KAPITEL 3

Selektoren und Queries

Selektoren

Universal-Selektor

Muster

*

Beschreibung

Dieser Selektor passt auf den Namen eines beliebigen Elements innerhalb der Dokumentsprache. Hat eine Regel keinen expliziten Selektor, wird automatisch der universelle Selektor benutzt.

Beispiele

```
* {color: red;}
div * p {color: blue;}
```

Typ-Selektor

Muster

```
element1
```

Beschreibung

Dieser Selektor passt auf den Namen eines Elements in der Dokumentsprache (z.B. HTML). Er passt auf alle Vorkommen des Elements. (In CSS1 wurde der Typ-Selektor noch als »Element-Selektor« bezeichnet.)

Beispiele

```
body {background: #FFF;}
p {font-size: 1em;}
```

Nachfahren-Selektor

Muster

`element1 element2 ...`

Beschreibung

Findet Elemente basierend auf ihrem Status als Nachfahre eines anderen Elements. Das gefundene Element kann Kind, Enkel, Großenkel etc. des Vorfahrenelements sein. (In CSS1 wurde der Nachfahren-Selektor noch als »Kontext-Selektor« bezeichnet.)

Beispiele

```
body h1 {font-size: 200%;}
table tr td div ul li {color: purple;}
```

Kind-Selektor

Muster

`element1 > element2`

Beschreibung

Findet Elemente, die direkte Nachfahren eines anderen Elements sind (Kindelemente). Dieser Selektor ist restriktiver als der Nachfahren-Selektor, da er ausschließlich auf Kindelemente passt.

Beispiele

```
div > p {color: cyan;}
ul > li {font-weight: bold;}
```

Selektor für benachbarte Geschwisterelemente

Muster

`element1 + element2`

Beschreibung

Passt auf Geschwisterelemente, die direkt auf ein anderes Element folgen. (Geschwisterelemente haben, wie der Name schon sagt, ein gemeinsames Elternelement.) Jeglicher anonymer Text zwischen den Elementen wird ignoriert. Es werden nur Elemente und ihre Position innerhalb der Dokumenthierarchie berücksichtigt.

Beispiele

```
table + p {margin-top: 2.5em;}
h1 + * {margin-top: 0;}
```

Allgemeiner Selektor für Geschwisterelemente

Muster

```
element1 ~ element2
```

Beschreibung

Passt auf nachfolgende Geschwisterelemente eines Elements (auf der gleichen Hierarchieebene). Jeglicher Text oder andere Elemente zwischen den Geschwisterelementen werden ignoriert. Es werden nur Elemente und ihre Position innerhalb der Dokumenthierarchie berücksichtigt.

Beispiele

```
h1 ~ h2 {margin-top: 2.5em;}
nav a ~ a {border-left: 1px solid border;}
```

Klassen-Selektor

Muster

```
element1.klassenname
element1.klassenname1.klassenname2
```

Beschreibung

In unterstützten Sprachen (z.B. HTML, SVG, MathML) findet ein Klassen-Selektor Elemente, deren `class`-Attribut einen oder mehrere bestimmte Werte hat. Dies geschieht anhand der Punktschreibweise. Hierbei muss der Klassenname unmittelbar auf den Punkt folgen. Mehrere Klassennamen können miteinander verkettet werden. Wurde dem Klassennamen kein Elementname vorangestellt, passt der Selektor auf alle Elemente, deren `class`-Attribut den oder die angegebenen Werte haben.

Beispiele

```
p.urgent {color: red;}
a.external {font-style: italic;}
.example {background: olive;}
.note.caution {background: yellow;}
```

ID-Selektor

Muster

`element1#idname`

Beschreibung

In unterstützten Sprachen, z. B. HTML oder SVG, findet ein ID-Selektor Elemente, deren `id`-Attribut einen bestimmten Wert hat. Der Name des ID-Werts muss unmittelbar auf die Raute (#) folgen. Steht vor der Raute kein Elementname, passt der Selektor auf alle Elemente, die diesen ID-Wert enthalten. Gemäß der HTML5-Spezifikation muss der Wert des `id`-Attributs im aktuellen Dokument *einmalig* sein.

Beispiele

```
h1#page-title {font-size: 250%;}
body#home {background: silver;}
#example {background: lime;}
```

Einfacher Attribut-Selektor

Muster

`element1[attr]`

Beschreibung

Passt auf Elemente mit einem bestimmten Attribut, unabhängig vom Wert des Attributs.

Beispiele

```
a[rel] {border-bottom: 3px double gray;}
p[class] {border: 1px dotted silver;}
```

Selektor für exakte Attributwerte

Muster

`element1[attr="wert"]`

Beschreibung

Findet Elemente mit genau dem angegebenen Attribut und dem vollständigen Attributwert.

Beispiele

```
a[rel="start"] {font-weight: bold;}
p[class="urgent"] {color: red;}
```

Selektor für teilweise Attributwerte

Muster

```
element1[attr~="wert"]
```

Beschreibung

Mithilfe dieses Selektors kann ein beliebiges Element anhand eines Teils der durch Leerzeichen getrennten Attributwerte ausgewählt werden. Die Formulierung `[class~="wert"]` ist gleichbedeutend mit *`.wert`* (siehe oben).

Beispiele

```
a[rel~="friend"] {text-transform: uppercase;}
p[class~="warning"] {background: yellow;}
```

Selektor für Substrings am Anfang eines Attributwerts

Muster

```
element1[attr^="substring"]
```

Beschreibung

Findet Elemente basierend auf dem angegebenen Substring am Anfang des Attributwerts, z.B. `<a href="/blog/14.09.2018/">Test</a>` (oberes Beispiel) oder `<p class="test-umgebung">Test</p>` (unteres Beispiel).

Beispiele

```
a[href^="/blog"] {text-transform: uppercase;}
p[class^="test-"] {background: yellow;}
```

Selektor für Substrings am Ende eines Attributwerts

Muster

```
element1[attr$="substring"]
```

Beschreibung

Findet Elemente basierend auf dem angegebenen Substring am Ende des Attributwerts.

Beispiel

```
a[href$=".pdf"] {font-style: italic;}
```

Selektor für einen Substring an beliebiger Stelle im Attributwert

Muster

```
element1[attr*="substring"]
```

Beschreibung

Findet Elemente basierend auf dem angegebenen Substring an beliebiger Stelle im Attributwert.

Beispiele

```
a[href*="oreilly.com"] {font-weight: bold;}
div[class*="port"] {border: 1px solid red;}
```

Selektor für Sprachattribute

Muster

```
element1[lang|="language-identifier"]
```

Beschreibung

Passt auf Elemente mit einem `lang`-Attribut. Der Wert des Attributs ist eine durch Bindestriche getrennte Werteliste, beginnend mit dem im Selektor angegebenen Wert.

In einem HTML-Dokument wird die Sprache eines Elements durch sein `lang`-Attribut bestimmt. Fehlt einem Element dieses Attribut, wird seine Sprache durch das `lang`-Attribut des nächstgelegenen Vorfahrenelements mit `lang`-Attribut bestimmt. Fehlt auch dieses, wird die Sprache durch das `Content-Language`-Feld im HTTP-Antwort-Header (bzw. dem entsprechenden `meta http-equiv`-Feld) des Dokuments festgelegt.

Beispiel

```
html[lang|="tr"] {color: red;}
```

Strukturelle Pseudoklassen

Streng genommen sind alle Pseudoklassen (und -Selektoren) strukturell. Schließlich hängen sie alle auf die eine oder andere Art von der Dokumentstruktur ab. Im Unterschied zu anderen Selektoren

handelt es sich bei den hier aufgelisteten Pseudoklassen allerdings um *Muster in der Dokumentstruktur*. Muster sind beispielsweise »jeder zweite Absatz« oder »Elemente, die das letzte Kindelement ihres Elternelements sind«.

:empty

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf Elemente, die keine Kindknoten, das heißt keine Kindelemente *oder* Inhaltsknoten besitzen. Inhaltsknoten sind definiert als beliebiger Text, Leerraum, Entity-Referenzen oder CDATA-Knoten. Demnach ist `<p> </p>` *nicht* leer, da es ein einzelnes Leerzeichen enthält. Das Element wäre auch dann nicht leer, wenn das Leerzeichen durch einen Zeilenumbruch ersetzt würde. Beachten Sie, dass diese Pseudoklasse nicht auf leere Elemente wie `<br>`, `<img>`, `<input>` etc. passt.

Beispiele

```
p:empty {padding: 1em; background: red;}
div:not(:empty) {border: 1px solid;
   padding: 1ch;}
li:empty {display: none;}
```

:first-child

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf ein Element, das das erste Kindelement eines anderen Elements ist. `div:first-child` wählt beispielsweise jedes `div`-Element aus, das das erste Kindelement eines anderen Elements ist, allerdings *nicht* das erste Kindelement eines `div`.

Beispiele

```
td:first-child {border-left: 1px solid;}
p:first-child {text-indent: 0; margin-top: 2em;}
```

:first-of-type

Anwendbar auf

Jedes Element.

Beschreibung

Wählt ein Element aus, wenn es das erste Kindelement seiner Art eines anderen Elements ist. So wählt `div:first-of-type` beispielsweise jedes `div` aus, das das erste Element vom Typ `div` unter den Kindelementen eines anderen Elements ist.

Beispiele

```
td:first-of-type {border-left: 1px dotted;}
h2:first-of-type {color: fuchsia;}
```

:lang

Anwendbar auf

Jedes Element mit Informationen zur verwendeten Sprachkodierung.

Beschreibung

Findet Elemente anhand ihrer natürlichen Sprachkodierung. Die Sprachinformationen müssen innerhalb des Dokuments vorliegen oder auf andere Weise mit dem Dokument verbunden sein. Diese Sprachinformationen lassen sich nicht per CSS zuweisen. Die Behandlung von `:lang` ist die gleiche wie für den Attribut-Selektor `|=`.

Beispiele

```
html:lang(en) {background: silver;}
*:lang(fr) {quotes: '&#171;' '&#187;';}
```

:last-child

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf ein Element, das das letzte Kindelement eines anderen Elements ist. `div:last-child` wählt beispielsweise jedes `div`-Ele-

ment aus, das das letzte Kindelement eines anderen Elements ist, allerdings *nicht* das letzte Kindelement eines `div`.

Beispiele

```
td:last-child {border-right: 1px solid;}
p:last-child {margin-bottom: 2em;}
```

:last-of-type

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf ein Element, das das letzte Kindelement seines Typs eines anderen Elements ist. `div:last-of-type` wählt beispielsweise jedes `div`-Element aus, das das erste Kindelement eines anderen Elements ist, allerdings *nicht* das letzte Kindelement seines Typs eines `div`.

Beispiele

```
td:last-of-type {border-right: 1px dotted;}
h2:last-of-type {color: fuchsia;}
```

:nth-child(*an*±*b*)

Anwendbar auf

Jedes Element.

Beschreibung

Wählt anhand des Auswahlmusters jedes *n*te Kindelement aus. Hierfür wird die Formel *a*n + *b* verwendet. *a* und *b* sind *<Integerwerte>*, n steht für eine beliebig lange Folge von Integerwerten, gezählt vom ersten Kindelement. Um z.B. jedes vierte Kindelement des `body`-Elements, beginnend beim ersten Kindelement, auszuwählen, könnten Sie schreiben: `body > *:nth-child(4n+1)`. Hierdurch wird das erste, fünfte, neunte, dreizehnte etc. Kindelement von `body` ausgewählt.

Wollen Sie das vierte, achte, zwölfte etc. Kindelement auswählen, können Sie den Selektor folgendermaßen anpassen: `body > *:nth-child(4n)`. *b* darf auch negativ sein: `body >`

`*:nth-child(4n-1)` wählt das dritte, siebte, elfte, fünfzehnte etc. Kindelement von `body` aus.

Anstelle der Formel *an±b* sind außerdem die zwei Schlüsselwörter `even` (gerade) und `odd` (ungerade) erlaubt. Diese entsprechen `2n` bzw. `2n+1`.

Beispiele

```
*:nth-child(4n+1) {font-weight: bold;}
tbody tr:nth-child(odd) {background-color: #EEF;}
```

:nth-last-child(*an±b*)

Anwendbar auf

Jedes Element.

Beschreibung

Wählt anhand des Auswahlmusters jedes *n*te Kindelement aus. Hierfür wird die Formel *an±b* verwendet. *a* und *b* sind <*Integerwerte*>, n steht für eine beliebig lange Folge von Integerwerten, rückwärts gezählt, ausgehend vom letzten Kindelement. Um beispielsweise jedes viertletzte Kindelement von `body`, beginnend beim letzten Kindelement, auszuwählen, können Sie schreiben: `body > *:nth-last-child(4n+1)`. Dies ist sozusagen das Spiegelbild von `:nth-child`.

Anstelle der Formel *an±b* sind außerdem die zwei Schlüsselwörter `even` (gerade) und `odd` (ungerade) erlaubt. Diese entsprechen `2n` bzw. `2n+1`.

Beispiele

```
*:nth-last-child(4n+1) {font-weight: bold;}
tbody tr:nth-last-child(odd) {
   background-color: #EEF;}
```

:nth-last-of-type(*an±b*)

Anwendbar auf

Jedes Element.

Beschreibung

Wählt jedes *n*te Kindelement aus, das den gleichen Typ hat wie das angegebene Element. Für das Auswahlmuster wird die For-

mel *a*n±*b* verwendet. *a* und *b* sind <*Integerwerte*>, n steht für eine beliebig lange Folge von Integerwerten, rückwärts gezählt vom letzten Element des angegebenen Typs. Um beispielsweise den drittletzten Absatz (`p`) auszuwählen, der ein Kindelement von `body` ist, beginnend beim letzten Absatz dieses Typs, könnten Sie schreiben: `body > p:nth-last-of-type(3n+1)`. Das funktioniert selbst, wenn sich andere Elemente (z.B. Listen, Tabellen etc.) zwischen den verschiedenen Absätzen befinden.

Anstelle der Formel *a*n±*b* sind außerdem die zwei Schlüsselwörter `even` (gerade) und `odd` (ungerade) erlaubt. Diese entsprechen `2n` bzw. `2n+1`.

Beispiele

```
td:nth-last-of-type(even) {
   background-color: #FCC;}
img:nth-last-of-type(3n) {float: left;
   border: 2px solid;}
```

:nth-of-type(*a*n±*b*)

Anwendbar auf

Jedes Element.

Beschreibung

Wählt jedes *n*te Kindelement aus, das den gleichen Typ hat wie das angegebene Element. Für das Auswahlmuster wird die Formel *a*n±*b* verwendet. *a* und *b* sind <*Integerwerte*>, n steht für eine beliebig lange Folge von Integerwerten, gezählt vom ersten Element des angegebenen Typs. Um beispielsweise, beginnend beim ersten Absatz, jeden dritten Absatz (`p`) auszuwählen, der ein Kindelement von `body` ist, könnten Sie schreiben: `body > p:nth-of-type(3n+1)`. Damit wird der erste, vierte, siebte, zehnte etc. Absatz ausgewählt, der ein Kindelement von `body` ist. Das funktioniert selbst, wenn sich andere Elemente (z.B. Listen, Tabellen etc.) zwischen den verschiedenen Absätzen befinden.

Anstelle der Formel *a*n±*b* sind außerdem die zwei Schlüsselwörter `even` (gerade) und `odd` (ungerade) erlaubt. Diese entsprechen `2n` bzw. `2n+1`.

Beispiele

```
td:nth-of-type(even) {background-color: #FCC;}
img:nth-of-type(3n) {float: right;}
```

:only-child

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf ein Element, das das einzige Kindelement eines direkten Vorfahren (Elternelements) ist. Ein häufiger Anwendungsfall für diesen Selektor ist das Entfernen des Rahmens eines verlinkten Bilds, wenn das Bild das einzige Element im Link ist. Ein Element kann auch dann ausgewählt werden, wenn es selbst ein oder mehrere Kindelemente besitzt. Es muss einfach nur das einzige Kind seines Elternelements sein.

Beispiele

```
a img:only-child {border: 0;}
table div:only-child {margin: 5px;}
```

:only-of-type

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf ein Element, das das einzige Element dieses Typs innerhalb seines Elternelements ist. Ein Element kann auch dann per `:only-of-type` ausgewählt werden, wenn es selbst ein oder mehrere Kindelemente des gleichen Typs besitzt (z. B. ein oder mehrere `div`s innerhalb eines `div`).

Beispiele

```
p em:only-of-type {font-weight: bold;}
section article:only-of-type {margin: 2em 0 3em;}
```

:root

Anwendbar auf

Das Wurzelelement.

Beschreibung

Passt auf das Wurzelelement eines Dokuments. In HTML ist dies grundsätzlich das `html`-Element, in SVG ist es das `svg`-Element. In XML-Formaten kann das Wurzelelement einen beliebigen Namen haben. Daher wird ein allgemeiner Selektor für das Wurzelelement gebraucht.

Beispiele

```
:root {font: medium serif;}
:root > * {margin: 1.5em 0;}
```

Die Negations-Pseudoklasse

Es gibt nur eine Pseudoklasse für Negationen. Die ist allerdings so speziell, dass sie einen eigenen Unterabschnitt verdient.

:not(*e*)

Anwendbar auf

Jedes Element.

Beschreibung

Passt auf jedes Element, das *nicht* durch den »einfachen Selektor« *e* beschrieben wird. Um beispielsweise alle Elemente auszuwählen, die *kein* Absatz sind, können Sie schreiben: `*:not(p)`.

Noch nützlicher wird die Verneinung im Zusammenhang mit Nachfahren-Selektoren. Ein Beispiel ist die Verwendung von `table *:not(td)`, um jedes Element in einer Tabelle auszuwählen, das *keine* Datenzelle ist. Ein weiteres Beispiel ist die Auswahl aller Elemente, die eine andere ID haben als `search`, mit dem Selektor `[id]:not([id="search"])`.

Zur Definition des »einfachen Selektors« für *e* gibt es eine Ausnahme: Es kann nicht die Negations-Pseudoklasse selbst sein. Es ist also nicht erlaubt, zu schreiben: `:not(:not(div))`.

Da `:not()` eine Pseudoklasse ist, kann sie mit anderen Pseudoklassen und weiteren Instanzen seiner selbst verkettet werden. Um beispielsweise jedes Element auszuwählen, das den

Fokus hat und kein a-Element ist, können Sie schreiben: `*:focus:not(a)`. Um jedes Element auszuwählen, das weder ein Absatz (`<p>`) noch ein Abschnitt (`<section>`) ist, verwenden Sie `*:not(p):not(section)`.

Im Frühjahr 2018 bedeutete die Beschränkung auf »einfache Selektoren«, dass gruppierte und kombinierte Selektoren sowie Selektoren für Nachfahrenelemente in `:not()`-Ausdrücken nicht erlaubt sind. Diese Beschränkung soll in der Spezifikation *CSS Selectors Level 4* jedoch gelockert werden.

Beispiele

```
ul *:not(li) {text-indent: 2em;}
*:not([type="checkbox"]):not([type="radio"]) {
   margin: 0 1em;}
```

Interaktions-Pseudoklassen

Die hier aufgelisteten Pseudoklassen haben alle mit der Interaktion zwischen dem Benutzer und dem Dokument zu tun: Es geht um Stildefinitionen für unterschiedliche Link-Zustände, die Hervorhebung eines Elements, das Ziel eines Fragment-Identifiers ist, oder um Stildefinitionen für Formularelemente, und zwar abhängig davon, ob sie aktiviert oder deaktiviert sind.

:active

Anwendbar auf

Jedes Interaktionselement.

Beschreibung

Passt auf ein Element, während es aktiviert wird. Das am häufigsten vorkommende Beispiel ist das Anklicken eines Links in einem HTML-Dokument: Während die Maustaste gedrückt ist, ist der Link aktiv. Es gibt weitere Möglichkeiten, ein Element zu aktivieren, und theoretisch können auch andere Elemente aktiviert werden. Diese sind in CSS aber nicht definiert.

Beispiele

```
a:active {color: red;}
*:active {background: blue;}
```

:checked

Anwendbar auf

Jedes Interaktionselement, das einen An-/Auszustand hat.

Beschreibung

Passt auf ein beliebiges Benutzeroberflächenelement, das »eingeschaltet« wurde, z. B. ein markiertes Ankreuzfeld oder einen aktivierten (»gefüllten«) Radiobutton.

Beispiele

```
input:checked {
   outline: 3px solid rgba(127,127,127,0.5);}
input[type="checkbox"]:checked {
   box-shadow: red 0 0 5px;}
```

:disabled

Anwendbar auf

Jedes Interaktionselement.

Beschreibung

Passt auf Elemente der Benutzeroberfläche, die aufgrund von Sprachattributen oder anderen von der Darstellung unabhängigen Faktoren keine Benutzereingaben entgegennehmen können, wie beispielsweise `<input type="text" disabled>` in HTML5. Beachten Sie, dass `:disabled` *nicht* auf `input`-Elemente passt, die durch Eigenschaften wie `position` oder `display` nur aus dem Ansichtsbereich entfernt wurden.

Beispiel

```
input:disabled {opacity: 0.5;}
```

:enabled

Anwendbar auf

Jedes Interaktionselement.

Beschreibung

Passt auf Elemente der Benutzeroberfläche, die Benutzereingaben entgegennehmen können und die durch die Markup-Sprache mit Zuständen wie »enabled« (benutzbar) und »disabled«

(nicht benutzbar) versehen werden können. Hierzu gehören sämtliche HTML-Eingabeelemente, aber keine Hyperlinks.

Beispiel

```
input:enabled {background: #FCC;}
```

:focus

Anwendbar auf

Jedes Interaktionselement.

Beschreibung

Passt auf ein Element, während es den Fokus hat. Ein HTML-Beispiel ist ein Texteingabefeld, das den Eingabecursor enthält, sodass der Benutzer Text in das Feld eingeben kann. Andere Elemente, z.B. Hyperlinks, können ebenfalls den Fokus erhalten, allerdings definiert CSS nicht ausdrücklich, welche das sind.

Beispiele

```
a:focus {outline: 1px dotted red;}
input:focus {background: yellow;}
```

:hover

Anwendbar auf

Jedes Interaktionselement.

Beschreibung

Passt auf ein Element im *hover*-Zustand (wenn sich der Cursor über dem Element befindet, ohne dass dieses aktiviert wird). Das bekannteste Beispiel dafür ist das Bewegen des Mauszeigers über einen Hyperlink in einem HTML-Dokument. Andere Elemente können theoretisch ebenfalls einen *hover*-Zustand haben, allerdings definiert CSS nicht, welche dies sind.

Beispiele

```
a[href]:hover {text-decoration: underline;}
p:hover {background: yellow;}
```

:link

Anwendbar auf

Einen Hyperlink auf eine noch nicht besuchte Ressource.

Beschreibung

Passt auf einen Link zu einem URI, der noch nicht besucht wurde, das heißt, der URI befindet sich nicht im Verlauf des Benutzerprogramms. Dieser Zustand und der Zustand `:visited` schließen sich gegenseitig aus.

Beispiele

```
a:link {color: blue;}
*:link {text-decoration: underline;}
```

:target

Anwendbar auf

Jedes Element.

Beschreibung

Findet ein Element, auf das seinerseits der Fragment-Identifier-Teil des URI passt, mit dem auf die Seite zugegriffen wurde. So passt `:target` beispielsweise auf `https://www.w3.org/TR/css3-selectors/#target-pseudo`, und die deklarierten Stile würden auf alle Elemente angewandt, deren `id`-Attribut den Wert `target-pseudo` hat. Ist das Element ein Absatz, erzielt auch der Selektor `p:target` einen Treffer.

Beispiel

```
:target {background: #EEO;}
```

:visited

Anwendbar auf

Einen Hyperlink auf eine bereits besuchte Ressource.

Beschreibung

Passt auf einen Link zu einem bereits besuchten URI. Der URI, auf den der Link verweist, befindet sich im Verlauf des Benutzerprogramms. Dieser Zustand und der Zustand `:link` schließen sich gegenseitig aus.

Beispiele

```
a:visited {color: purple;}
*:visited {color: gray;}
```

Pseudoelemente

In CSS1 und CSS2 wurden Pseudoelementen, wie Pseudoklassen, einzelne Doppelpunkte vorangestellt. In CSS3 und später werden Pseudoelemente durch doppelte Doppelpunkte eingeleitet, um sie von Pseudoklassen zu unterscheiden. Aus historischen Gründen unterstützen Browser beide Schreibweisen. Dennoch wird die Syntax mit den doppelten Doppelpunkten empfohlen.

::after

Erzeugt

Ein Pseudoelement für erzeugte Inhalte, das nach dem Inhalt eines regulären Elements platziert wird.

Beschreibung

Fügt am Ende des Inhalts eines Elements erzeugten Inhalt ein. Standardmäßig ist `::after` ein Inline-Element. Dies kann anhand der Eigenschaft `display` verändert werden.

Beispiele

```
a.external:after {
   content: " " url(/icons/globe.gif);}
p:after {content: " |; ";}
```

::before

Erzeugt

Ein Pseudoelement für erzeugten Inhalt, das vor dem Inhalt eines regulären Elements platziert wird.

Beschreibung

Fügt am Anfang des Inhalts eines Elements erzeugten Inhalt ein. Standardmäßig ist `::before` ein Inline-Element. Dies kann anhand der Eigenschaft `display` verändert werden.

Beispiele

```
a[href]:before {content: "[LINK] ";}
p:before {content: attr(class);}
a[rel|;="met"]:after {content: " *";}
```

::first-letter

Erzeugt

Ein Pseudoelement, das den ersten Buchstaben eines Elements enthält.

Beschreibung

Versieht den ersten Buchstaben eines Elements mit Stildefinitionen. Vorangestellte Interpunktionszeichen sollten als Teil des ersten Buchstabens betrachtet werden. In manchen Sprachen gibt es Buchstabenkombinationen, die als ein Zeichen behandelt werden sollten. In diesem Fall kann das Benutzerprogramm die Stile auf beide Buchstaben anwenden. Vor CSS 2.1 konnte `::first-letter` nur auf Blockelemente angewandt werden. CSS 2.1 hat den Geltungsbereich erweitert, sodass auch Elemente mit dem `display`-Wert `block`, `list-item`, `table-cell`, `table-caption` oder `inline-block` angesprochen werden können. Auf den ersten Buchstaben können nicht alle Eigenschaften angewandt werden.

Beispiele

```
h1:first-letter {font-size: 166%;}
p:first-letter {text-decoration: underline;}
```

::first-line

Erzeugt

Ein Pseudoelement, das die erste formatierte Zeile eines Elements enthält.

Beschreibung

Versieht die erste Textzeile eines Elements mit Stilen, unabhängig davon, wie viele Wörter auf der Zeile erscheinen. `::first-line` kann nur auf Blockelemente angewandt werden. Auf die erste Zeile können nicht alle Eigenschaften angewandt werden.

Beispiel

```
p.lead:first-line {font-weight: bold;}
```

Media-Queries (Medienabfragen)

Mit Media-Queries können Sie die Verwendung von Stylesheets oder Teilen davon auf eine oder mehrere bestimmte Medienumgebungen festlegen. In der Vergangenheit wurden die Medientypen anhand des `media`-Attributs an `link`-Elementen oder durch den Mediendeskriptor für `@import`-Deklarationen festgelegt. Media-Queries erweitern dieses Konzept, indem Stylesheets basierend auf den Merkmalen eines bestimmten Medientyps ausgewählt werden können.

Grundkonzepte

Sofern Sie schon einmal einen Medientyp festgelegt haben, wird Ihnen die Platzierung von Media-Queries bekannt vorkommen. Hier sehen Sie zwei Möglichkeiten, ein externes Stylesheet (*print-color.css*) einzubinden, falls das Dokument auf einem Farbdrucker ausgegeben wird:

```
<link href="print-color.css" type="text/css"
 media="print and (color)" rel="stylesheet">

@import url(print-color.css) print and (color);
```

Überall dort, wo ein Medientyp benutzt werden kann, kann auch eine Media-Query verwendet werden. Daher ist es möglich, mehr als eine Media-Query in einer durch Kommata getrennten Liste anzugeben:

```
<link href="print-color.css" type="text/css"
 media="print and (color), projection and (color)"
 rel="stylesheet">

@import url(print-color.css) print and (color),
 projection and (color);
```

Sobald eine der Media-Queries zu »wahr« ausgewertet wird, kommt das verknüpfte Stylesheet zum Einsatz. Bei Verwendung der vorherigen `@import`-Anweisung wird *print-color.css* verwendet, sofern das Dokument auf einem Farbdrucker oder einem Projektor ausgegeben wird, der Farben darstellen kann. Wird das Dokument dagegen auf einem Schwarz-Weiß-Drucker ausgegeben, werden beide Queries zu »falsch« ausgewertet, und *print-color.css* wird nicht

auf das Dokument angewendet. Das Gleiche gilt für bestimmte Bildschirmumgebungen, Projektoren, die nur Graustufen darstellen können, akustische (aurale) Medienumgebungen und so weiter.

Jede Query besteht aus einem Medientyp sowie einem oder mehreren Medienfeatures. Jedes Medienfeature wird von runden Klammern umgeben. Mehrere Features können mit dem Schlüsselwort `and` verkettet werden. Für Media-Queries gibt es zwei logische Schlüsselwörter:

`and`
: Verbindet zwei oder mehr Medienfeatures. Damit die Query »wahr« ist, müssen alle Teilbedingungen wahr sein. Bei `(color) and (orientation: landscape) and (min-device-width: 800px)` müssen zum Beispiel alle Teile zur »wahr« ausgewertet werden: Sofern die Medienumgebung Farbe darstellen kann, sich im Querformat (Landscape) befindet und der Anzeigebereich mindestens 800 Pixel breit ist, wird das verknüpfte Stylesheet verwendet.

`not`
: Negiert die gesamte Query. Sind alle Teilbedingungen wahr, wird das Stylesheet *nicht* verwendet. Die Abfrage `not (color) and (orientation: landscape) and (min-device-width: 800px)` bedeutet: Sofern die Medienumgebung Farbe darstellen kann, sich im Querformat befindet und der Anzeigebereich mindestens 800 Pixel breit ist, wird das Stylesheet *nicht* verwendet. In allen anderen Fällen wird es verwendet. Das Schlüsselwort `not` kann ausschließlich am Anfang einer Media-Query verwendet werden. Es ist nicht erlaubt, etwas wie `(color) and not (min-device-width: 800px)` zu schreiben. Machen Sie das dennoch, wird die Query ignoriert. Browser, die zu alt sind, um Media-Queries zu verstehen, werden ein Stylesheet ignorieren, dessen Mediendeskriptor mit `not` beginnt.

Zwar gibt es kein `or`-Schlüsselwort, das innerhalb einer Query verwendet werden könnte, allerdings haben die Kommata, mit denen die Liste der Queries getrennt werden, den gleichen Effekt. So bedeutet die Formulierung `screen, print`: »Verwende das Stylesheet, wenn es sich um ein Bildschirmmedium (*screen*) oder einen

Drucker (*print*) handelt.« Anstelle der ungültigen Query `screen and (max-color: 2) or (monochrome)` sollten Sie daher schreiben: `screen and (max-color: 2), screen and (monochrome)`.

Das Schlüsselwort `only` kann verwendet werden, um absichtlich eine Abwärts-Inkompatibilität herzustellen.

`only`
: Wird verwendet, um ein Stylesheet vor Browsern zu verstecken, die aufgrund ihres Alters keine Media-Queries verstehen. Soll ein Stylesheet ausschließlich für Medien verwendet werden, wenn die Browser Media-Queries verstehen, könnten Sie beispielsweise `@import url(new.css) only all` schreiben. Browser, die keine Media-Queries verstehen, ignorieren das Schlüsselwort `only`. Beachten Sie, dass `only` ausschließlich am Anfang einer Media-Query verwendet werden kann.

Werte für Media-Queries

Media-Queries verwenden zwei neue Wertetypen, die (Anfang 2018) nur in diesem Kontext verwendet werden:

<Quotient> (Seitenverhältnis)
: Ein Quotientenwert (Verhältnis) besteht aus zwei positiven *<Integerwert>*en, die durch einen Schrägstrich (`/`) und optionale Leerzeichen voneinander getrennt werden. Der erste Wert bezieht sich auf die Breite und der zweite auf die Höhe. Um also ein Seitenverhältnis von 16:9 auszudrücken, können Sie `16/9` oder `16 / 9` schreiben.

<Auflösung>
: Eine Auflösung wird als positiver *<Integerwert>* gefolgt von einer der Einheitenangaben `dpi` oder `dpcm` formuliert. Wie üblich ist zwischen dem *<Integerwert>* und der Einheit kein Leerzeichen erlaubt.

Medienfeatures

Anfang 2018 standen folgende Medienfeatures zur Verfügung. Beachten Sie, dass die Werte nicht negativ sein dürfen:

`width`, `min-width`, `max-width`

Werte: <Länge>

Bezieht sich auf die Breite des Anzeigebereichs des Benutzerprogramms. In einem bildschirmbasierten Webbrowser ist dies die Breite des Ansichtsbereichs (Viewport) zuzüglich möglicher Scrollbalken. In seitenbasierten Medien ist dies die Breite der Seitenbox. Die Query (`min-width: 850px`) greift also, wenn der Anzeigebereich mehr als 850 Pixel breit ist.

`device-width`, `min-device-width`, `max-device-width`

Werte: <Länge>

Bezieht sich auf die Breite des vollständigen Rendering-Bereichs des Ausgabegeräts. In bildschirmbasierten Medien ist dies die Breite des Bildschirms. In seitenbasierten Medien ist dies die Breite der Seite. Die Query (`max-device-width: 1200px`) greift also, wenn der Ausgabebereich des Geräts höchstens 1.200 Pixel breit ist.

`height`, `min-height`, `max-height`

Werte: <Länge>

Bezieht sich auf die Höhe des Anzeigebereichs des Benutzerprogramms. In einem bildschirmbasierten Webbrowser ist dies die Höhe des Ansichtsbereichs (Viewport) zuzüglich möglicher Scrollbalken. In seitenbasierten Medien ist dies die Höhe der Seitenbox. Die Query (`height: 567px`) greift also, wenn der Ansichtsbereich genau 567 Pixel hoch ist.

`device-height`, `min-device-height`, `max-device-height`

Werte: <Länge>

Bezieht sich auf die Höhe des vollständigen Rendering-Bereichs des Ausgabegeräts. In bildschirmbasierten Medien ist dies die Höhe des Bildschirms. In seitenbasierten Medien ist dies die Höhe der Seite. Die Query (`max-device-height: 400px`) greift also, wenn die Höhe des Ausgabebereichs weniger als 400 Pixel beträgt.

`aspect-ratio`, `min-aspect-ratio`, `max-aspect-ratio`

Werte: <Quotient>

Bezieht sich auf den Quotienten aus den Medienfeatures `width` und `height` (siehe die Definition von *<Quotient>*).

Die Query (`min-aspect-ratio: 2/1`) greift also für Anzeigebereiche, deren Seitenverhältnis (Breite zu Höhe) geringer ist als 2:1.

`device-aspect-ratio`, `min-device-aspect-ratio`, `max-device-aspect-ratio`

Werte: <Quotient>

Bezieht sich auf das Verhältnis zwischen den Medienfeatures `device-width` und `device-height` (siehe die Definition von *<Quotient>*). Die Query (`device-aspect-ratio: 16/9`) greift also, wenn das Seitenverhältnis zwischen Breite und Höhe des Anzeigebereichs genau 16:9 ist.

`color`, `min-color`, `max-color`

Werte: <Integerwert>

Bezieht sich auf die Fähigkeit des Ausgabegeräts, Farben darzustellen. Eine optionale Zahl kann verwendet werden, um die Anzahl der Bits in jedem Farbanteil anzugeben. Die Query (`color`) passt also auf alle Geräte mit einer beliebigen Farbtiefe. Die Query (`min-color: 4`) bedeutet, dass die Farbtiefe pro Farbanteil mindestens 4 Bit betragen muss. Geräte ohne Farbunterstützung geben `0` zurück.

`color-index`, `min-color-index`, `max-color-index`

Werte: <Integerwert>

Bezieht sich auf die Gesamtzahl der Farben in der Lookup-Tabelle des Ausgabegeräts. Die Query (`min-color-index: 256`) passt also auf alle Geräte mit mindestens 256 verfügbaren Farben. Geräte, die keine Lookup-Tabelle für Farben verwenden, geben `0` zurück.

`monochrome`, `min-monochrome`, `max-monochrome`

Werte: <Integerwert>

Bezieht sich auf das Vorhandensein eines monochromen (einfarbigen) Displays. Eine optionale Zahl kann verwendet werden, um die Anzahl der Bits pro Pixel im Framebuffer des Geräts anzugeben. Die Query (`monochrome`) passt also auf alle monochromen Ausgabegeräte, während (`min-monochrome: 2`) nur auf monochrome Ausgabegeräte passt, deren Framebuffer mindestens 2 Bit pro Pixel verwendet. Nicht monochrome Ausgabegeräte geben `0` zurück.

`resolution`, `min-resolution`, `max-resolution`

Werte: *<Auflösung>*

Bezieht sich auf die visuelle Auflösung des Ausgabegeräts, das heißt die Pixeldichte, entweder gemessen in Punkten pro Zoll (dots per inch, `dpi`) oder Punkten pro Zentimeter (`dpcm`). Verwendet das Ausgabegerät nicht quadratische Pixel, wird die Achse mit der geringsten Dichte verwendet. Hat ein Gerät entlang einer Achse beispielsweise eine Pixeldichte von `100dpcm` und entlang einer anderen Achse eine Dichte von `120dpcm`, ist der Rückgabewert `100dpcm`. Außerdem kann eine reine `resolution`-Feature-Query niemals einen Treffer erzielen (`min-resolution` und `max-resolution` dagegen schon).

`orientation`

Werte: `portrait` | `landscape`

Bezieht sich auf den gesamten Ausgabebereich des Ausgabegeräts. Hierbei wird `portrait` (Hochformat) zurückgegeben, wenn das Medienfeature `height` größer ist als das Medienfeature `width`. Ansonsten ist das Ergebnis `landscape` (Querformat).

`scan`

Werte: `progressive` | `interlace`

Bezieht sich auf das verwendete Zeilenabtastverfahren (Scanning) eines Ausgabegeräts vom Typ `tv`.

`grid`

Werte: `0` | `1`

Bezieht sich auf das Vorhandensein (oder Fehlen) eines rasterbasierten Ausgabegeräts, wie z.B. ein `tty`-Terminal. Ein rasterbasiertes Gerät gibt `1` zurück; ansonsten lautet der Rückgabewert `0`..

Feature-Queries

Eine *Feature-Query* ist ein `@`-Regelblock, ähnlich einer Media-Query. Der Unterschied zu Media-Queries besteht darin, dass Feature-Queries überprüfen, ob das Benutzerprogramm eine bestimmte

Kombination aus Eigenschaft und Wert unterstützt. Gibt das Benutzerprogramm an, dass es die Query unterstützt, werden die Regeln innerhalb des `@`-Blocks angewendet. Ansonsten werden sie ignoriert.

Ein einfaches Beispiel wäre, den Browser zu fragen, ob die Regel `background-color red` unterstützt:

```
@supports (background-color: red) {
     html {background-color: yellow;}
     body {background-color: white;}
}
```

Die Eigenschaft-Wert-Kombination aus der Feature-Query muss nicht zwingend in folgenden Regeln verwendet werden. Tatsächlich muss nicht einmal die Eigenschaft benutzt werden, die Teil der Feature-Query war. So können Sie den Browser fragen, ob er `color: #FFF` unterstützt, und dann Regeln schreiben, die `color` überhaupt nicht verwenden. (Allerdings sollten Sie das nicht unbedingt tun, nur weil Sie es können.)

Feature-Queries können nützlich sein, wenn fortgeschrittene CSS-Merkmale verwendet werden sollen. Die Umwandlung eines Float-basierten Layouts in ein CSS-Grid könnte beispielsweise so aussehen:

```
[...Regeln für das Float-Layout hier...]

@supports (display: grid) {
     [...Regeln für das Grid-Layout...]
     [...Regeln, die Außenabstände und clear-Anweisungen
      deaktivieren, sowie weitere Regeln, die für
      ein Float-basiertes Layout gebraucht werden,
      aber nicht für ein Grid-Layout...]
}
```

Anhand des Schlüsselworts `not` können außerdem negierte Feature-Queries formuliert werden:

```
@supports not (shape-outside: circle()) {
     [...Regeln für Browser, die keine
      kreisförmigen Floats verstehen...]
}
```