1. Grundlagen

CSS-Anweisungen (Cascading Style Sheets / auch Formatdefinitionen genannt) dienen der optischen Gestaltung eines WEB-Dokuments. Dabei wird dem Prinzip der strikten Trennung von Struktur (HTML) und Präsentation (CSS) Rechnung getragen. Wesentliche Vorteile liegen in der Möglichkeit der Kaskadierung (Überlagerung) von Anweisungen sowie einer zentralen Layoutgestaltung. Der Aufbau einer Formatdefinition hat immer die Form:

```
① [Selektor] { Eigenschaft:Wert;[ Eigenschaft:Wert;...] }
```

Syntax 1

Die Namen möglicher CSS-Eigenschaften sowie das Format oder Schema der erlaubten Werte sind in der CSS-Spezifikation des W3-Konsortiums festgelegt.

2. Einbindung von Stylesheets in HTML

Es gibt grundsätzlich drei verschiedene Möglichkeiten der Einbindung von Formatdefinitionen (die sich in Ihrer Auswirkung auch unterscheiden)

2.1. Inline-Formatierung

Dabei werden die Formatdefinitionen mit dem Style-Attribut im Start-Tag des HTML-Elements notiert.

```
① <h1 style="background-color:blue">Hauptteil</h1>
```

Syntax 1

2.2. Stylesheet-Anweisung zentral im Dokument

Zentrale Formatvorlagen werden, durch das Style-Tag geschachtelt, im Head-Bereich des Dokuments definiert. Hierbei ist die MIME-Type Angabe „text/css“ notwendig. Da nun eine Zuordnung zu den zu formatierenden fehlt, ist die Angabe eines Selektors (hier z.B.: h1 und h2) nötig.

```
① <style type="text/css">
② h1 { background-color:blue; }
③ h2 { background-color:red; }
④ </style>
```

Syntax 2

2.3. Separates Stylesheet-Dokument

Die größte Flexibilität bieten externe Formatvorlagen, die über zwei Arten eingebunden werden können.

2.3.1. Separates Stylesheet über Link

Eine separate Vorlage-Datei wird durch das Link-Tag eingebunden. Das href-Attribut verweist auf die CSS-Datei mit den Formatdefinitionen. Die Angabe erfolgt im Head-Bereich des Web-Dokuments.

```
① <link rel="stylesheet" type="text/css" href="main.css" />
```

Syntax 3

2.3.2. Separates Stylesheet-über @import

Eine weitere Möglichkeit ist die Einbindung durch eine Import-Anweisung. Diese muss als erste Anweisung innerhalb des Style-Tags erfolgen. Dabei ist es möglich, für verschiedene Ausgabemedien (z.B.: screen, print ...) unterschiedliche Vorlagen zu definieren. Zur Abwärtskompatibilität können auch beide Varianten kombiniert werden.

```
① <style type="text/css">
② <!--
③ @import url(main.css);
④ -->
⑤ </style>
```

Syntax 4

3. Überlagerung (Kaskadierung) von Formatanweisungen

Grundsätzlich können die verschiedenen Arten der Einbindung auch kombiniert werden, wodurch sich Mehrfachformatierungen für Elemente ergeben können.

- Bei verschiedenen (sich nicht ausschließenden) Formatanweisungen werden alle wirksam.
- Bei konkurrierenden (meist gleichen) Anweisungen wird die Spezifizität berechnet, die dann über die Formatierung entscheidet (Faustregel: die dem Element am nächste stehende Anweisung zieht).

3.1. Mehrfachzuweisung

Durch die Zuordnung von id-Attributen bzw. class-Attributen zu HTML-Tags besteht die Möglichkeit, innerhalb gleicher HTML-Tags unterschiedliche Formatierungen (id) oder für verschiedene HTML-Tags gleiche Formatierungen (class) zu erzielen. Ein Element kann auch mehreren Klassen zugeordnet werden.

3.2. Selektorregeln

Bei zentralen Vorlagen werden durch Angabe von Selektoren (vor den eigentliche Formatierungsanweisungen) die Zuordnungen zu den Tags festgelegt. Nachfolgend werden wesentliche Selektionsoperatoren beschrieben.

Zeichen	Bedeutung	Beispiel
,	Aufzählung	`h1, h2 ,h3 {color:red;}`

.	class-Wert	`h1.rot {color:red;}`
		formatiert nur h1-Tags dieser Klasse: <h1 class="rot">
		`.rot {color:red;}`
		formatiert alle Tags der Klasse: <h1 class="rot"> /

#	id-Wert	`h1#rot {color:red;}`
		formatiert nur das h1-Tag mit dieser id: <h1 id="rot">
		`#rot {color:red;}`
		formatiert alle Tags mit dieser id: <h1 id="rot"> /

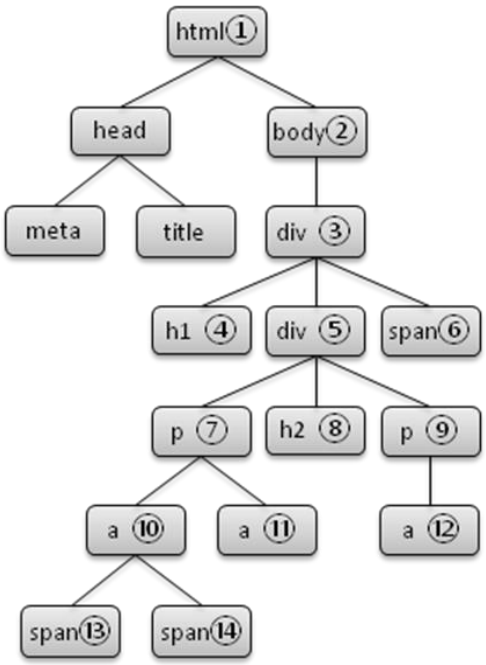
(Leer)	Verschachtelung (innerhalb von ...)	`h1 a {color:red;}` formatiert alle a-Tags innerhalb von h1, also: <a>Beispiel1 oder <a>Beispiel1
>	Verschachtelung (direkt aufeinander folgend von ...)	`h1 > a {color:red;}` formatiert nur a-Tags direkt nach h1, also: <a>Beispiel1 aber nicht : <a>Beispiel1 oder nicht : Text <a>Beispiel1
*	Verschachtelung (nur nicht direkt folgend von ...)	`h1 * a {color:red;}` formatiert nur a-Tags indirekt nach h1, also: <a>Beispiel1 aber nicht Text <a>Beispiel1
+	Reihenfolge (direkt nach ...)	`h1 + a {color:red;}` formatiert nur a-Tags direkt nach h1, also: Titel <a>Beispiel1 aber nicht : Titel Inhalt <a>Beispiel1
~	Reihenfolge (indirekt nach ...)	`h1 ~ a {color:red;}` formatiert a-Tags nach h1, also: Titel Inhalt <a>Beispiel1 aber nicht Titel <a>Beispiel1
[]	Attributsbezug	`a[lang="de"] {color:red;}` formatiert a-Tags mit dem lang="de" , also: Beispiel1 aber nicht Beispiel2
:	Pseudoklassen	`a:hover {color:red;}` formatiert a-Tags über denen sich gerade der Cursor befindet. Weitere Pseudoklassen :link / : visited / :first-letter

4. Vererbung

Eine wesentliche Eigenschaft von CSS-Formatanweisungen ist die Möglichkeit der Vererbung. Dazu wird das DOM-Model einer WEB-Sites betrachtet (siehe (X)HTML5 Dokumentation).

CSS-Eigenschaften werden in vererbbare und nichtvererbbare Eigenschaften unterteilt (was jeder Referenz entnommen werden kann).

Exemplarisch werden hier die Eigenschaft **`color`** (als vererbbare) und **`border`** (als nicht vererbbar) betrachtet.

Beispiel:	**DOM Modell**
Eine Zuweisung der Form: **`p#⑦ {color:red;}`** würde allen Kinder- (und Kindes-Kinder) Elemente von p#⑦ eine rote Schriftfarbe zuweisen (Hier also: a⑩ , a⑪ , span⑬ und span⑭).	
Die Überlagerungen (und damit verbundenen Regel) behalten weiterhin ihre Gültigkeit. So würde ein Zusätzliche Formatiereung der Form : **`a#⑩ {color:blue;}`**, die Vererbung ab diesem Tag unterbrechen, so dass span⑬ und span⑭ von a#⑩ die Farbe Rot erben würden.	
Eine Zuweisung (einer nichtvererbbaren Eigenschaft) **`div#⑤ {border:1pt; solid green;}`** würde nur diesen div#⑤-Block grün umrahmen. Allerdings kann durch die Angabe: **`p#⑨ {border:inherit;}`** im Kind-Element von p#⑨ explizit festgelegt werden, dass auch nichtvererb-bare Eigenschaften vom Eltern-Element übernommen werden.	

5. Layoutgestaltung mit CSS

In früheren Versionen wurde die Layout-Gestaltung (also die räumliche Position von Elementen) noch mit (unsichtbaren) Tabellen oder mit Frames realisiert. Dies ist heute nicht mehr „state oft the art“ und wird auch mittels CSS realisiert. Der Vorteil liegt in der vielfältigeren Gestaltungsmöglichkeiten. Basis ist das Boxenmodell.

5.1. Das Boxen-Modell

Jedes sichtbare Element auf einer WEB-Seite kann als rechteckiger Box betrachtet werden, die das Anordnungsverhalten innerhalb dieser Seite bestimmt. Dabei wird unterschieden zwischen:

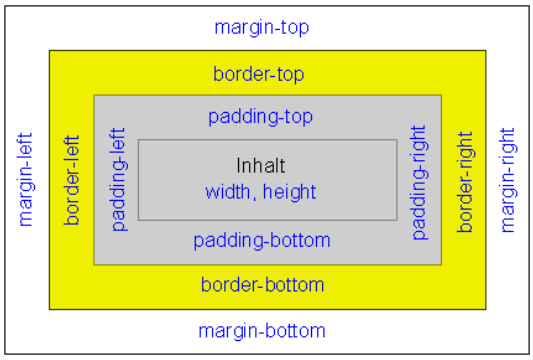
Block-Elemente . . .	**Inline-Elemente . . .**
... werden so breit wie **möglich** (maximale Breite) und so hoch wie **nötig** dargestellt.	... werden so breit wie **nötig** (minimale Breite) und so hoch wie **nötig** dargestellt.
... dürfen (ausgewählte) Block Elementen und/oder weiter Inline-Elemente enthalten.	... dürfen nur (geschachtelt) Inline-Elemente enthalten.
... sind zum Beispiel: `address, blockquote, del, div, dl, fieldset, form, h1-6, hr, ins , menu, noscript, ol, p, pre, table, ul`	... sind zum Beispiel: `a, b, button, code, del, dfn, em, i, img, ins, input, label, map, q, samp, select, small, span, strong, sub, sup`

Diese vordefinierte Eigenschaft (Block oder Inline) kann mittels der Style-Anweisung auch geändert werden:

①	`{ display : block | inline | none }`
	Syntax 5

5.2. Abstände und Abmessungen im Boxenmodell

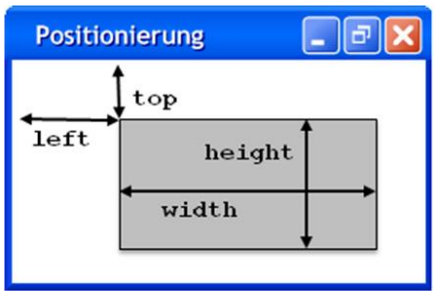
Jedes Element belegt eine definierte Größe im Browserfenster. Für die Festlegung der Größe und der Abstände im Boxenmodell sind die Stylesheet-Eigenschaften **margin** (Außenabstand) und **padding** (Innenabstand) vorgesehen:




```
① { margin :  top, right, bottom, left }
② { padding : top, right, bottom, left }
```
Syntax 6

Beim Aufbau (Rendering) einer WEB-Site verarbeitet die Browser-Engine sequenziell die Anweisungen zur Darstellung der Elemente (entsprechend der Reihenfolge im DOM). Daraus resultiert das Seitenlayout. Mit der Eigenschaft **position** lässt sich für jedes Element ein abweichendes Verhalten dieser Vorschrift definieren.

```
① { position : absolute | relative | fixed | sticky }
```
Syntax 7



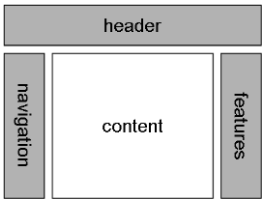
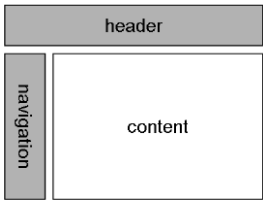
So kann beispielsweise bei einer absoluten Positionierung mit den Eigenschaften **width** und **height** die Größe der Elemente festgelegt werden. Mit den Eigenschaften **top** und **left** können Diese dann auf der Seite positioniert werden.

Nachteil : bei dieser Gestaltungs-Variante ist eine automatische Anpassung an die Browserfenstergröße nicht möglich.

5.3. Layoutgestaltung

Bei fast allen Arten von WEB-Auftritten (deren Aufgabe nicht darin besteht durch Originalität zu bestechen) haben sich zwei Standard-Layouts etabliert.

Die Hauptelemente werden über Container (div-Elemente) im HTML Dokument deklariert. Position und Verhalten werden über Formatvorlagen festgelegt und eingebunden. Mögliche Beispiele (der angegebenen Quelle entnommen) sind exemplarisch dargestellt.

Portallayout	**Winkellayout**
 „Das Portallayout signalisiert einerseits überlegene Ruhe und Ausgewogenheit, andererseits aber auch eine Fülle von Informationen und unterschiedlichen Angeboten“ *	 „Das Winkellayout signalisiert mehr Dynamik als das Portallayout. Die Navigation ist dominanter, es wird mehr Hypertextfeeling vermittelt.“ *

`<body>`
`<div id="header">Titel </div>`
`<div id="navi"> Menue </div>`
`<div id="content"> Inhalt </div>`
`<div id="features"> Hilfe </div>`
`</body>`
	`<body>`
	`<div id="top"> Titel </div>`
	`<div id="main">`
	`<div id="navi"> Menue </div>`
	`<div id="content"> Inhalt </div>`
	`</div>`
	`</body>`

*Quelle : „Profesionelle Websites“ / Stefan Münz Seite 229 /Addison Wesley

CSS-Grid bietet die Möglichkeit ein Gestaltungsraster zu schaffen und die Positionierung der einzelnen Elemente vollständig durch CSS zu steuern. Damit wird eine klare Trennung des Gestaltungsrasters von der HTML-Textstruktur erreicht. Zusammen mit Media Queries erlaubt es den Aufbau flexibler Layouts.
Die Layout Gestaltung erfolgt in zwei Schritten:

1. Im ersten Schritt wird festgelegt, für welchen Bereich eines HTML-Dokumentes (container) CSS-Grids angewendet werden sollen und wie das Element in Rasterlinien eingeteilt wird.
2. Im zweiten Schritt werden die Elemente (items) innerhalb des Rasters positioniert (explizit für jedes Element).

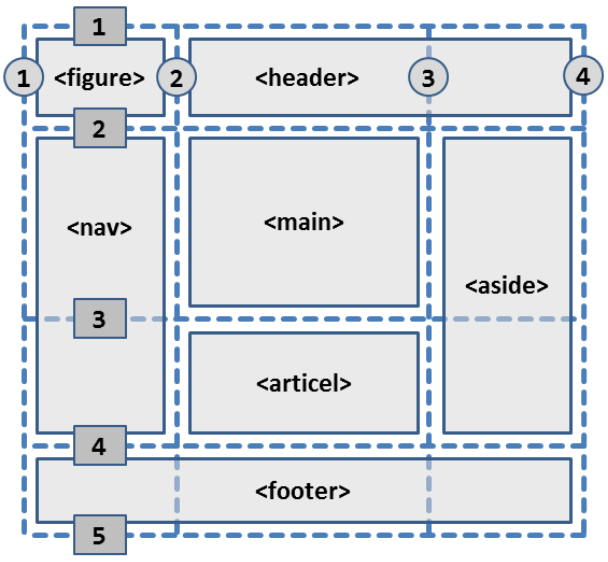
```
① container {
② display: grid ;
③ grid-template-columns: <track-size> ... ;
④ grid-template-rows: <track-size> ... ;
⑤ [ grid-gap: <line-size>; ]
⑥ [ grid-column-gap: <line-size>; ]
⑦ [ grid-row-gap: <line-size>; ]
⑨ [ justify-items: start | end | center | stretch; ]
⑩ [ align-items: start | end | center | stretch; ]
⑪ }
```

Eigenschaften des GRID-Containers

```
① item {
② grid-column: <start-line> / <end-line> | <start-line> / span <value>;
③ grid-row: <start-line> / <end-line> | <start-line> / span <value>;
④ [ justify-self: start | end | center | stretch; ]
⑤ [ align-self: start | end | center | stretch; ]
⑥ }
```

Eigenschaften der GRID-Elemente

Im abgebildeten Beispiel wird ein Grid mit 3 Spalten und 4 Zeilen im Container <body> definiert.




```
body { display: grid;
       grid-template-columns: 100px 1fr 20%;
       grid-template-rows:
       80px calc(100% - 310px) 150px 80px ;
       grid-gap:10px; }
figure { grid-column: 1;   grid-row: 1;  }
header { grid-column: 2/4; grid-row: 1;  }
nav    { grid-column: 1;   grid-row: 2/4;}
main   { grid-column: 2;   grid-row: 2;  }
aside  { grid-column: 3;   grid-row: 2/4;}
article{ grid-column: 2;   grid-row: 3;  }
footer { grid-column: 1/4; grid-row: 4;  }
```

CSS-Formatierung

HTML-Container

```
<body> <figure> figure </figure>
       <header> header </header>
       <nav>    nav    </nav>
       <main>   main   </main>
       <aside>  aside  </aside>
       <article>article</article>
       <footer> footer </footer>    </body>
```

Die Spezifikation wurde mehrfach geändert. Im neuen Working-Draft definiert man damit nicht eigentlich Spalten, sondern vertikale **Rasterlinien**. Die **aktuelle Version** (Stand 3/2017) funktioniert zum Beispiel in Chrome ab 57, und im Firefox ab Version 52.