrucksmittel vorstellen, die für die Umsetzung der Übungsaufgaben von Interesse sind.

➤ Selektoren

➤ Media Queries

➤ Anwendung von CSS

2.1 Selektoren

Selektoren sind in CSS das wichtigste Ausdrucksmittel zur Formulierung der Vorbedingungen von Regeln für Stileigenschaftszuweisungen. Mittels Selektoren können Mengen von Elementen in einem HTML Dokument identifiziert und diesen Elementen Mengen von Stileigenschaften zugewiesen werden. Dafür wird die nachfolgende Syntax verwendet.

```
001 Selektor {
002   Style-Property_1: Property-Wert_1;
003   Style-Property_2: Property-Wert_2;
004   /* ... */
005 }
```

Gemäß der beschriebenen allgemeinen Form von Regeln – „*WENN Vorbedingung DANN Aktion*“ – drückt `Selektor` hier die Vorbedingung aus, während die Property-Zuweisungen die Aktion darstellen, die bei Zutreffen einer Vorbedingung ausgeführt wird: in diesem Fall werden dem durch `Selektor` identifizierten Element die betreffenden Stileigenschaften zugewiesen. Im [www Originalton der CSS Spezifikation](#) wird die Aufgabe von Selektoren wie folgt unter Bezugnahme auf die Baumstruktur eines HTML Dokuments charakterisiert:



Definition

Aufgabe von Selektoren

“Selectors are patterns that match against elements in a tree, and as such form one of several technologies that can be used to select nodes in an XML document. Selectors have been optimized for use with HTML and XML, and are designed to be usable in performance-critical code.”

Selektoren als Muster

Wenn mittels Selektoren nicht nur einzelne Elemente in HTML Dokumenten, sondern *Mengen von Elementen* identifiziert werden sollen, handelt es sich bei Selektoren also nicht um eindeutige Identifikatoren, sondern um „Muster“, die die mit Stileigenschaften zu versehenen Elemente beschreiben. Welche Möglichkeiten haben wir aber, Elemente in HTML Dokumenten zu identifizieren? Wenn Sie sich den Aufbau eines HTML-Dokuments vergegenwärtigen, erscheint die Identifikation eines Elements „e“ anhand der folgenden Merkmale bzw. einer Kombination dieser Merkmale denkbar:

- anhand des *Elementnamens* von e
- anhand der *Attribute* von e
- anhand des Auftretens von e als *Tochter* und/oder *Mutter anderer Elemente*

Entsprechend dieser Überlegung stellt CSS uns auch tatsächlich mit *Typselektoren* und *Attributselektoren* zwei elementare Selektoren zur Verfügung, die den beiden ersten Alternativen entsprechen und lässt uns außerdem, wie im dritten Punkt angesprochen, Elemente anhand ihrer Position relativ zu anderen Elementen identifizieren. Für die Identifikation von Elementen anhand ihrer Attribute werden außerdem Ausdrucksmittel eingeführt, die der besonderen Bedeutung der Attribute `class` und `id` Rechnung tragen.

Diese Attribute ermöglichen es, innerhalb eines HTML-Dokuments Gestaltungseinheiten und gestaltungsrelevante Sinneinheiten zu markieren. Um den entsprechend ausgezeichneten Elementen auf einfache Weise Stileigenschaften zuzuweisen, stellt uns CSS mit *Klassenselektoren* und *ID-Selektoren* spezifische Ausdrucksmittel zur Verfügung, die Sie zusammen mit den anderen Beispielen für Selektoren in folgender Tabelle finden.

Selektor	Notation	Erläuterung
ID-Selektor	<code>#mainterm</code>	dasjenige Element, das in <code>id</code> als <code>mainterm</code> gekennzeichnet ist
Klassenselektor	<code>.term</code>	alle Elemente, die in <code>class</code> als <code>term</code> gekennzeichnet sind
Attributselektor	<code>[attr1=val1]</code> <code>[attr2]</code>	alle Elemente, deren Attribut <code>attr1</code> den Wert <code>val1</code> hat alle Elemente, deren Attribut <code>attr2</code> gesetzt ist
Typselektor	<code>li</code>	alle <code>li</code> Elemente
Universalselektor	<code>*</code>	alle Elemente

Tab.: Beispiele für elementare Selektoren in CSS

CSS stellt außerdem Ausdrucksmittel zur Verknüpfung elementarer Selektoren zur Verfügung und erlaubt es damit, Elemente in Abhängigkeit von mehreren Merkmalen zu identifizieren. Zu den wichtigsten Verknüpfungen dürften hier die Konjunktion (Verbindung) elementarer Selektoren und die Identifikation von Elementen anhand ihrer Vorgänger im HTML-Dokument darstellen. Für diese Verknüpfungen stellt uns CSS eine sehr reduzierte Syntax zur Verfügung, die nicht unanfällig für Fehler ist. So besteht denn der Unterschied zwischen beiden Verknüpfungen im Nichtvorhandensein bzw. Vorhandensein eines Leerzeichens zwischen den Selektoren. Eine Übersicht zeigt die verschiedenen Möglichkeiten zur Verknüpfung von Selektoren.

Verknüpfung	Notation	Erläuterung	Hinweise
„Nachkommenschaft“	<code>selektor1 selektor2</code>	alle durch <code>selektor2</code> identifizierten Elemente, die Nachkommen eines durch <code>selektor1</code> identifizierten Elements sind	Aneinanderreihung mit Leerzeichen
„Kindsein“	<code>selektor1 > selektor2</code>	alle durch <code>selektor2</code> identifizierten Elemente, die Töchter eines durch <code>selektor1</code> identifizierten Elements sind	Verwendung von <code>></code>
„Geschwisterschaft“	<code>selektor1 + selektor2</code>	alle durch <code>selektor2</code> identifizierten Elemente, die unmittelbar auf ein durch <code>selektor1</code> identifiziertes Element folgen	Verwendung von <code>+</code>

	<code>selektor1 ~ selektor2</code>	alle durch <code>selektor2</code> identifizierten Elemente, die Geschwister eines vorangehenden durch <code>selektor1</code> identifizierten Elements sind	Verwendung von <code>~</code>
Konjunktion	<code>span.term#mainterm</code>	dasjenige <code>span</code> Element, das via <code>@class</code> als <code>term</code> und via <code>@id</code> als <code>mainterm</code> gekennzeichnet ist	Aneinanderreihung ohne Leerzeichen
	<code>div.class1.class2</code>	alle <code>div</code> Elemente, die sowohl als <code>class1</code> , als auch als <code>class2</code> deklariert sind	
	<code>input[attr1=val1][attr2=val2]</code>	alle <code>input</code> Elemente, die <code>val1</code> als Wert von Attribut <code>attr1</code> und <code>val2</code> als Wert von <code>attr2</code> haben	
Disjunktion	<code>selektor1, selektor2</code>	alle Elemente, die entweder durch <code>selektor1</code> oder durch <code>selektor2</code> identifiziert werden	Kommaseparierung von ggf. zusammengesetzten Selektoren, nicht verwendbar innerhalb eines zusammengesetzten Selektors
	<code>selektor1 > element3 *, selektor2 > element3 *</code>	alle Elemente, die sich in einem Element <code>element3</code> befinden, auf dessen Mutter entweder <code>selektor1</code> oder <code>selektor2</code> zutrifft.	
Negation	<code>span.term:not(#mainterm)</code>	alle <code>span</code> Elemente, denen in <code>@class</code> die Klasse <code>term</code> zugewiesen ist und deren <code>@id</code> nicht <code>mainterm</code> ist.	Verwendung der Pseudoklasse <code>:not()</code> . Als Argumente können nur elementare Selektoren verwendet werden.
	<code>span:not(.term):not(#mainterm)</code>	alle <code>span</code> Elemente, denen weder in <code>@class</code> die Klasse <code>term</code> , noch als <code>@id</code> der Wert <code>mainterm</code> zugewiesen ist	

Pseudoklassen

Mittels der genannten Typen von Selektoren können Elemente anhand explizit vorliegender Eigenschaften wie Elementnamen, Attributen und Beziehungen zu anderen Elementen identifiziert werden. Zusätzlich stellt Ihnen CSS mit Pseudoklassen ein weiteres Ausdrucksmittel zur Verfügung, um Mengen von Elementen anhand von Merkmalen zu beschreiben, welche anwendungsübergreifend relevant sind und sich nicht mittels Selektoren überprüfen lassen. Dazu gehören insbesondere strukturelle Eigenschaften wie z. B. die Eigenschaft, das einzige, das erste oder das letzte Kind eines Elements zu sein oder eine wiederkehrende Stelle in einer Folge von Geschwisterelementen zu besetzen, welche mittels der Pseudoklassen `:only-child`, `:first-child`, `:last-child` bzw. `:nth-child(n)` überprüft werden können.

Pseudoklassen werden aber auch eingesetzt, um die ggf. temporären *Interaktionszustände* oder auch die Bedienbarkeit von Elementen zu identifizieren. Beispielsweise kann mittels `:enabled` vs. `:disabled` überprüft werden, ob Eingabeelemente in Formularen oder Aktionselemente bedienbar sind oder nicht, und bezeichnet die Pseudoklasse `:active` einen Link oder Button, der gerade durch den Nutzer bedient wird und dem für diesen Fall ggf. eine alternative Darstellung – sei es durch Veränderung der Hintergrundfarbe und/oder der Durchsichtigkeit – zugewiesen werden kann.

Nicht zuletzt wird uns auch der Negationsoperator bezüglich Selektoren, wie in voriger Tabelle gezeigt, durch eine Pseudoklasse – `:not()` zur Verfügung gestellt. Eine Übersicht über die aktuell verfügbaren Pseudoklassen finden Sie z. B. in Dokumentationen oder Tutorials wie der www.mozilla.org/docs/Developer/Tools/Selectors oder www.w3schools.com.

Wildcards

Der Vollständigkeit halber sei hier außerdem erwähnt, dass Wildcards in Selektoren nicht nur in Form des Universalselektors als eines elementaren Selektors auftreten können, sondern dass auch für die Auswertung von Attributwerten innerhalb eines Attributselektors Möglichkeiten zur Generalisierung von String-Werten zur Verfügung stehen. So können die String-Operationen „starts-with“, „ends-with“ und „contains“ mit den Operatoren `^=`, `$=` bzw. `*=` realisiert werden und lässt sich mit den Operatoren `~=` und `|=` überprüfen, ob ein Attributwert einen String als Teil einer Leerzeichen- bzw. Bindestrich-separierten Liste von Strings enthält.

Spezifität

Wie aber werden nun auf Basis einer Menge von CSS Regeln die Stileigenschaften ermittelt, die tatsächlich einem Element zugewiesen werden sollen?

Hierfür ist es erforderlich, zunächst das Konzept der *Spezifität* von Selektoren einzuführen. Dieses erlaubt es, für eine Menge gegebener Selektoren zu entscheiden, welche Selektoren spezifischer als die anderen sind. Die Grundlage hierfür ist die oben vorgestellte Unterscheidung elementarer Selektoren und eine grundsätzliche Annahme, wie „scharf“ ein gegebenes Element jeweils durch diese Selektoren beschrieben wird. So erscheint es intuitiv, dass ID-Selektoren als eindeutige Bezeichner die höchste Schärfe aufweisen. Klassenselektoren sind ihnen an Schärfe untergeordnet, werden jedoch als gleich spezifisch wie Attributselektoren und Pseudoklassen betrachtet und sind spezifischer als Typselektoren, die ein Element nur anhand seines Elementnamens bezeichnen. Der Universalselektor als der am wenigsten spezifische Selektor wird für die Ermittlung der Spezifität nicht berücksichtigt. Es ergibt sich damit die folgende Spezifitätshierarchie für elementare Selektoren, wobei das Symbol ' $>$ ' hier im Sinne von „ist spezifischer als“ zu lesen ist:

```
ID-Selektor > Klassenselektor, Attributselektor, Pseudoklasse >
Typselektor (> Universalselektor)
```

Wie aber kann auf dieser Grundlage die Spezifität eines zusammengesetzten Selektors ermittelt werden?

Hierfür sieht CSS vor, dass für jeden der oben genannten Selektorentypen die Auftretenshäufigkeit in einem zusammengesetzten Selektor gezählt wird. Die Ziffern für die Auftretenshäufigkeiten der einzelnen Typen werden dann *aneinandergefügt* und bilden eine dreistellige Zahl, deren dritte Stelle (von hinten gezählt!) die Anzahl der ID-Selektoren, die zweite Stelle die Anzahl der Klassen-, Attribut- und Pseudoselektoren und deren erste Stelle die Anzahl der Typselektoren darstellen. Betrachten wir dafür die folgenden Selektoren, die alle ein Element des Typs `<section>` identifizieren und die Ergebnisse bezüglich deren Spezifität, die uns das genannte Verfahren liefert. Die Abkürzung „KAP“ steht hier für die Anzahl der Klassen- Attribut- oder Pseudoselektoren, die die zweite Stelle des Spezifitätswerts bilden:

Tab.: Selektoren vom Typ
<section>

	Selektor	ID	KAP	Typ	Spezifizität
A	section#main	1	0	1	101
B	article section	0	0	2	2
C	section.intro	0	1	1	11

Spezifizitätsreihenfolge

Der Vergleich der hier ermittelten Zahlenwerten liefert uns dann die folgende Spezifizitätsreihenfolge für die drei Selektoren:

A > C > B

Vielleicht fragen Sie sich aber, wie Fälle behandelt würden, in denen eine Auftretenshäufigkeit einen Zahlenwert von > 9 hat, d. h. im Dezimalsystem mehr als einstellig wäre. In diesem Fall würde die bloße Zusammenfügung der Zahlenwerte ja ggf. eine mehr als dreistellige Zahl ergeben und die Ergebnisse über einer Menge von Selektoren evtl. verfälschen. Hierfür trifft die CSS Spezifikation jedoch die durchaus elegante Annahme, dass dann die Basis des Zahlensystems so gewählt werden sollte, dass wiederum eine dreistellige Zahl gebildet werden kann, d. h. die Basis sollte mindestens der höchsten Auftretenshäufigkeit entsprechen.

Zuweisung von Stileigenschaften

Unter Annahme, dass die Spezifität eines Selektors wie vorstehend beschrieben ermittelt werden kann, wird einem Element die Menge aller Stileigenschaften zugewiesen, für die es eine Regel gibt, auf deren Selektor das betreffende Element zutrifft. Dies entspricht unserem Beispiel aus der Tierwelt, wonach wir von einer Ente wissen, dass sie als Vogel fliegen kann und als Ente (bzw. als Wasservogel) auch dazu in der Lage ist zu schwimmen.

Weisen mehrere Regeln allerdings für einunddieselbe Stileigenschaft unterschiedliche Werte zu, dann wird derjenige Wert verwendet, der durch die Regel mit höchster Spezifität zugewiesen wird. Diese „überschreibt“ sozusagen die Zuweisungen der Regeln, die niedrigere Spezifität haben. CSS nimmt hier also tatsächlich ein „Pinguinprinzip“ an, dessen Grundlage die Spezifitätshierarchie bildet: so haben wir die Annahme, dass Pinguine fliegen können, oben ja aus dem Grund verworfen, dass die hierfür geltende Normalfallregel für Vögel durch den spezifischeren Fall von Pinguinen außer Kraft gesetzt wird.

Rückblickend auf die eingangs erwähnte Verwendung eines `style` Attributs innerhalb des HTML-Dokuments sei noch erwähnt, dass dessen Stilzuweisungen als grundsätzlich spezifischer als die Zuweisungen durch CSS-Regeln angenommen werden und sich daher ggf. gegenüber einer Menge konkurrierender Selektoren durchsetzen.

Die fundierte Definition der Spezifität von Selektoren in CSS ermöglicht es also, dass Konflikte bei der Anwendung von CSS Regeln aufgelöst werden können. Beachten Sie aber, dass es hier nur um „einfache“ Konflikte geht, die sich auf ein einziges Stilmerkmal beziehen. Etwaige Widersprüche oder Inkonsistenzen einer Menge von Stilzuweisungen, die durch verschiedene Regeln veranlasst werden, lassen sich damit weder erkennen noch beheben.

2.2 Media Queries

Media Queries sind ein neues Ausdrucksmittel, das in der aktuellen Version von CSS eingeführt wird. Betrachtet man CSS als eine Regelsprache, dann können Media Queries als eine Erweiterung der Möglichkeiten gesehen werden, mittels derer sich Vorbedingungen für die Zuweisung von Stileigenschaften formulieren lassen. Wie wir gerade gesehen haben, können mit Selektoren Vorbedingungen bezüglich der Eigenschaften eines HTML Dokuments formuliert werden, das durch einen Browser dargestellt wird. Media Queries nehmen demgegenüber eine Erweiterung insofern vor, als mit Ihnen Bedingungen bezüglich der Geräte formuliert werden können, auf denen ein Browser ausgeführt wird, sowie bezüglich der Anzeigefläche die darauf für die Darstellung eines Dokuments zur Verfügung steht.

Viewport

Was wir hier informell „Anzeigefläche“ genannt haben, wird im Rahmen von CSS als *Viewport* bezeichnet. Viewports lassen sich hinsichtlich einer Menge von Merkmalen wie *Größe*, *Ausrichtung*, *Auflösung*, *Farbigkeit* etc. beschreiben, die wir nachfolgend *Media Features* nennen werden und die größtenteils auf Eigenschaften der zugrundeliegenden Gerätehardware zurückgeführt werden können. Media Queries ermöglichen es uns, durch die Formulierung von *Media Expressions* die Zuweisung von Stileigenschaften von Ausprägungen dieser Merkmale abhängig zu machen. Vorgesehen sind in der Spezifikation z. B. die folgenden Media Features bezüglich der genannten Dimensionen:

- **Größe:** `width`, `height`, `device-width`, `device-height`
- **Ausrichtung und Seitenverhältnis:** `orientation`, `aspect-ratio`, `device-aspect-ratio`
- **Farbigkeit:** `color`, `color-index`, `monochrome`
- **Auflösung:** `resolution`

Auch wenn Media Queries damit wie Selektoren als Ausdrucksmittel für die Vorbedingungen von Stilzuweisungsregeln angesehen werden können, legt die dafür vorgesehene Syntax nahe, dass sie konzeptuell auf einer anderen Ebene als Selektoren angesiedelt sind:

```
001 @media Media-Type Media-Expressions {
002     CSS-Regeln
003 }
```

Der Ausdruck `Media-Expressions` bezeichnet hier eine Menge von Bedingungen bezüglich der Media Features eines Viewports. Bei `Media-Type` handelt es sich um einen optionalen Ausdruck, der die Klasse des zur Darstellung verwendeten Geräts bezeichnet. Vorgesehen hierfür sind derzeit insbesondere die Geräteklassen `screen`, `handheld`, `print`, `projection` und `tv`, aber auch `speech`, und `braille`, die mit Screenreadern und Ausgabegeräten für Blindenschrift auch nicht visuelle Wiedergabegeräte umfassen. Auch für diese ist eine wiedergabebezogene Aufbereitung von HTML-Dokumenten relevant, wie wir sie hier am Beispiel *visueller* Wiedergabe betrachten.

Media Expressions

Wie Selektoren können auch Media Expressions in elementare und zusammengesetzte Expressions unterschieden werden. Erstere werden aus dem Namen eines Media Features, einem Vergleichsoperator und einem Literalwert gebildet, mit dem der vorliegende Wert des Media Features verglichen wird. Die Notation erfolgt in runden Klammern – z. B. kann die Überprüfung der Bildschirmorientierung wie folgt durchgeführt werden:

```
@media (orientation: portrait) { /* CSS-Regeln */ }
```

Bei skalenwertigen Features können außerdem Wertebereiche mittels der Hinzufügung der Präfixe `min-` bzw. `max-` festgelegt werden, so gelten die CSS Regeln im nachfolgenden Beispiel nur für Geräte, deren Bildschirmbreite maximal 350 px beträgt:

```
@media (max-width: 350px) { /* CSS-Regeln */ }
```

Komplexe Ausdrücke können schließlich mittels der Operatoren `and` und `not` sowie des auch für Selektoren verwendeten Operators `,` für Disjunktionen gebildet werden:

```
@media (min-width: 500px) and (orientation: landscape) { /* CSS-Regeln */ }
```

Auswertung von Media Queries

Trifft ein Media Query zu, dann werden dessen Selektoren und die damit verbundene Zuweisung von Stileigenschaften *jeweils* bezüglich *dieses* Media Queries ausgewertet, d. h. das oben für Selektoren dargestellte Auswertungsverfahren wird nicht übergreifend über eine Menge möglicherweise zutreffender Media Queries durchgeführt. Die Auswertung von Media Queries selbst erfolgt in der Notationsreihenfolge innerhalb eines CSS Stylesheets bzw. in der Reihenfolge, in der Stylesheets via `<rel>` in ein HTML Dokument eingebunden werden. Es ist damit also möglich, dass im Zuge der Auswertung der Regeln für einen Media Query Stileigenschaften rückgängig gemacht werden, die im Rahmen zuvor ausgewerteter Media Queries gesetzt worden waren.

Diese Verarbeitungsreihenfolge ist insbesondere dann zu beachten, wenn aus Sicht des Entwicklers die Regeln verschiedener Media Queries aufeinander aufbauen sollen. So überwiegt die reihenfolgebezogene Auswertung von Media Queries insbesondere die spezifizitätsbezogene Auswertung von Selektoren, d. h. es ist möglich, dass die einem Element zugewiesenen Stileigenschaften durch einen Selektor niedrigerer Spezifität überschrieben werden, der sich jedoch innerhalb eines später ausgewerteten Media Queries befindet.

Angemerkt sei an dieser Stelle, dass Media Queries nicht nur *innerhalb* von CSS Stylesheets verwendet werden können, sondern bereits für den Import von Stylesheets oder anderer Ressourcen mittels des `<link>` Elements genutzt werden können. Hierfür steht auf `<link>` ein `media` Attribut zur Verfügung, dessen Wert ein Media Query entsprechend der hier vorgestellten Syntax ist. Beispielsweise könnte der im vorstehenden Beispiel verwendete Media Query auch wie folgt eingesetzt werden:

```
<link rel="stylesheet" href="css/widescreen.css" media="(min-width: 500px) and (orientation: landscape)"/>
```

Geräteklassen

Die Form der Verarbeitung wie auch ihre Notation legen eine typische Verwendung von Media Queries nahe, da beide annehmen, dass diese einen Rahmen für die Formulierung von CSS-Regeln auf Basis von Selektoren bilden. So erscheint es sinnvoll, dass mittels Media Queries eine beschränkte Menge möglicher Klassen von Endgeräten bzw. „Bedienszenarien“ bezüglich dieser Endgeräte formuliert werden.

Beispielsweise könnten die Geräteklassen „Bildschirm“, „Tablet“ und „Smartphone“ formuliert werden und bezüglich der letzteren beiden die Szenarien einer horizontalen vs. einer vertikalen Bildschirmausrichtung charakterisiert werden. Denkbar wäre es außerdem, durch Formulierung von Anforderungen bezüglich der Bildschirmauflösung hochauflösende von „normalauflösenden“ Geräten zu unterscheiden. In welchem Maße eine Differenzierung sinnvoll ist, hängt von der erwartenden Anzahl zugreifender Geräte und dem wünschenswerten / erforderlichen / finanzierbaren Maß an Anpassung ab, das die optimale Bedienung einer Anwendung auf den betreffenden Geräten gewährleisten kann. Grundsätzlich aber erlauben es Media Queries, jeder auf diese Weise definierten Klasse eigene Stileigenschaften zuzuweisen.

Responsive Design

Media Queries erweisen sich damit als ein wichtiges Ausdrucksmittel, um die Anforderungen eines Gestaltungsansatzes für Webanwendungen zu erfüllen, der unter dem (buzzwordverdächtigen) Begriff des *Responsive Design* bekannt ist. Damit werden Anwendungen bezeichnet, die versuchen, in optimaler Weise auf die Rahmenbedingungen ihrer Nutzung zu reagieren, wobei das verwendete Endgerät ein grundlegendes Merkmal dieser Rahmenbedingungen bildet. Typisches Kennzeichen eines solchen Ansatzes ist, wie in folgender Abbildung die Verwendung eines „Spalten-“ oder *Gitterlayouts* für die darzustellenden Inhalte, bei dem die Spalten oder auch die einzelnen Zellen eines Gitters je nach Bildschirmbreite nebeneinander oder übereinander angeordnet werden.

Zur Abgrenzung von Responsive Design von „Adaptive Design“ als einem – von außen betrachtet nahezu synonymen – Ansatz verweisen wir an dieser Stelle auf zahlreiche Diskussionen und Erläuterungen im Netz und geben zu bedenken, dass zur Motivation von Begriffsbildungen neben inhaltlichen Argumenten immer auch der „Reiz des Neuen an sich“ beiträgt (siehe z. B. <http://www.techrepublic.com/blog/web-designer/what-is-the-difference-between-responsive-vs-adaptive-web-design/>).



Abb.: Illustration der Grundzüge eines Responsive Design

Illustration der Grundzüge eines Responsive Design anhand der verschiedenen Realisierungen eines spaltenbasierten Layouts auf Geräten unterschiedlicher Bildschirmbreite.

Media Queries erscheinen in besonderer Weise geeignet, um Anforderungen an Webanwendungen umzusetzen, welche ausschließlich oder als eine alternative Nutzungsmöglichkeit auf mobilen Endgeräten ausgeführt werden. Zumindest die Unterscheidung von „Tablets“ und „Smartphones“ sowie ggf. Differenzierungen einzelner Ansichten in Abhängigkeit von Bildschirmausrichtung und evtl. auch der Displayauflösung lassen sich mit Media Queries geradlinig und ohne Intervention in die eigentlichen Regeln für die Zuweisung von Stileigenschaften vornehmen. Für einen kompletten Überblick über alle aktuellen und zukünftigen Ausdrucksmittel für Media Queries verweisen wir auf die www.w3.org/ Spezifikation beim W3C.

2.3 Anwendung von CSS

Mit Selektoren und Media Queries haben wir zwei Ausdrucksmittel vorgestellt, die für CSS als Regelsprache von besonderer Bedeutung sind. Die spezifischen Möglichkeiten, mittels CSS den Elementen eines HTML Dokuments Stileigenschaften zuzuweisen, werden wir nun anhand einiger Beispiele aus dem Umfeld der Implementierungsbeispiele erläutern.

Wir werden hierfür zunächst anhand eines Beispiels die schrittweise Realisierung einer Gestaltungsvorlage nachvollziehen und anhand dieser die Verwendung des „Box Model“ von CSS zur Gestaltung von Flächen und Text illustrieren. Mit diesem haben Sie im Rahmen Ihrer Vorbeschäftigung mit CSS vermutlich bereits Bekanntschaft gemacht. Weitere Beispiele zur Verwendung von CSS finden Sie außerdem in der folgenden Lerneinheit, in der wir uns in u. a. mit dem Zusammenspiel von HTML, CSS und JavaScript beschäftigen werden.

Wichtige Grundlage für die Verwendung von CSS zur Umsetzung von Flächenelementen mit und ohne Text ist das sogenannte Box Model. Dieses bildet die konzeptuelle Grundlage für die Gestaltung jeglicher Elemente, welchen als Block-Elementen zweidimensionale Dimensioneigenschaften zugewiesen werden können. Dazu gehören insbesondere Höhen, Breiten und Abstände zu anderen Elementen, aber z. B. auch die Dicke von Trennlinien oder die Wahl des Bezugspunkts für ein Hintergrundbild. Eine „offizielle“ Visualisierung des Modells durch das W3C finden Sie in folgender Abbildung. Wichtig ist hier insbesondere die Unterscheidung der Dimensionen *Margin* und *Padding*. Diese bezeichnen die Abstände eines Elements zu seinen benachbarten Elementen bzw. zu seinen Inhalten, wobei der Bezugspunkt für beide Dimensionen jeweils die sichtbare oder unsichtbare Randlinie (*Border*) eines Elements ist.

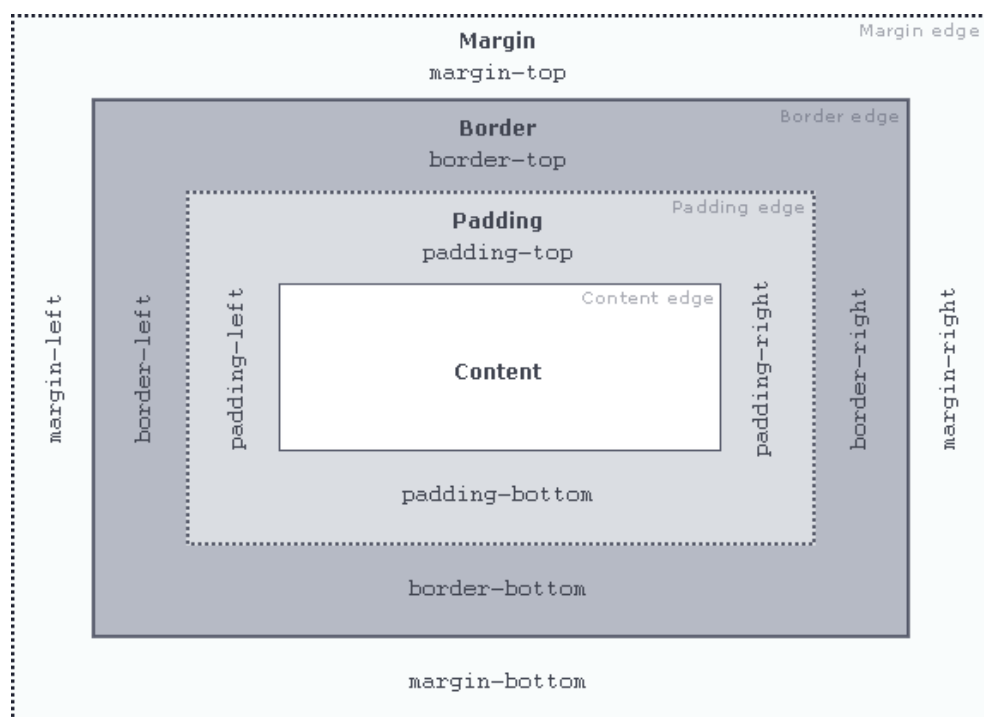


Abb.: Box Model für Blockelemente in CSS

Box Model für Blockelemente in CSS. In den hier verwendeten Abkürzungen „TM“, „BB“, „RP“ stehen die Initialbuchstaben für „Top“ vs. „Bottom“ und „Right“ vs. „Left“, jeweils bezogen auf den als „Content“ bezeichneten Bereich als Zentrum einer Box. Beachten Sie aber auch, dass diese Richtungsbezeichnungen in den Namen der Stileigenschaften als `margin-top`, `border-bottom` bzw. `padding-right` notiert werden, d. h. sie werden hinten an die Bezeichnung der betreffenden Box angehängt.

Zur Illustration des Box Model betrachten wir eine mögliche Umsetzung der Gestaltungsvorlage für das im Designkonzept vorgesehene Gestaltungselement „Verknüpfungen“, das verfügbare Querverweise auf einer Ansicht anzeigt. Dieses soll wie folgt dargestellt werden:

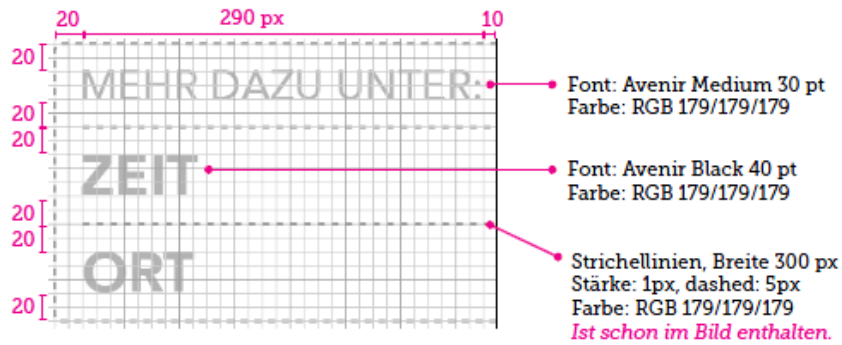


Abb.: Querverweise auf einer Ansicht

Für die nachfolgenden Ausführungen sei beachtet, dass das Designdokument ein iPhone mit hochauflösendem „Retina“-Display als Zielgerät annimmt. Abweichend davon formulieren wir unserer Stilregeln bezüglich eines Geräts mit normalauflösendem Display und halbieren daher mindestens die vorgesehenen Größenangaben. Als Hintergrundfarbe nutzen wir, wie auch im Designdokument an anderer Stelle vorgesehen, ein dunkles Grau.

Ausgangspunkt der Realisierung des Gestaltungselements ist das folgende HTML-Markup. Dieses bringt die Tatsache zum Ausdruck, dass das betreffende Element im Kern eine Listenansicht enthält, welche im vorliegenden Fall zwei Listeneinträge, nämlich „Ort“ und „Zeit“ umfasst, und mit einer Überschrift versehen ist:



Quellcode

Realisierung des Gestaltungselements

```
001 <aside id="verknuepfungen">
002   <h2>Mehr dazu unter:</h2>
003   <ul>
004     <li><a href="#">Ort</a></li>
005     <li><a href="#">Zeit</a></li>
006   </ul>
007 </aside>
```

Wenn wir dieses Element ohne jegliche Stilzuweisungen im Browser darstellen lassen, bekommen wir dafür die folgende Ansicht:



Abb.: Ansicht

Mag diese Ansicht noch sehr weit vom erstrebenswerten Zielzustand entfernt sein, so liefert sie uns dennoch bereits einige Anhaltspunkte bezüglich der Änderungen, die wir in unseren eigenen CSS Regeln herbeiführen müssen, um die Default-Realisierung des Browsers „auszuschalten“. So muss u. a.

- die Schriftgröße für Überschrift und Listenelemente angepasst werden
- die Einrückung der Listenelemente entfernt werden
- das Bullet-Symbol für die Listenelemente entfernt werden
- die Unterstreichung und Farbgebung für die <a> Elemente in den Listenelementen entfernt werden

Auf diese Anforderungen werden wir im weiteren Verlauf noch zurückkommen. Wir betrachten dafür die Regeln, die unser CSS Stylesheet definiert und beginnen beim Wurzelement `<aside>`. Da es sich um das einzige Element des Typs „Verknuepfungen“ handelt, das in einer Ansicht auftreten kann, identifizieren wir dieses mit einem id Attribut und können auf dieses wiederum mittels eines ID-Selektors zugreifen. Unsere Regeln sind hier und im folgenden also unabhängig davon, welches konkrete HTML-Element als Wurzelement von `verknuepfungen` verwendet wird.

Die nachfolgende Regel weist außerdem mittels `color` bereits die Schriftfarbe zu, die für alle Texte innerhalb von `verknuepfungen` verwendet werden soll. Mittels des `border` Attributs wird außerdem die Umrandung hinsichtlich Stärke, Stil und Farbe festgelegt. Während `margin` den Abstand des Elements zu anderen (hier nicht betrachteten) Elementen festlegt, trägt die Zuweisung eines `padding` von 10px den Abständen Rechnung, die im Design zwischen der Umrandung und allen anderen Inhalten, inklusive der horizontalen Trennlinien vorgesehen sind. Nicht berücksichtigt wird u. a. die vorgesehene Strichlänge der Trennlinien, die in CSS nicht festgelegt werden kann:



Quellcode

#verknuepfungen

```
001 #verknuepfungen {
002     color: rgb(179,179,179);
003     border: 1px dashed rgb(179,179,179);
004     margin: 10px;
005     width: 200px;
006     padding: 10px;
007 }
```

Für die Überschrift in `<h2>` müssen wir nur die spezifischen Texteingenschaften zuweisen, die dafür vorgesehen sind. Dazu gehört neben der Setzung der Schriftgröße mittels `font-size` und des Schriftgewichts, das wir hier explizit auf `normal` setzen, insbesondere die Forcierung einer Schreibweise in durchgehenden Großbuchstaben, für die wir die `text-transform` Eigenschaft verwenden. Als Abstand zum nachfolgenden Element fügen wir außerdem nach unten 10px hinzu, während alle anderen Abstände auf 0 gesetzt werden, da sie bereits durch die `padding` Einstellung des umgebenden `<section>` Elements abgedeckt sind:



Quellcode

<h2> Element

```
001 #verknuepfungen h2 {
002     text-transform: uppercase;
003     font-size: 15pt;
004     font-weight: normal;
005     margin: 0px;
006     margin-bottom: 10px;
007 }
```

Für das `` Element, das als Wurzelement der Liste fungiert, müssen wir verhindern, dass die einzelnen Listeneinträge horizontal oder vertikal gegenüber dem Wurzelement eingerückt werden, und weisen daher ein `padding` von 0 zu. Die Setzung von `list-style-type` erlaubt es uns, die Darstellung der Bullet Points zu unterdrücken:



Quellcode

 Element

```
001 #verknuepfungen ul {
002     list-style-type: none;
003     padding: 0px;
004     margin: 0px;
005 }
```

Die spezifische Angabe von `padding` ist auch für die einzelnen Listenelemente erforderlich. Der darin enthaltene Text soll ja zwar nach oben und unten einen hier als 10px gesetzten Abstand zur Trennlinie aufweisen, aber nach links sowie ggf. nach rechts bündig abschließen. Aus diesem Grund wird das `Padding` für diese Richtungen auf 0 gesetzt;



Quellcode

 Element

```
001 #verknuepfungen li {
002     border-top: 1px dashed rgb(179,179,179);
003     padding: 10px;
004     padding-left: 0px;
005     padding-right: 0px;
006 }
```

Die Trennlinien zwischen Überschrift und den einzelnen Listenelementen sowie innerhalb der Listenelemente werden also gleichermaßen dadurch herbeigeführt, dass wir den Elementen eine obere Umrandung zuweisen und den Abstand von 10px zwischen Textinhalt und Trennlinie mittels der `padding` Eigenschaft kontrollieren. Beachten Sie aber, dass das Design auch zwischen dem letzten Listenelement und der Umrandung nur einen Abstand von 10px annimmt. Da wir jedoch für das Wurzelement bereits ein Padding von 10px angegeben haben, würde das für `` angegebene `padding-bottom` hierzu noch hinzuaddiert werden und damit in einem Abstand von 20px resultieren. Aus diesem Grund weisen wir in einer separaten Regel dem letzten Element der Liste einen abweichenden Wert für `padding-bottom` zu, der sich aufgrund der höheren Spezifität gegenüber der Zuweisung der vorstehenden Regel durchsetzt:



Quellcode

<li:last-child> Element

```
001 #verknuepfungen li:last-child {
002     padding-bottom: 0px;
003 }
```

Übrig bleibt uns nun nur noch die Realisierung der `<a>` Elemente innerhalb der Listenelemente. Deren Textfarbe setzen wir durch die Zuweisung von `inherit` auf die Textfarbe des umgebenden Elements, die wir bereits im Wurzelement angegeben haben und überschreiben so den durch den Browser zugewiesenen Default für `<a>` Elemente. Mittels `text-decoration` können wir außerdem die Textunterstreichung für diese Elemente unterbinden. Die anderen Zuweisungen dienen dazu, den Text hinsichtlich Großschreibung, Gewicht und Schriftgröße an die Vorgabe des Designs anzupassen:



Quellcode

<a> Element

```
001 #verknuepfungen a {
002     text-decoration: none;
003     color: inherit;
004     font-weight: bold;
005     text-transform: uppercase;
006     font-size: 20pt;
007 }
```

Mit den hier formulierten Regeln wird das Gestaltungselement für „Verknuepfungen“ schließlich folgendermaßen dargestellt:



Abb.: Gestaltungselement für „Verknuepfungen“

Die Umsetzung der Vorlage zeigt nicht nur, dass die in einer Gestaltungsvorlage vorgesehenen Eigenschaften von Text und Linien nahezu 1:1 in die Erstellung von CSS Regeln übertragen werden können. Auf Grundlage des Box Model von CSS können außerdem Gestaltungsvorlagen für Flächenelemente mit eingebetteten dynamischen Inhalten auf gut nachvollziehbare Weise umgesetzt werden. So ist denn unserer Gestaltung unabhängig davon realisierbar, wie viele Listeneinträge ein „Verknuepfungen“ Element tatsächlich enthält. Ermessensspielraum bei der Erstellung der Regeln besteht insbesondere hinsichtlich der Dimensionen Margin und Padding. Diese müssen letzten Endes so aufeinander abgestimmt werden, dass aus ihrer Kombination die in der Vorlage vorgesehenen Abstände resultieren – unabhängig davon, wie oft ein ggf. vorhandenes dynamisches Element tatsächlich auftritt. Letzteres gilt auch für die Umsetzung von Trennlinien zwischen dynamischen Elementen.


Zusammenfassung

- Wichtigste Voraussetzung von CSS ist die Überlegung, dass die visuelle Darstellung eines HTML-Elements durch eine Menge von Stileigenschaften beschrieben werden kann.
- Stileigenschaften können nicht nur durch explizite Setzung des `style` Attributs zugewiesen werden. Um die Zuweisung von Stileigenschaften an die Elemente einer Menge von HTML-Dokumenten zu ermöglichen, können diese in einem separaten Dokument, einem sogenannten Stylesheet zusammengeführt oder auch über eine Menge von Stylesheets verteilt werden.
- CSS als Regelsprache stellt eine passende Ergänzung von HTML als deklarativer Markupsprache dar.
- Selektoren sind in CSS das wichtigste Ausdrucksmittel zur Formulierung der Vorbedingungen von Regeln für Stileigenschaftszuweisungen.
- Die präzise Definition der Spezifität von Selektoren in CSS ermöglicht es, dass einfache Konflikte bei der Anwendung von CSS Regeln aufgelöst werden können.
- Media Queries erscheinen in besonderer Weise geeignet, um Anforderungen an Webanwendungen umzusetzen, welche ausschließlich oder als eine alternative Nutzungsmöglichkeit auf mobilen Endgeräten ausgeführt werden.

Sie sind am Ende dieser Lerneinheit angelangt. Auf den folgenden Seiten finden Sie noch Übungen.

Übungen

Die Implementierungsbeispiele für die vorliegende Lerneinheit finden Sie im Projekt `org.dieschnittstelle.iam.css`. Die kompletten Beispiele können Sie hier herunterladen:

 `iam_beispiele_komplett.zip` (45 MB)

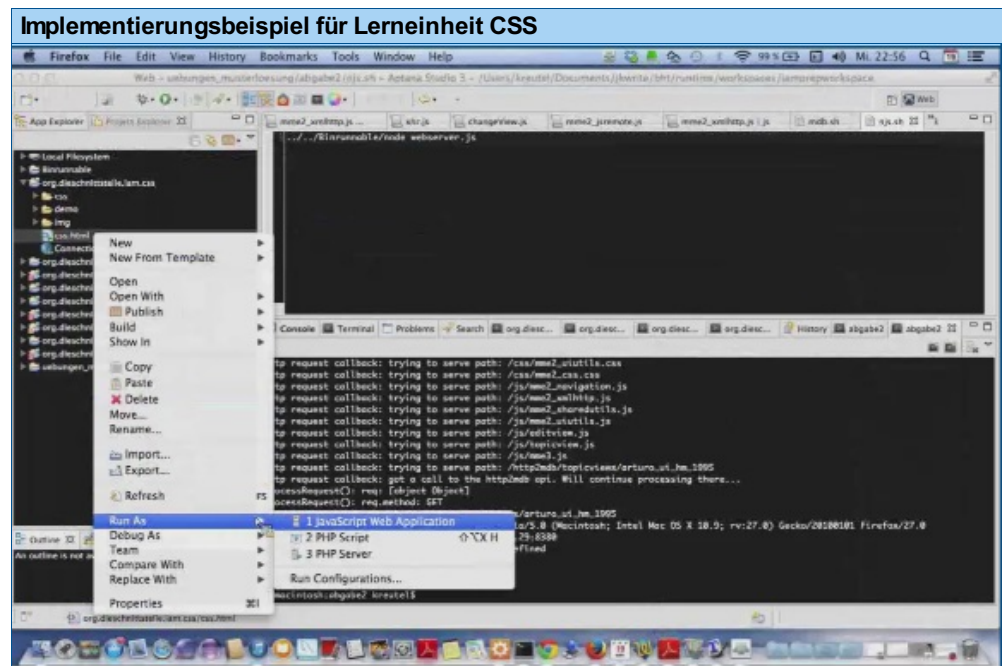
Aus urheberrechtlichen Gründen enthalten die Implementierungsbeispiele andere Abbildungen als im Lehrmaterial gezeigt. Bitte beachten Sie stets das Copyright und stellen Sie sicher, dass geschützte Abbildungen nicht frei über das Internet verfügbar werden - bspw. über ein GIT Repository.

Die prüfungsverbindlichen Übungen und deren Bepunktung werden durch die jeweiligen Lehrenden festgelegt.

Im folgenden kurzem Film zeigt der Autor das Implementierungsbeispiel zur Lerneinheit.



Film



© Beuth Hochschule Berlin - Dauer: 00:43 Min. - Streaming Media 2 MB



Übung CSS-01

Entwicklungswerkzeuge

Ein CSS-Editor mit Codevervollständigung steht Ihnen in Aptana zur Verfügung. Als offizielle Validierungsinstanz können Sie auch für CSS einen Service des W3C nutzen: http://jigsaw.w3.org/css-validator/#validate_by_upload

Firefox bietet Ihnen die Kontextmenü-Aktion Inspect Element an, auf die hin der Firefox Inspector geöffnet wird. Dieser verfügt u. a. über die folgende Funktionalität:

- Visualisierung des aktuellen Zustands des HTML-Dokuments, inklusive aller durch JavaScript vorgenommenen Manipulationen (siehe auch die folgende Lerneinheit), mit WYSIWYG Editierfunktion.
- Auflistung der zur Darstellung des ausgewählten Elements verwendeten CSS-Regeln mit Visualisierung überschriebener Property-Setzungen und WYSIWYG Editierfunktionen (Achtung: Änderungen müssen Sie dann ggf. manuell in die CSS Originaldatei übernehmen).
- Darstellung der Realisierung des Box-Model für das ausgewählte Element.

Die Dokumentation finden Sie unter:

https://developer.mozilla.org/en-US/docs/Tools/Page_Inspector.

Zum Testen verschiedener Bildschirmgrößen können Sie in Firefox die Aktion „Extras/Web-Entwickler/Bildschirmgrößen testen“ verwenden.

Bearbeitungszeit: 20 Minuten



Übung CSS-02

Inspizieren der Beispiele

- Importieren Sie die Implementierungsbeispiele in Ihren Workspace.
- „Starten“ Sie die Datei `css.html` in Aptana (d. h. öffnen Sie die URL `http://127.0.0.1:8020/org.dieschnittstelle.iam.css/css.html`)
- Vergleichen Sie die gewählte HTML-Struktur mit der von Ihnen in Übung HTM-03 umgesetzten. Worin bestehen die wesentlichen Unterschiede?
- Beschäftigen Sie sich mit der Verwendung von CSS zur Darstellung der nachfolgend genannten Elemente.

„Zeitdokumente“

- Wie verändert sich die Darstellung, wenn Sie als Bildquelle `img/hm_mit_bk_tragelehn.jpg` zuweisen?
- Welche Attribute sind für diese Änderung verantwortlich?
- Ist die Zuweisung des Schrifttyps in der Regel für `#zeitdokumente` erforderlich?

„Einführungstext“

- Warum ist die explizite Zuweisung von `font-family` hier erforderlich?
- Wie wird das „Abschneiden“ des Texts umgesetzt?
- Wie erfolgt die Positionierung des Pfeils vor „weiter lesen“?

„Verknüpfungen“ („Mehr dazu...“)

- Welche Strukturelemente werden hierfür verwendet? Sind anwendungsspezifische Elemente (`div`, `span`) erforderlich?
- Welche Elemente verwenden die gestrichelte Linie als Umrandungs- bzw. Trennlinie?
- Wie wird die Bündigkeit von Trennlinie und der Bezeichnung der Optionen „Ort“ und „Zeit“ umgesetzt, und wie der gemeinsame Abstand zur Umrandung?
- Welche Property ist für die Großschrift der Überschrift verantwortlich?
- Wie wird das Unterstreichen der Links in den `<a>`-Elementen unterbunden?

Bearbeitungszeit: 45 Minuten



Übung CSS-03

Einzelne Gestaltungselemente

Aufgabe

Setzen Sie die Elemente „Medienverweise“, „Textauszug“, „Imgbox“ entsprechend den nachfolgend skizzierten und umseitig abgebildeten Gestaltungsvorgaben um.

Anforderungen

1. Medienverweise

- (a) Überschrift
- (b) Titelliste
- (c) Hintergrund
- (d) Trennlinien mit Rand
- (e) Pfeil

2. Textauszug

- (a) Schneiden Sie unter Verwendung von CSS längere Texte nach unten ab
- (b) Verwenden Sie anwendungsspezifisches „semantisches Markup“ für die Textstruktur
- (c) Titel
- (d) Text
- (e) Verknüpfung mit Pfeil

3. Objekt

- (a) Titel
- (b) Strichellinie
- (c) Kringel
- (d) Bild

Bearbeitungshinweise

- Das Projekt `org.dieschnittstelle.iam.css.uebungen` können Sie als Ausgangspunkt für diese Aufgabe verwenden. In der folgenden Lerneinheit werden Sie dieses Projekt erweitern.
- Ändern Sie für die Umsetzung ggf. die von Ihnen in Übung HTM-03 vorgesehene Dokumentstruktur.
- **Anforderung 2:** ein Beispiel für das geforderte „Abschneiden“ finden Sie in der Umsetzung des Elements „Einführungstext“ in den aktuellen Beispielen.
- **Anforderung 3:** verwenden Sie in „Imgbox“ ein Bild Ihrer Wahl, ggf. auch die in den Implementierungsbeispielen genutzten Bilder.
- Halbieren Sie die Größenangaben aus den Gestaltungsvorgaben (diese sind für ein „Retina“-Display angenommen) und nehmen Sie ggf. weitere Änderungen nach Augenmaß vor.
- **Achtung:** die Raster dienen der Angabe von Maßen und sind nicht umzusetzen. Schattierungen sind gleichermaßen zu vernachlässigen.
- Bildmaterial zur Realisierung von Pfeilen, Kringeln und Kreuzen finden Sie im Verzeichnis `css/img` des Projekts `org.dieschnittstelle.iam.css.uebungen`.

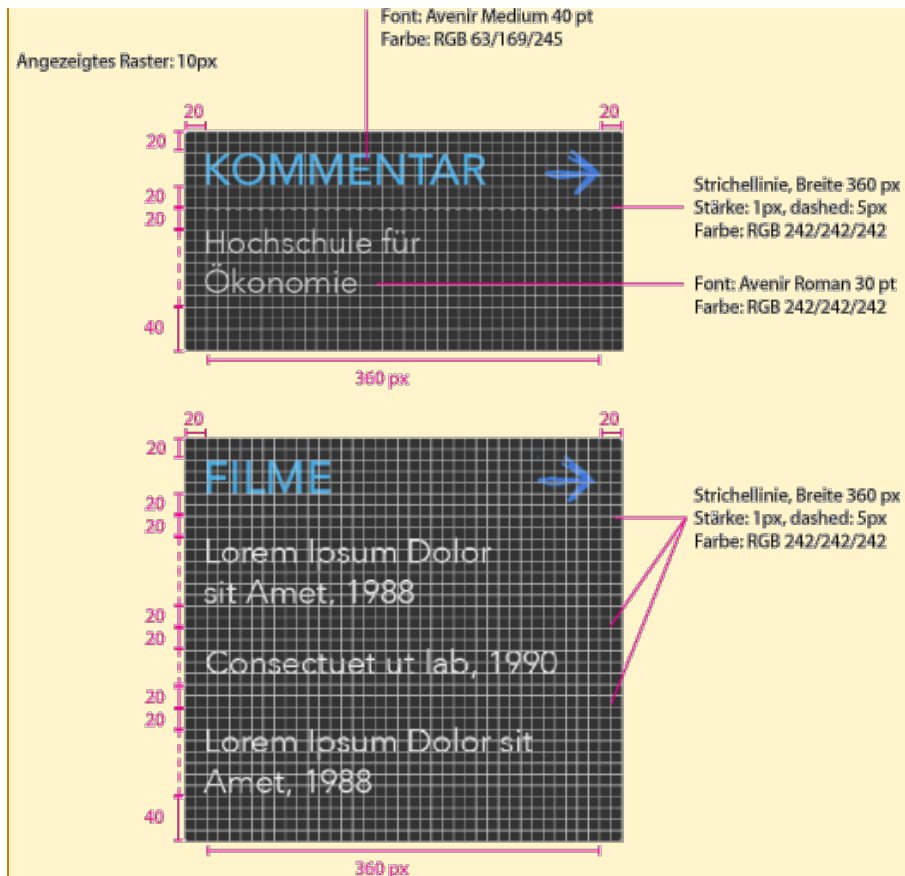
Sonstiges

Vorlage „Medienverweise“

Anm.: Objektbreite ist fix. Objekthöhe ist variabel.

Die Hintergrundfarbe ist RGB 45/45/45.

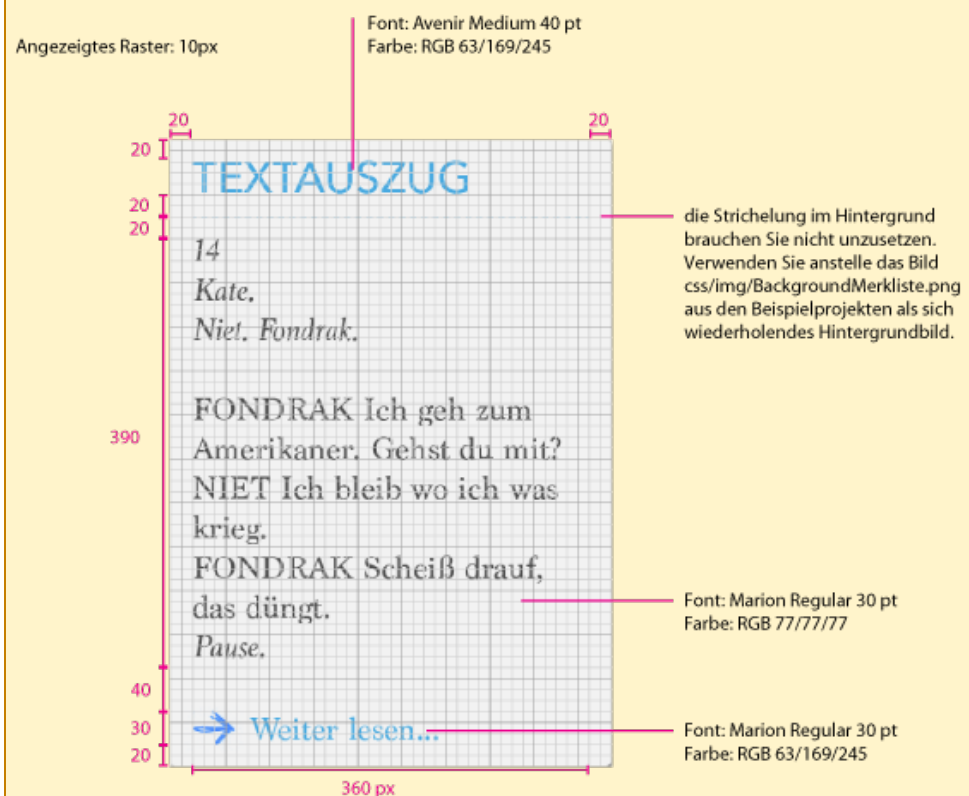
Der Titel („Kommentar“, „Filme“, etc.) ist variabel.



Vorlage „Textauszug“

Anm.: Objektbreite und Objekthöhe sind fix.

Der Titel („Textauszug“) ist fix. Textinhalt und Textlänge sind variabel.



Vorlage „Imbox“ (nicht zu verwechseln mit dem Element „Zeitobjekt“)

Anm.: Objektbreite und Objekthöhe sind variabel (bis jeweils 560px gesamt).

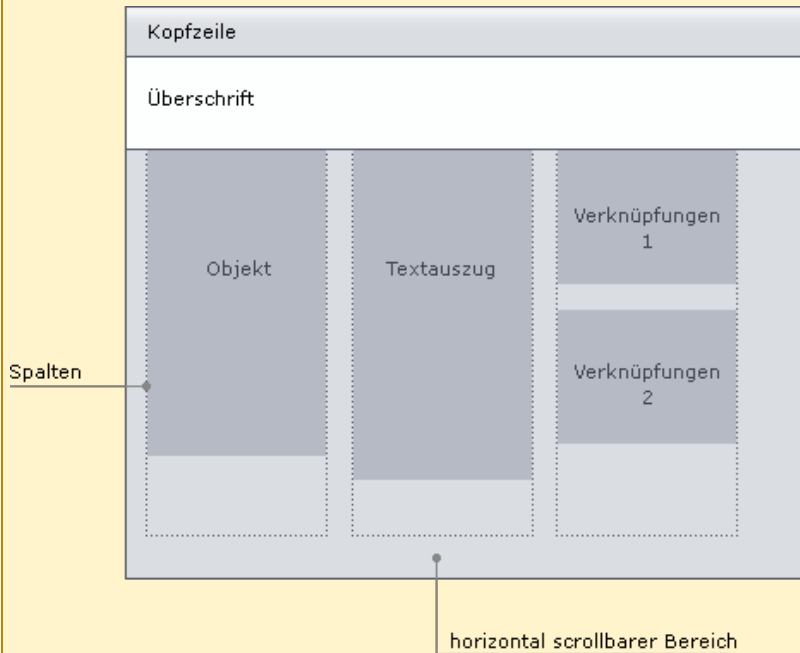
Der Titel („Objekt“) ist fix. Dargestellt wird immer eine Abbildung.





Übung CSS-04

Realisierung des Gesamtlayout



Aufgabe

Stellen Sie die drei in Übung „CSS-03 “ bearbeiteten Elemente, wie oben gezeigt, nebeneinander in einem Spaltenlayout dar.

Anforderungen

1. Vom Elementtyp „Medienverweise“ sollen wie in der Abbildung zur Übung HTM-03 zwei Exemplare verwendet werden.
2. Die Elemente sollen, wie in der Gestaltungsvorlage vorgesehen, nach oben bündig abschließen.
3. Die linksaußen platzierte Spalte soll bündig mit der Überschrift („Die Umsiedlerin“) abschließen.
4. Der Abstand zwischen den Spalten soll 20px betragen.
5. Verhindern Sie, dass bei Verkleinerung der Breite des Browserfensters die Anordnung der Spalten geändert wird.
6. Ermöglichen Sie, dass bei Verkleinerung der Breite des Browserfensters die Ansicht der Spalten insgesamt horizontal scrollbar ist, ohne dass der Titel der Ansicht („Die Umsiedlerin“) und die Kopfzeile beim Scrollen verschoben wird.

Bearbeitungshinweise

- **Anforderung 5:** dafür können Sie z. B. dem Mutter-Element der Spalten eine fixe Breite zuweisen. Anhaltspunkte hierfür finden Sie auch in den Implementierungsbeispielen zu den JSL Übungen.
- **Anforderung 6:** Ziehen Sie dafür z. B. in Betracht, ein zusätzliches div Element einzuführen, das mit der `overflow` Property für horizontales Scrolling versehen wird.
- **Anforderung 6:** Der Punkt bezüglich der Fixierung von Kopfzeile und Überschrift wird nur vergeben, wenn horizontales Scrolling wie beschrieben möglich ist.

Bearbeitungszeit: 60 Minuten

Wissensüberprüfung

Versuchen die hier aufgeführten Fragen zu den Inhalten der Lerneinheit selbständig kurz zu beantworten, bzw. zu skizzieren. Wenn Sie eine Frage noch nicht beantworten können kehren Sie noch einmal auf die entsprechende Seite in der Lerneinheit zurück und versuchen sich die Lösung zu erarbeiten.



Formulieren

Übung CSS-05

Style

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Wie können ohne Verwendung von CSS `Style-Properties` auf einem HTML Element gesetzt werden?
2. Nennen Sie 4 Gestaltungsmerkmale, hinsichtlich derer die Darstellung von HTML Elementen mittels CSS kontrolliert werden kann.
3. Wofür werden `Vendor-Prefixes` in CSS verwendet? Nennen Sie ein Beispiel Prefix.

Bearbeitungszeit: 30 Minuten



Formulieren

Übung CSS-06

Regeln

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Was ist die allgemeine Form einer Regel in einer Regelsprache und die spezifische Form einer CSS Regel?
2. Welche Entscheidungsprobleme können bei Verwendung einer Regelsprache bezüglich der auszuführenden Aktion auftreten?
3. Wie sollten Regeln idealerweise formuliert werden?

Bearbeitungszeit: 30 Minuten



Formulieren

Übung CSS-07

CSS als Regelsprache

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Von welcher Art sind die Entscheidungen, die CSS als Regelsprache ermöglichen soll?
2. Inwiefern ergänzen sich HTML und CSS?
3. Welche beiden Ausdrucksmittel stellt CSS zur Formulierung der Vorbedingungen von Regeln zur Verfügung?
4. Wie löst CSS das Problem, dass die Vorbedingung von mehr als einer CSS Regel zutreffen kann

Bearbeitungszeit: 30 Minuten



Formulieren

Übung CSS-08

Selektoren

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Was tut ein Selektor?
2. Wie können Elemente in HTML Dokumenten identifiziert werden?
3. Über welche einfachen Selektoren verfügt CSS?
4. Was wird durch ein Leerzeichen zwischen zwei CSS Selektoren zum Ausdruck gebracht?
5. Welches Zeichen fungiert in CSS als Disjunktionsoperator („oder“)?
6. Wie kann eine Negation in einem CSS Selektor ausgedrückt werden?
7. Wofür werden Pseudoklassen in CSS verwendet?
8. Wie wird die Spezifität eines Selektors ermittelt?
9. Weshalb ist es nicht ausgeschlossen, dass einem Element inkonsistente `style`-Properties zugewiesen werden?
10. Kann ein Nutzer die durch einen Entwickler gesetzten `style`-Properties „überschreiben“?

Bearbeitungszeit: 30 Minuten



Formulieren

Übung CSS-09

Media Queries

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Was ermöglichen Media Queries in Bezug auf die Anwendung von CSS Regeln?
2. Welche Kategorien von Eigenschaften können in Media Queries überprüft werden?
3. Wie verhalten sich Media Queries zur Spezifitätshierarchie für Selektoren und wie erfolgt ihre Auswertung?

Bearbeitungszeit: 30 Minuten



Formulieren

Übung CSS-10

CSS Ausdrucksmittel

Versuchen die hier aufgeführten Fragen selbständig kurz zu beantworten, bzw. zu skizzieren.

1. Was wird als „Vererbung“ in CSS bezeichnet?
2. Was wird durch die `display`-Property zum Ausdruck gebracht?
3. Was können Sie tun, um die Darstellung eines Elements mittels CSS zu unterbinden?
4. Welche Möglichkeiten haben Sie zur Angabe von Farbwerten in CSS?
5. Nennen Sie von innen nach außen die 4 Boxen des CSS-Box-Modells.
6. Werden Tochterelemente eines HTML Elements entsprechend der Reihenfolge ihres Auftretens in horizontaler oder vertikaler Anordnung dargestellt?

Bearbeitungszeit: 30 Minuten