



CODERS.**BAY**

SASS

Sass ist ein CSS- Präprozessor. Präprozessoren erleichtern das Schreiben von Code, indem sie eine vereinfachte oder bessere Syntax bereitstellen als die Programmiersprache selbst, lästige Aufgaben automatisieren und neue Funktionen bieten. Neben CSS-Präprozessoren wie Sass, existieren auch Präprozessoren für HTML (z.B. HAML) oder für JavaScript (z.B. CoffeeScript)

Vorteile von CSS-Präprozessoren

- Sie vereinfachen die Syntax, damit ihr Stylesheets schneller schreiben könnt
- Sie helfen euch dabei ein Projekt zu organisieren und in verständliche Sinnabschnitte zu unterteilen
- Sie erlauben Variablen, übernehmen die Berechnung von Layout-Abmessungen, generierten Farbsets uvwm.
- Sie reduzieren die Anzahl von HTTP-Requests, da Stylesheets kombiniert werden können
- Sie automatisieren lästige Arbeitsabläufe wie die Kompression von Dateien oder Stylesheets

Vorteile von Sass gegenüber anderen CSS-Präprozessoren wie LESS, Stylus, Myth etc.:

- Sass ist GPL-kompatibel und unter MIT lizenziert
- Sass ist abwärtskompatibel mit allen CSS-Versionen - zumindest mit der SCSS-Syntax
- Sass unterstützt komplexere Logiken wie **if-else**-Bedingungen, **while**, **for** oder **each**-Schleifen
- Sass ist am populärsten und hat die größte Community

ENTSTEHUNG

Sass wurde 2006 von Hampton Catlin ins Leben gerufen. Das Projekt wurde von vielen Unterstützern weiterentwickelt, wobei die Core-Entwickler Nathan Weizembau (bis Version 2.0) und Christopher Eppstein (ab Version 2.2) eine besondere Rolle einnehmen. Christopher Eppstein ist u. a. auch Entwickler des Compass-Frameworks, das im Umfeld von Sass eine große Bedeutung hat

In der modernen Programmierung versucht man Änderungen immer nur an einer Stelle vornehmen zu müssen. Diese Vorgehensweise entspricht sowohl dem Templating in HTML, als auch der objektorientierten Programmierung in Programmiersprachen wie PHP. Auch Content Management Systeme sind häufig nach dem sog. **DRY-Prinzip (Don't repeat yourself)** aufgebaut und arbeiten mit Modulen.

CSS hingegen zwingt uns dazu unzählige Wiederholungen im Code einzubauen. Wenn die Hausfarbe unseres Unternehmens Blau ist und wir möchten die Überschriften, die Links und die Buttons unserer Website in diesem Blau gestalten, dann bleibt uns kaum etwas anderes übrig als allen Elementen den gleichen

Farbcode zuzuweisen. Wenn sich die Farbe ändert, müssen wir alle Code-Passagen anpassen. Diese Arbeitsweise ist nicht effektiv und führt leicht zu Fehlern.

Ziel ist es, Wiederholungen im CSS Code weitestgehend zu vermeiden, das Stylesheet also **DRYer** zu machen.

Ein weiterer Nachteil von CSS ist, dass die Sprache nicht für so komplexe Layouts entwickelt wurde, wie wir sie heute umsetzen. Das merken wir nicht nur an den unzähligen Wiederholungen, auch fehlende Layout-Modelle, diverse Vendor-Präfixe etc. sind ein klares Indiz.



CODERS.BAY

WAS IST SASS

WAS IST SASS BZW. SCSS

Sass steht für "Syntactically Awesome Stylesheets" und ist ein CSS-Präprozessor. Ein CSS-Präprozessor ist eine weitere Ebene zwischen dem Stylsheet, das geschrieben wird und dem Stylesheet, das letztendlich dem Browser zur Interpretation vorgelegt wird.

Geschrieben wird heute meist ein ***.scss** (Sassy CSS)-Dokument. Dieses Stylesheet lässt sich sehr einfach verwalten und ermöglicht es uns bislang nicht verfügbare Funktionen zu nutzen. Mit Hilfe von Sass wird es in ein normales CSS-Dokument umgewandelt (kompiliert, engl. compiled). Dieses normale CSS-Dokument wird dann auf dem Server eingesetzt, da der Browser nur CSS interpretieren kann, nicht aber Sass oder andere Präprozessoren.

Die Website von Sass **beschreibt den Präprozessor** mit folgenden Sätzen

CSS with superpowers

Die **ausführliche und sachliche Version** lautet:

Sass is a meta-language on top of CSS that's used to describe the style of a document cleanly and structurally, with more power than a flat CSS allows. Sass both provides a simpler, more elegant syntax for CSS and implements various features that are useful for creating manageable stylesheets.

Sass kann als **Erweiterung von CSS3** verstanden werden

*Das bedeutet automatisch, dass jedes valide CSS3-Dokument auch ein valides *.scss-Dokument ist. Diese Tatsache erlaubt es euch, Sass nach und nach in euren Workflow zu integrieren, da CSS immer auch SCSS ist.*

Nehmt einfach ein bestehendes CSS-Dokument und ändert die Dateiendung von ***.css** auf ***.scss**. Ihr habt nun ein voll funktionsfähiges SCSS-Dokument erstellt.

DIE SASS-SYNTAX

Es existieren zwei Schreibweisen:
SASS und SCSS

Die neuere Schreibweise ist die SCSS-Syntax. Diese Syntax hat verschiedene Vorteile, u. a. die bereits erwähnte automatische Kompatibilität von CSS-Dokumenten und die damit verbundene unkomplizierte Einarbeitung in Sass bzw. die stückweise Umwandlung von CSS-Dokumenten in SCSS-Dokumente.

Die SASS-Syntax hingegen hat den Vorteil, dass sie noch stärker reduziert wird. Es wird u. a. auf die Semikola am

Ende einer Zeile und auf geschwungene Klammern verzichtet. Verschachtelungen werden mit Einrückungen erreicht, weshalb die Schreibweise auch als "Indented Syntax" bezeichnet wird. Diese Reduzierung hat aber den Nachteil, dass ein bestehendes CSS-Dokument nicht automatisch ein valides SASS-Dokument ist. Das generierte CSS-Dokument ist jedoch bei beiden Schreibweisen identisch.

Die nachfolgenden Beispiele von Variablen zeigen den Unterschied zwischen der beiden Schreibweisen.

```
// *.sass  
$pink: #ff007e;  
  
p  
    color: $pink
```

```
// *.scss  
$pink: #ff007e;  
  
p {  
    color: $pink;  
}
```

```
// generiertes CSS  
p { color: #ff007e; }
```

SASS INSTALLIEREN UND EINRICHTEN

Sass ist in der Programmiersprache Ruby geschrieben und übernimmt die Umwandlung (Kompilierung) des SCSS- bzw. SASS-Dokuments in ein normales CSS-Dokument. In diesem Kapitel lernt ihr wie Ruby und Sass installiert und eingerichtet werden. Dabei habt ihr zwei verschiedene Möglichkeiten:

- Installation über die Kommandozeile
- Installation mit Hilfe von Desktop-Apps

Die Installation über die Kommandozeile ist sehr unkompliziert und schnell erledigt. Es empfiehlt sich diesen Weg zumindest in der Theorie zu verstehen.

Eine Step-by-Step Anleitung gibt es hier:

<https://blog.kulturbanause.de/2014/06/sass-ueber-die-kommandozeile-installieren/>

KOMPILIERUNG (COMPILING)

Die Umwandlung von SCSS-Dateien in CSS-Dateien nennt man Kompilierung (engl. compiling). Nachdem Sass installiert wurde, muss zunächst festgelegt werden welche Dateien von Sass überwacht werden sollen. Wenn ausschließlich Änderungen am SCSS-Stylesheet vorgenommen werden, kompiliert Sass die Dateien im Hintergrund automatisch zu CSS

Wichtig: Sobald mit Sass gearbeitet wird, darf keine CSS-Datei mehr editiert werden. Änderungen am CSS-Dokument würden von einer Änderung am SCSS-Dokument und der daraufhin erfolgenden automatischen Kompilierung überschrieben

Ihr habt folgende Möglichkeiten die Kompilierung einzurichten:

- Ihr nutzt die Kommandozeile
- Ihr verwendet eine Desktop-App

Wie bereits erwähnt, existieren zahlreiche Desktop-Anwendungen um mit Sass zu arbeiten. Diese Programme übernehmen u. a. auch die Kompilierung, aktualisieren bei Änderungen am Stylesheet automatisch den Browser und verwalten mehrere Projekte parallel.

Hier eine Anleitung für die Umwandlung von SASS/SCSS-Dateien in CSS-Dateien:

<https://blog.kulturbanause.de/2014/06/einstieg-in-sass-scss-zu-css-kompilieren/>

OUTPUT-STYLE - NESTED

Es existieren verschiedene "output-Styles" für das kompilierte CSS-Dokument. Als Output-Style bezeichnet man die Struktur des generierten CSS-Dokuments.

Nested (Standard)

Der Output-Style "Nested" ist der Standardwert. Wenn für die Kompilierung kein abweichender Output-Style definiert wurde, wird dieser Stil verwendet. "Nested" bedeutet soviel wie verschachtelt. Die CSS-Selektoren werden eingerückt um die HTML-Struktur des Dokuments zu verdeutlichen. Der Code ist für Menschen sehr gut lesbar.

```
// Beispiel für den Output-Style "Nested":  
ul {  
    margin: .5em 0;  
    float: left;  
    width: 100%; }  
ul li {  
    float: left; }  
ul li a {  
    color: #333;  
    text-decoration: none;  
    display: block;  
    padding: .5em; }
```

OUTPUT-STYLE - EXPANDED

Der Output-Style "Expanded" wird mit folgendem Befehl in der Kommandozeile aktiviert. Alternativ kann der Output-Style auch über eine der Desktop-Apps eingestellt werden.

```
// Kommandozeile
>$ sass --watch --style expanded stylesheets/scss:stylesheets/css
```

```
// Beispiel für den Output-Style "Expanded":
ul {
  margin: .5em 0;
  float: left;
  width: 100%;
}
ul li {
  float: left;
}
ul li a {
  color: #333;
  text-decoration: none;
  display: block;
  padding: .5em;
}
```

OUTPUT-STYLE - COMPACT

Der Output-Style "Compact" wird mit folgendem Befehl in der Kommandozeile aktiviert.

```
// Kommandozeile
>$ sass --watch --style compact stylesheets/scss:stylesheets/css
```

Der kompakte Stil zeichnet sich dadurch aus, dass alle CSS Eigenschaften zu einem Selektor in einer Zeile nebeneinander geschrieben werden. Leerzeichen zwischen den Regeln entfallen. Auch dieser Stil wird von einigen Web Design-Kollegen per Hand geschrieben, da es ihnen so leichter fällt die verschiedenen Selektoren zu identifizieren.

```
// Beispiel für den Output-Style "compact":
ul { margin: .5em 0; float: left; width: 100%; }
ul li { float: left; }
ul li a { color: #333; text-decoration: none; display: block; padding: .5em; }
```

OUTPUT-STYLE - COMPRESSED

Der Output-Style "Compressed" wird mit folgendem Befehl in der Kommandozeile aktiviert.

```
// Kommandozeile  
$ sass --watch --style compressed stylesheets/scss:stylesheets/css
```

Der komprimierte Stil wird weder von Hand geschrieben, noch ist er für Menschen gut lesbar. Alle Leerzeichen und Zeilenumbrüche werden entfernt. Dieser Output-Style dient der Performance-Optimierung.

```
// Beispiel für den Output-Style "compact":  
ul{margin:.5em 0;float:left;width:100%;}ul li{float:left;}ul li a{color:#333;text-decoration:none;display:block;padding:.5em;}
```

MIT SASS ARBEITEN

Sass generiert nicht automatisch besseren CSS-Code. Wenn ihr CSS nicht ausreichend beherrscht, füllt Sass diese Wissenslücken nicht auf. Sass unterstützt euch dabei, sauberen Code schneller zu schreiben und einfacher zu verwalten.

In den verschiedenen Editoren wie Visual Studio Code gibt es für das Compiling und die Syntax Highlighten diverse Packages um das Arbeiten mit Sass zu erleichtern.

VERSCHACHTELTE SELEKTOREN (SELECTOR NESTING)

Innerhalb des SCSS-Dokuments kann mit Verschachtelungen gearbeitet werden, die die HTML-Struktur repräsentieren. Das erleichtert die Arbeit insofern, dass weniger Code geschrieben werden muss. Bis zu einer gewissen Verschachtelungstiefe wird auch die Lesbarkeit des Codes verbessert.

SCSS-Verschachtelung bildet nicht 1:1 die HTML-Verschachtelung ab. Achtet darauf verschachtelte Selektoren behutsam einzusetzen! Andernfalls treibt ihr die CSS Spezifität unnötig in die Höhe und erzeugt Performance-Probleme und komplexen Code. Tiefer als drei Ebenen solltet ihr nur im Ausnahmefall verschachteln.

```
// Beispiel für Selector Nesting
.site-header {
    background: blue;
    height: 200px;

    .logo {
        height: 80px;
        width: 300px;
    }
}
```

```
//Kompiliert zu:
.site-header {
    background: blue;
    height: 200px;
}
.site-header .logo {
    height: 80px;
    width: 300px;
}
```

NEGATIVBEISPIEL

```
// Kompiliert zu
.nav-main ul li {
    float: left;
    margin-right: 1em;
}
.nav-main ul li a {
    color: black;
    text-decoration: none;
}
```

VERSCHACHTELTE EIGENSCHAFTEN (PROPERTY NESTING)

In CSS existieren verschiedene Eigenschaften, die unter dem gleichen Nameskürzel zusammengefasst sind. Das trifft beispielsweise auf background-, text-, font-, margin-, padding-, usw. zu. Auch hier kann mit Verschachtelungen gearbeitet werden.

```
// Property Nesting
.site-header {
    height: 200px;
    margin: {
        top: 20px;
        bottom: 40px;
    }
    padding: 10px;
    background: {
        color: blue;
        image: url("background.png");
        repeat: no-repeat;
        position: top center;
    }
}
```

```
// Kompiliert zu
.site-header {
    height: 200px;
    margin-top: 20px;
    margin-bottom: 40px;
    padding: 10px;
    background-color: blue;
    background-image: url("background.png");
    background-repeat: no-repeat;
    background-position: top center;
}
```

ELTERN-SELEKTOREN REFERENZIEREN

Mit dem &-Zeichen können Eltern-Selektoren referenziert werden. Diese Eigenschaft wird in CSS3 häufig schmerhaft vermisst. Das folgenden Beispiel zeigt, wie mit Hilfe des sog. "Parent-Selectors" der Hover-Effekt für einen Link gestaltet werden kann

```
a {  
    font-size: 1.5em;  
    text-decoration: none;  
    color: red;  
  
    &:hover {  
        text-decoration: underline;  
        color: black;  
    }  
}
```

```
a {  
    font-size: 1.5em;  
    text-decoration: none;  
    color: red;  
}  
a:hover {  
    text-decoration: underline;  
    color: black;  
}
```

BEISPIEL

Das &-Zeichen muss nicht am Anfang des Selektors stehen.

```
header {  
    height: 200px;  
    background: {  
        image: url("header.png");  
        position: top center;  
        repeat: no-repeat;  
    }  
    #home & {  
        height: 400px;  
        background-image: url("header-home.png");  
    }  
}
```

```
header {  
    height: 200px;  
    background-image: url("header.png");  
    background-position: top center;  
    background-repeat: no-repeat;  
}  
#home header {  
    height: 400px;  
    background-image: url("header-home.png");  
}
```

VARIABLEN

Unser CSS-Code beinhaltet sehr viele Wiederholungen: Farben, Schriftarten, Abstände, Grafiken etc. werden mehrfach im CSS-Code definiert und müssen folglich auch an verschiedenen Stellen geändert werden. Eine der hilfreichsten Funktionen von Sass sind daher die Variablen.

Mit Variablen ist es möglich einen Wert zu speichern und anschließend an verschiedenen Stellen im Dokument einzusetzen.

Die Syntax von Variablen entspricht dem Aufbau einer CSS-Eigenschaft mit Wert. Eine Variable wird mit einem Dollarzeichen (\$) eingeleitet, es folgt der Name der Variable, der frei wählbar ist. Nach einem Doppelpunkt folgt dann der Wert der Variable. Die Zeile wird mit dem Semikolon abgeschlossen

```
$color-primary: red;  
$color-secondary: orange;
```

BEISPIEL

```
$color-primary: red;
$color-secondary: orange;
$color-dark: black;
$color-light: white;
$font: Gotham, "Helvetica Neue", Helvetica, Arial, sans-serif;
$padding-medium: .5em;

body {
  background: $color-light;
  color: $color-dark;
  font-family: $font;
}

header {
  background: $color-primary;
  color: $color-light;
  padding: $padding-medium;
}

nav {
  background: $color-secondary;
  color: $color-light;
  padding: $padding-medium;
}
```

```
body {
  background: white;
  color: black;
  font-family: Gotham, "Helvetica Neue", Helvetica, Arial, sans-serif;
}

header {
  background: orange;
  color: white;
  padding: .5em;
}

nav {
  background: orange;
  color: white;
  padding: .5em;
}
```

VARIABLEN SINNVOLL BEZEICHNEN

Variablen sollten (wie CSS-Klassen) entsprechend ihrer Funktionen und nicht entsprechend des Aussehens benannt werden. Eine Variable namens **\$color-company** ist beispielsweise sinnvoller **\$dark-red**. Würde die Firmenfarbe von Rot auf Grün geändert, würde die zweite Variable nämlich nicht logisch funktionieren. Die erste hingegen schon.

Darüber hinaus ist es sinnvoll, Variablennamen mit einheitlicher Namenskonvention zu benennen. Das folgende Beispiel zeigt Vor- und Nachteile verschiedener Bezeichnungen:

```
$blue;  
$dark-blue;  
$medium-blue;  
$darkest-blue;  
$light-blue;  
$lightest-blue;  
  
//bessere Bezeichnung  
$blue;  
$blue-dark;  
$blue-darkest;  
$blue-light;  
$blue-lightest;
```

Wenn eine Variable mehrfach definiert wird, kompiliert Sass immer die letzte Definition im Code.

Mit Hilfe der Angabe `!default` kann eine Variable als Standard definiert werden

`!default` kennzeichnet eine Variable, die als Standardwert definiert wurde. Dieser Standard wird aber nur dann verwendet, wenn dieser Variablen kein anderer Wert zugewiesen wurde. Dabei ist es egal, wo im Code das geschieht. Sinnvoll ist dieses Verhalten, wenn Variablen für mehrere Projekte verwendet werden sollen. Es wäre also möglich ein Dokument mit Standard-Variablen zu erstellen und für verschiedene Projekte zu verwenden. Wenn ein Wert geändert werden soll, definiert ihr die Variable einfach neu und überschreibt somit den Standard. Besonders interessant ist dieses Verhalten im Zusammenhang mit Mixins.

```
$header-height: 300px !default;
```

Farben manipulieren

Eine Website basiert normalerweise auf einem durchdachten Farbschema und einigen Farbabstufungen innerhalb der jeweiligen Farbe. Sass bietet umfangreiche Möglichkeiten, Farben zu manipulieren. Wenn der Sass-Code entsprechend vorbereitet wurde, ist es sogar möglich ein Design mit nur einer Variable komplett umzufärben - inkl. Helligkeitsabstufungen, Komplementärfarben etc.

Mit der Sass-funktion `lighten` und `darken` ist es möglich Farben aufzuhellen oder abzudunkeln. In Kombination mit Variablen ist das ein sehr mächtiges und häufig eingesetztes Feature.

```
$color1: red;

div:nth-of-type(1) {
    background:$color1;
    border-top-color: lighten($color1, 30%);
    border-right-color: darken($color1, 15%);
    border-left-color: lighten($color1, 15%);
    border-bottom-color: darken($color1, 30%);
    border-width: 10px;
    border-style: solid;
}
```

```
//kompliert zu
div:nth-of-type(1) {
    background: red;
    border-top-color: #ff9999;
    border-right-color: #b30000;
    border-left-color: #ff4d4d;
    border-bottom-color: #660000;
    border-width: 10px;
    border-style: solid;
}
```

MÖGLICHKEITEN

Farbton verändern

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: adjust-hue($color, 80%);  
}  
div:nth-of-type(2) {  
    background: saturate($color, -80%);  
}
```

Es ist auch möglich die Sättigung einer Farbe zu verändern.

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: desaturate($color, 50%);  
}  
div:nth-of-type(2) {  
    background: saturate($color, 50%);  
}
```

Farbe invertieren

```
$color: steelblue;  
div:nth-of-type(1) {  
    border: 10px solid invert($color);  
}
```

Komplementärfarben erzeugen

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: complement($color);  
}
```

Farben in Graustufen umrechnen

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: grayscale($color);  
}
```

MÖGLICHKEITEN

Deckkraft und Farbmodus verändern

Es ist auch möglich die Deckkraft von Farben zu verändern - selbst wenn innerhalb der Variablen die Farbe in einem Farbmodus angegeben wurde, der keine Transparenzen zulässt (z. B. als Hexadezimalzahl). Das folgende Beispiel nutzt das RGBA-Farbmodell (Rot, Grün, Blau, Alpha-Transparenz) zum Bestimmung der Transparenz. Ein Wert von .6 entspricht 60%.

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: rgba($color, .6);  
}
```

Farben in Graustufen umrechnen

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: grayscale($color);  
}
```

Farbe invertieren

```
$color: steelblue;  
div:nth-of-type(1) {  
    border: 10px solid invert($color);  
}
```

Komplementärfarben erzeugen

```
$color: steelblue;  
div:nth-of-type(1) {  
    background: complement($color);  
}
```

Farben mischen

Ihr könnt auch zwei Farbwerte mischen

```
$color1: gray;  
$color2: aqua;  
div:nth-of-type(1) {  
    background: mix($color1, $color2);  
}  
div:nth-of-type(2) {  
    background: mix($color1, $color2, 75%);  
}
```

MIXINS

Mixins sind ein weiteres äußerst praktisches Feature von Sass. Mixins erlauben es euch, ganze Blöcke von CSS-Eigenschaften wieder zu verwenden und mit Variablen anzureichern. Mit Hilfe von Mixins können häufig verwendete oder lästige CSS-Abschnitte wie der Code für Animationen, Vendor-Präfixe für verschiedene Browser oder komplexe Website-Module wiederverwendet werden. Neben Variablen gehören Mixins wohl zu den meist genutzten Funktionen von Sass

Die Definition eines Mixins erfolgt über die Angabe `@ mixin NAME-DES-MIXINS {}`. Anschließend kann über `@include NAME-DES-MIXINS;` innerhalb eines CSS-Selektors auf den Code-Block referenziert werden.

```
@mixin headline-style {
  font: {
    family: Arial, sans-serif;
    size: 1.5rem;
  }
  color: blue;
  margin-bottom: 1.5rem;
}

h2 {
  @include headline-style;
}
.search-results h3 {
  @include headline-style;
  font-weight: normal;
}
```

```
//kompiliert zu
h2 {
  font-family: Arial, sans-serif;
  font-size: 1.5rem;
  color: blue;
  margin-bottom: 1.5rem;
}

.search-results h3 {
  font-family: Arial, sans-serif;
  font-size: 1.5rem;
  color: blue;
  margin-bottom: 1.5rem;
  font-weight: normal;
}
```

MIXINS MIT ARGUMENTEN

Mixins können auch mit Argumenten versehen werden. Argumente erlauben es, Variablen innerhalb des Mixins festzulegen, die anschließend beim Inkludieren des Mixins mit einem Wert gefüllt werden können. Mit Hilfe von Argumenten können einzelne Bereiche eines Mixins individualisiert werden. Der Aufbau des Mixins wird dabei geringfügig erweitert - nach dem Namen des Mixin folgt innerhalb von runden Klammern die Variable.

Im Beispiel wird die Farbe als Argument festgelegt

```
@mixin headline-style($color) {  
    font: {  
        family: Arial, sans-serif;  
        size: 1.5rem;  
    }  
    color: $color;  
    margin-bottom: 1.5rem;  
}  
h2 {  
    @include headline-style(green);  
}  
.search-results h3 {  
    @include headline-style(red);  
}  
  
//kompiliert zu  
h2 {  
    font-family: Arial, sans-serif;  
    font-size: 1.5rem;  
    color: green;  
    margin-bottom: 1.5rem;  
}  
.search-results h3 {  
    font-family: Arial, sans-serif;  
    font-size: 1.5rem;  
    color: red;  
    margin-bottom: 1.5rem;  
}
```

MIXINS MIT MEHREREN ARGUMENTEN

Es ist auch möglich mehrere Argumente für ein Mixin zu definieren. In diesem Fall werden innerhalb der runden Klammern einfach mehrere Variablen durch Komma getrennt angegeben.

Das folgende Beispiel verwendet zwei Variablen - eine für die Farbe und eine für die Fettung der Überschrift

```
@mixin headline-style($color, $font-weight) {  
  font: {  
    family: Arial, sans-serif;  
    size: 1.5rem;  
    weight: $font-weight;  
  }  
  color: $color;  
  margin-bottom: 1.5rem;  
}  
h2 {  
  @include headline-style(green, bold);  
}  
.search-results h3 {  
  @include headline-style(red, normal);  
}  
  
//kompiliert zu  
h2 {  
  font-family: Arial, sans-serif;  
  font-size: 1.5rem;  
  color: green;  
  margin-bottom: 1.5rem;  
  font-weight: bold;  
}  
.search-results h3 {  
  font-family: Arial, sans-serif;  
  font-size: 1.5rem;  
  color: red;  
  margin-bottom: 1.5rem;  
  font-weight: normal;  
}
```

MIXINS MIT ARGUMENTEN UND STANDARDWERTEN

Es bietet sich häufig an, Standardwerte für Argumente festzulegen. Wenn das geschehen ist, kann das Mixin auch ohne die Angabe des entsprechenden Arguments aufgerufen werden. Es wird dann erwartungsgemäß der Standardwert verwendet. Soll vom Standard abgewichen werden, muss das Argument wie bereits bekannt angegeben werden.

Standardwerte haben vor allem den Vorteil, dass ihr beim Aufruf (`@include`) des Mixins auf Argumente verzichten können.

Das nachfolgende Beispiel verwendet ein Mixin mit zwei Argumenten. Das Argument für `font-weight` hat dabei den Standardwert `bold` zugewiesen bekommen. Der Standardwert wird mit Doppelpunkt getrennt direkt hinter die Variable geschrieben

```
@mixin headline-style($color, $font-weight:bold) {  
  font: {  
    family: Arial, sans-serif;  
    size: 1.5rem;  
    weight: $font-weight;  
  }  
  color: $color;  
  margin-bottom: 1.5rem;  
}  
h2 {  
  @include headline-style(green);  
}  
.search-results h3 {  
  @include headline-style(red, normal);  
}  
  
//kompiliert zu  
h2 {  
  font-family: Arial, sans-serif;  
  font-size: 1.5rem;  
  color: green;  
  margin-bottom: 1.5rem;  
  font-weight: bold;  
}  
.search-results h3 {  
  font-family: Arial, sans-serif;  
  font-size: 1.5rem;  
  color: red;  
  margin-bottom: 1.5rem;  
  font-weight: normal;  
}
```

BEISPIELE FÜR MIXINS

Folgendes Beispiel vereinfacht den Code für die Manipulation des Box-Modells. Als Standardwert wird **border-box** festgelegt. Darüber hinaus habt ihr die Möglichkeit über das Mixin die angegebenen Vendor-Präfixe zu entfernen, sollten sie eines Tages nicht mehr im Projekt benötigt werden.

```
@mixin box-sizing ($type:border-box) {  
    -webkit-box-sizing: $type;  
    -moz-box-sizing: $type;  
    box-sizing: $type;  
}  
*, *:before, *:after {  
    @include box-sizing;  
}  
.element {  
    @include box-sizing(content-box);  
}  
  
//kompiliert zu  
*, *:before, *:after {  
    -webkit-box-sizing: border-box;  
    -moz-box-sizing: border-box;  
    box-sizing: border-box;  
}  
.element {  
    -webkit-box-sizing: content-box;  
    -moz-box-sizing: content-box;  
    box-sizing: content-box;  
}
```

Das zweite Beispiel verwendet ein Mixin mit drei Argumenten um den Code für CSS-Transitions zu vereinfachen

```
@mixin transition($transition-property, $transition-time, $transition-method) {  
    -webkit-transition: $transition-property $transition-time $transition-method;  
    -moz-transition: $transition-property $transition-time $transition-method;  
    -o-transition: $transition-property $transition-time $transition-method;  
    transition: $transition-property $transition-time $transition-method;  
}  
div {  
    @include transition(all, .1s, ease-in-out);  
}  
  
//kompiliert zu  
div {  
    -webkit-transition: ease-in-out all .1s;  
    -moz-transition: ease-in-out all .1s;  
    -o-transition: ease-in-out all .1s;  
    transition: ease-in-out all .1s;  
}
```

EXTEND

Häufig unterscheidet sich der CSS-Code für verschiedene Selektoren nur durch wenige Eigenschaften. Eine Box im Inhaltsbereich kann beispielsweise sowohl für Fehlermeldungen als auch für Informationen genutzt werden. Die Grundstruktur des Moduls bleibt jedoch gleich, lediglich die Farbe der Außenlinie wird verändert.

In OOCSS (Objektorientiertem CSS) wird daher normalerweise zunächst eine Basis-Klasse (`.box`) mit allgemeinen Gestaltungsangaben erstellt. Erweiternde Klassen mit dem Präfix der Basis-Klasse (z. B. `.box-error`, `.box-info`) überschreiben dann die notwendigen Eigenschaften. In Sass existiert zu diesem Zweck die Funktion `@extend`. Sie erweitert Module um den Code anderer Module und kombiniert die Eigenschaften in durch Komma getrennten Listen.

Extend erweitert einen Selektor über den Befehl `@exted NAME-DES-SELEKTORS`. Dabei ist es irrelevant ob es sich beim Referenz-Selektor um eine Klasse, eine ID oder einen anderen Selektor handelt.

```
.box {  
    padding: 1em;  
    color: black;  
    border: 2px solid silver;  
}  
.box-info {  
    @extend .box;  
    border-color: yellow;  
}  
.box-error {  
    @extend .box;  
    border-color: red;  
}  
  
//Kompiliert zu  
.box, .box-info, .box-error {  
    padding: 1em;  
    color: black;  
    border: 2px solid silver;  
}  
.box-info {  
    border-color: yellow;  
}  
.box-error {  
    border-color: red;  
}
```

PLATZHALTER (%) FÜR @EXTEND EINSETZEN

Nun kommt es recht häufig vor, dass ein über `@extend` erweiterter Selektor alleine gar nicht vorkommen kann. Im soeben gesehenen Beispiel trifft dies auf die Klasse `.box` zu. `.box` wird nur in Verbindung mit `.box-info` oder `.box-error` eingesetzt. Typisch ist dieses Verhalten auch bei Webfont-Icons wie Font Awesome oder Genericon. Wenn wir uns den kompilierten Code des letzten Beispiels anschauen, merken wir, dass der Selektor `.box` überflüssig ist.

Um solch unnötigen Code zu vermeiden, sollten Platzhalter eingesetzt werden. Ein Platzhalter wird damit & eingeleitet. Wenn die Eigenschaften eines Platzhalters über `@extend` auf andere Klassen übertragen wurden, kompiliert Sass diese Eigenschaften wie zuvor gesehen. Der Selektor des Platzhalters taucht allerdings nicht mehr alleine auf.

Immer wenn ihr einen Selektor ausschließlich erzeugt um in später über `@extend` zu erweitern, verwendet statt dessen einen Platzhalter!

```
%box {  
  padding: 1em;  
  color: black;  
  border: 2px solid silver;  
}  
.box-info {  
  @extend &box;  
  border-color: yellow;  
}  
.box-error {  
  @extend &box;  
  border-color: red;  
}  
  
//Kompiliert zu  
.box-info, .box-error {  
  padding: 1em;  
  color: black;  
  border: 2px solid silver;  
}  
.box-info {  
  border-color: yellow;  
}  
.box-error {  
  border-color: red;  
}
```

@extend kann auch mehrfach eingesetzt werden. Achtet allerding darauf, den Überblick nicht zu verlieren.

```
%box {  
    padding: 1em;  
    color: black;  
    border: 2px solid silver;  
}  
.box-big {  
    padding: 2em;  
    font-size: 2em;  
    border-width: 4px;  
}  
.box-info {  
    @extend %box;  
    @extend .box-big;  
    border-color: yellow;  
}  
.box-error {  
    @extend %box;  
    @extend .box-big;  
    border-color: red;  
}  
  
//Kompiliert zu  
.box-info, .box-error {  
    padding: 1em;  
    color: black;  
    border: 2px solid silver;  
}  
.box-big, .box-info, .box-error {  
    padding: 2em;  
    font-size: 2em;  
    border-width: 4px;  
}  
.box-info {  
    border-color: yellow;  
}  
.box-error {  
    border-color: red;  
}
```

EXTEND vs. MIXIN

Viele Web Designer fragen sich, worin der Vorteil von `@extend` gegenüber Mixins (`@include`) besteht. Häufig arbeiten beide Funktionen gleich gut. Der generierte CSS-Code unterscheidet sich aber deutlich. Mixins können schnell zu Wiederholungen im generierten CSS-Code führen. Das bläst den Code unnötig auf und schadet der Performance.

Beispiel auf der nächsten Folie

```
%icon {
  display: inline-block;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Webfont';
  font-weight: normal;
  font-style: normal;
}

.icon-upload {
  @extend %icon;
  /* Styles für das Upload-Icon */
}

.icon-download {
  @extend %icon;
  /* Styles für das Download-Icon */
}
```

```
// kompiliert zu

.icon-upload, .icon-download {
  display: inline-block;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Webfont';
  font-weight: normal;
  font-style: normal;
}

.icon-upload {
  /* Styles für das Upload-Icon */
}

.icon-download {
  /* Styles für das Download-Icon */
}
```

Mit Extend

Mit Mixin

```
@ mixin icon {
  display: inline-block;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Webfont';
  font-weight: normal;
  font-style: normal;
}

.icon-upload {
  @include icon;
  /* Styles für das Upload-Icon */
}

.icon-download {
  @include icon;
  /* Styles für das Download-Icon */
}
```

```
// kompiliert zu

.icon-upload {
  display: inline-block;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Webfont';
  font-weight: normal;
  font-style: normal;
  /* Styles für das Upload-Icon */
}

.icon-download {
  display: inline-block;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  font-family: 'Webfont';
  font-weight: normal;
  font-style: normal;
  /* Styles für das Download-Icon */
}
```

DATEIEN IMPORTIEREN - @IMPORT

Um bei komplexen Websites die Übersicht zu behalten, bietet es sich an große SCSS-Dateien in mehrere kleinere SCSS-Dateien aufzuteilen. Typische Ausgliederungen sind Mixins, Variablen, das Gestaltungsraster, ein Reset/Normalizer und große Strukturelemente (z. B. der Header oder die Navigation).

Mit **@import** können im Haupt-Stylesheet die ausgelagerten Stylesheets importiert werden. Sass kompiliert in diesem Fall alle Dateien zu einem gemeinsamen Stylesheet um HTTP-Requests zu sparen. Für den Webdesigner bleibt die Übersicht mit verschiedenen SCSS-Dokumenten jedoch erhalten.

```
// reset.scss
* {
  margin: 0;
  padding: 0;
}

//style.scss
@import "reset.scss";
body {
  background: red;
}
```

```
// reset.css
* {
  margin: 0;
  padding: 0;
}

//style.css
* {
  margin: 0;
  padding: 0;
}
body {
  background: red;
}
```

IMPORTIERTES STYLESHEETS NICHT DIREKT KOMPILIEREN

Ein Stylesheet kann über `@import "NAME-DES-STYLESHEETS.css"` importiert werden.

Wenn einzelne Stylesheets nicht direkt kompiliert werden sollen, verwendet einen Unterstrich (underscore _) als Präfix.
`(_reset.scss)`

Nach der Kompilierung befindet sich nur ein CSS-Dokument im Ziel-Ordner. Die Datei `style.css` enthält dennoch den Code beider Stylesheets.

Wenn ihr verschiedene Stylesheets verwendet um die Übersicht innerhalb des Projekts zu verbessern, achtet darauf die Dateien anschließend in der korrekten Reihenfolge zu importieren. Wenn ihr in einem Stylesheet Variablen verwendet, die wiederum in einem anderen Dokument definiert wurden, können diese Variablen nur fehlerfrei kompiliert werden, wenn die Datei mit den Variablen zuvor importiert wurde. Es bietet sich daher an, ein SCSS-Dokument mit allen Variablen anzulegen und vor allen anderen Dokumenten zu importieren.

Die in den Mixins verwendeten Variablen können problemlos verwendet werden, da die Datei `variables.scss` vor `mixins.scss` importiert wurde. Würden sich allerdings auch Variablen in `normalizer.scss` befinden (was normalerweise nicht der Fall ist), wäre das Ergebnis fehlerhaft ("Variable `undefined`").

```
//_reset.scss
* {
  margin: 0;
  padding: 0;
}

//style.scss
@import "_reset.scss";

body {
  background: red;
}
```

```
//style.scss

@import "_normalize.scss";
@import "_variables.scss";
@import "_mixins.scss";
```

OPERATOREN

```
// Addition
.addition {
  width: 500px + 500px;
}

// kompiliert zu
.addition {
  width: 1000px;
}

// Subtraktion
.subtraction {
  width: 1200px - 200px;
}

// kompiliert zu
.subtraction {
  width: 1000px;
}

// Multiplikation
.multiplication {
  width: 250px * 4;
}

// kompiliert zu
.multiplication {
  width: 1000px;
}

// Division
.division {
  width: (2000px / 2);
}

// kompiliert zu
.division {
  width: 1000px;
}

// Prozentrechnung
.percentage {
  width: 600px / 960px * 100%;
}

// kompiliert zu
.percentage {
  width: 62.5%;
}
```

In Sass stehen die typischen Rechenoperatoren zur Verfügung. Mit ihrer Hilfe können z. B. Gestaltungsraster oder der goldene Schnitt berechnet werden.

Bette beachtet, dass zwischen den Werten und den Operatoren ein Leerzeichen stehen muss. z. B. **100px - 50px**. Ohne das Leerzeichen interpretiert Sass die zwei Zahlen als Negativwert. **100px-50px** ergibt daher einen Fehler.

OPERATOREN

```
// Prozentrechnung
.percentage {
    width: 600px / 960px * 100%;
}
// kompiliert zu
.percentage {
    width: 62.5%;
}

// Punkt vor Strich
div:nth-of-type(1) {
    width: 600px + 360px / 2;
}
div:nth-of-type(1) {
    width: (600px + 360px) / 2;
}

// kompiliert zu
div:nth-of-type(1) {
    width: 780px;
}
div:nth-of-type(1) {
    width: 480px;
}

// Rechnung mit Variablen
$width-site: 700px;
$width-sidebar: 260px;
$full-width: $width-site + $width-sidebar;
#container {
    max-width: $full-width;
}
// kompiliert zu
#container {
    max-width: 960px;
}
// Einheiten hinzufügen
$base-size: 1;
div {
    width: $base-size * 50%;
    font-size: $base-size * 1em;
    border: ($base-size * 5px) solid black;
}
// kompiliert zu
div {
    width: 50%;
    font-size: 1em;
    border: 5px solid black;
}
```

MEDIA QUERIES

Seit dem Siegeszug von Responsive Design sind Media Queries aus Web-Projekten nicht mehr wegzudenken. Unabhängig davon, ob Mobile- oder Desktop-First gearbeitet wird, werden die Media Queries meist am Ende des Dokuments notiert. Innerhalb der verschiedenen Media-Query-Abschnitte werden zuvor vergebene CSS-Anweisungen überschrieben. Bei komplexen Websites ist diese Handhabung häufig etwas unübersichtlich und lästig. Wir müssen während der Entwicklung einer Website ständig im Code springen und Änderungen mal oben und mal unten im Stylesheet vornehmen.

INLINE MEDIA QUERIES

Mit Sass sind sogenannte Inline Media Queries möglich. Das Beispiel ist Mobile First aufgebaut und erhöht die Schriftgröße, sowie die Höhe des Website-Headers bei größer werdendem Viewport

Gemessen am Umfang dieses Beispiels mag der Vorteil dieser Technik gering erscheinen. Bei großen Projekten stellen Inline Media Queries allerdings einen enormen Zeitgewinn dar, da alle Styles eines Elements in einem Code-Abschnitt notiert werden können. Unabhängig davon, in welchem Viewport das Element betrachtet wird.

Zweifel an der Effizienz dieser Lösung könnten auftreten, da im komplizierten Stylesheet Media Queries doppelt auftauchen. Das ist zwar in der Tat nicht optimal, wirkt sich in der Praxis jedoch nicht merklich negativ aus. Der Größenunterschied des Stylesheets liegt bei komprimierten CSS-Dateien im Byte- oder im kleinen Kilobyte-Bereich. Für mich steht die komfortable Pflege des Stylesheets klar im Vordergrund

INLINE MEDIA QUERIES

```
body {  
    font-size: 1em;  
  
    @media screen and (min-width: 32em) {  
        font-size: 1.1em;  
    }  
    @media screen and (min-width: 50em) {  
        font-size: 1.2em;  
    }  
}  
  
header[role="banner"] {  
    height: 150px;  
    @media screen and (min-width: 32em) {  
        height: 200px;  
    }  
    @media screen and (min-width: 50em) {  
        height: 250px;  
    }  
}
```

```
//kompiliert zu  
body {  
    font-size: 1em;  
}  
@media screen and (min-width: 32em) {  
    body {  
        font-size: 1.1em;  
    }  
}  
@media screen and (min-width: 50em) {  
    body {  
        font-size: 1.2em;  
    }  
}  
header[role="banner"] {  
    height: 150px;  
}  
@media screen and (min-width: 32em) {  
    header[role="banner"] {  
        height: 200px;  
    }  
}  
@media screen and (min-width: 50em) {  
    header[role="banner"] {  
        height: 250px;  
    }  
}
```

Media Queries werden in einem Dokument viele Male definiert, insbesondere wenn wir die Media Queries inline schreiben. Es bietet sich also an, die Breakpoints als Variablen zu definieren. Außerdem können auf Variablen definierten Breakpoints Rechenoperatoren anwenden.

VARIABLEN FÜR MEDIA QUERIES VERWENDEN

```
$mq-medium: 32em;  
$mq-large: 50em;  
  
body {  
    font-size: 1em;  
  
    @media screen and (min-width: $mq-medium) and (max-width: $mq-large - 0.1) {  
        font-size: 1.1em;  
    }  
    @media screen and (min-width: $mq-large) {  
        font-size: 1.2em;  
    }  
}  
  
header[role="banner"] {  
    height: 150px;  
    @media screen and (min-width: $mq-medium) {  
        height: 200px;  
    }  
    @media screen and (min-width: $mq-large) {  
        height: 250px;  
    }  
}
```

//kompliert zu

```
body {  
    font-size: 1em;  
}  
@media screen and (min-width: 32em) and (max-width: 49.9em) {  
    body {  
        font-size: 1.1em;  
    }  
}  
@media screen and (min-width: 50em) {  
    body {  
        font-size: 1.2em;  
    }  
}  
header[role="banner"] {  
    height: 150px;  
}  
@media screen and (min-width: 32em) {  
    header[role="banner"] {  
        height: 200px;  
    }  
}  
@media screen and (min-width: 50em) {  
    header[role="banner"] {  
        height: 250px;  
    }  
}
```

Wir können Media Queries auch mit Hilfe von Mixins erzeugen. Techniken gibt es in diesem Zusammenhang viele – hier ein Lösungsansatz mit Hilfe von [@content](#).

Mixins mit Argumenten haben wir bereits kennengelernt, mit [@content](#) kann ein Inhalt in ein Mixin eingeschleust werden. Das Mixin fungiert dann als Container um den Inhalt herum. Für Media Queries ist diese Technik prädestiniert.

Das folgende Beispiel zeigt, wie Inhalt mit Hilfe von [@content](#) in ein Mixin eingeschleust werden kann. Bei der Definition des Mixins steht [@content](#) an der Stelle, an der später beim Kompilieren der Inhalt eingefügt werden soll:

```
@ mixin NAME-DES-MIXINS {  
  .selector {  
    @content;  
  }  
}  
  
@ include NAME-DES-MIXINS {  
  /* INHALT */  
}  
  
// kompiliert zu  
.selector {  
  /* INHALT */  
}
```

MEDIA QUERIES MIT MIXINS UND @CONTENT

```
/* Variablen */  
$mq-medium: 32em;  
$mq-large: 50em;  
  
/* Mixins */  
@mixin breakpoint($mq-width) {  
    @media screen and (min-width: $mq-width) {  
        @content  
    }  
}  
body {  
    font-size: 1em;  
    @include breakpoint($mq-medium) {  
        font-size: 1.1em;  
    }  
    @include breakpoint($mq-large) {  
        font-size: 1.2em;  
    }  
}  
  
//kompiliert zu  
body {  
    font-size: 1em;  
}  
@media screen and (min-width: 32em) {  
    body {  
        font-size: 1.1em;  
    }  
}  
@media screen and (min-width: 50em) {  
    body {  
        font-size: 1.2em;  
    }  
}
```

KOMMENTARE

In CSS steht nur eine Schreibweise für Kommentare zur Verfügung (`/* Kommentar */`). In SCSS können insgesamt drei verschiedene Kommentare verwendet werden.

Wichtiger mehrzeiliger Kommentar

Wenn der Kommentar auch im komprimierten Stylesheets (Output-Style **compressed**) sichtbar sein soll, verwendet folgende Syntax. Diese Kommentare werden auch als "loud comments" bezeichnet.

Einzeiliger Kommentar

Wie aus PHP und anderen Programmiersprachen bekannt, existiert in Sass auch der einzeilige Kommentar. Dieser Kommentar wird bei allen Output-Typen entfernt. Er bietet sich für umfangreiche Dokumentationen des Codes an, die unabhängig vom Output-Style in der Live-Version der Website nicht erscheinen sollen.

```
//scss
/* Dieser mehrzeilige Kommentar
ist auch im kompilierten Stylesheet sichtbar.
Allerdings nicht in komprimierten Stylesheets! */
```

```
//scss
/*! Ein mit Ausrufezeichen arkierter,
mehrzeiliger Kommentar ist auch in
komprimierten Stylesheets sichtbar !*/
```

```
//scss
// Ein einzeiliger Kommentar ist im kompilierten
// Stylesheet nicht enthalten
```

Mit Sass ist es auch möglich zu programmieren, was in CSS sonst bekanntlich nicht möglich ist. mit Hilfe von if/else-Bedingungen, Schleifen und Funktionen könnt ihr Logiken und Abhängigkeiten in eurem Stylesheets einbauen und euren Workflow noch weiter optimieren.

Mit Hilfe von `@if` könnt ihr überprüfen, ob eine Bedingung zutrifft. Wenn das der Fall ist, wird ein entsprechender Codeabschnitt kompiliert. Um die Überprüfung vorzunehmen, stehen euch verschiedene Operatoren zur Verfügung

- `==` gleich
- `!=` ungleich
- `>` größer als
- `<` kleiner als
- `>=` größer gleich
- `<=` kleiner gleich

```
div {  
    @if 3 > 1 { background: red; }  
    @if 3 == 1 { background: green; }  
    @if 3 < 1 { background: blue; }  
}  
  
// kompiliert zu  
div {  
    background: red;  
}
```

Im folgenden Beispiel wird geprüft, ob die Sidebar links oder rechts vom Inhalt positioniert wurde. Wenn sie sich links befindet, fügen wir rechts einen Abstand hinzu. Bei einer Sidebar auf der rechten Seite wird links der Abstand ergänzt.

```
$sidebar-alignment: left;  
.sidebar {  
  float: $sidebar-alignment;  
  @if $sidebar-alignment== left {  
    margin-right: 2em;  
  }  
  @else {  
    margin-left: 2em;  
  }  
}  
  
// kompiliert zu  
.sidebar {  
  float: left;  
  margin-right: 2em;  
}
```

```
$sidebar-alignment: bottom;  
.sidebar {  
  @if $sidebar-alignment == left {  
    margin-right: 2em;  
    float: $sidebar-alignment;  
    width: 30%;  
  }  
  @else if $sidebar-alignment == right {  
    margin-left: 2em;  
    float: $sidebar-alignment;  
    width: 30%;  
  }  
  @else {  
    margin-top: 2em;  
    float: left;  
    width: 100%;  
  }  
}  
  
// kompiliert zu  
.sidebar {  
  float: none;  
  margin-top: 2em;  
  width: 100%;  
}
```

Die `@for`-Schleife ermöglicht es euch, Code beliebig oft zu wiederholen und nach jeder Wiederholung anzupassen. Um die Anzahl der Wiederholungen festzulegen, werden ein Start- und ein Endwert definiert. Anschließend habt ihr zwei Möglichkeiten der Ausführung

```
.@for $variable from [Startwert] through [Endwert]  
.@for $variable from [Startwert] to [Endwert]
```

Der Unterschied liegt in der Anzahl der Schleifendurchläufe. Angenommen, der Startwert ist **1** und der Endwert **4**, dann wird die Schleife mit **through** viermal ausgeführt, bei **to** allerdings nur dreimal. Sie läuft bis zum Endpunkt, stoppt aber davor

Die Schleife beginnt mit `@for` – damit wird Sass darüber informiert, dass wir jetzt eine `@for`-Schleife schreiben wollen. Es folgt die Variable `$i`. Der Wert dieser Variable wird bei jedem Schleifendurchlauf um `1` erhöht. Ihr könnt die Variable nennen wie ihr möchten, `$i` hat sich allerdings als Bezeichnung für die sogenannte Iterations-Variable eingebürgert.

Nach der Variable folgt der Schlüsselbegriff `from` gefolgt von einem entsprechenden Zahlwert. Der Schlüsselbegriff `through` mit folgendem Zahlwert legt fest wie oft die Schleife durchlaufen werden soll.

Zwischen den geschwungenen Klammern steht der Code, der bei jedem Schleifendurchlauf erneut ausgegeben werden soll. Da der Code sich an zwei Stellen verändern soll, schleusen wir die Variable `$i` ein.

`$i` wird zunächst verwendet, um den Namen der CSS-Klasse mit einem Zahlwert zu ergänzen. Da die Variable an dieser Stelle interpoliert wird, muss sie mit `# { }` umschlossen werden, um keinen Fehler zu erzeugen. Interpolation bedeutet, dass eine Sass-Variable an einer anderen Stelle im Code importiert wird. Etwas weiter rechts verwenden wir `$i` noch einmal. Diesmal berechnen wir die Breite des Elements, indem wir mit dem Wert von `$i` multiplizieren.

```
@for $i from 1 through 4 {  
  .class-#${$i} { width: 10px * $i; }  
}
```

//kompiliert zu
.class-1 {
 width: 10px;
}
.class-2 {
 width: 20px;
}
.class-3 {
 width: 30px;
}
.class-4 {
 width: 40px;
}

```
@for $i to 1 through 4 {  
  .class-#${$i} { width: 10px * $i; }  
}
```

//kompiliert zu
.class-1 {
 width: 10px;
}
.class-2 {
 width: 20px;
}
.class-3 {
 width: 30px;
}

Interpolation:
Das bedeutet dass ein Ausdruck durch den zugewiesenen Wert ersetzt wird.

BEISPIEL 2

```
$grid-columns: 12;  
@for $i from 1 through $grid-columns {  
  .column-span-#${$i} {  
    width: 100% / $grid-columns * $i;  
  }  
}  
  
// kompiliert zu  
.column-span-1 { width: 8.33333%; }  
.column-span-2 { width: 16.66667%; }  
.column-span-3 { width: 25%; }  
.column-span-4 { width: 33.33333%; }  
.column-span-5 { width: 41.66667%; }  
.column-span-6 { width: 50%; }  
.column-span-7 { width: 58.33333%; }  
.column-span-8 { width: 66.66667%; }  
.column-span-9 { width: 75%; }  
.column-span-10 { width: 83.33333%; }  
.column-span-11 { width: 91.66667%; }  
.column-span-12 { width: 100%; }
```

Mit einer `@each`-Schleife können Listen mit Elementen abgearbeitet werden.

Ich möchte die Code-Ausgabe für eine Liste von Icons automatisieren. Alle Icon-Grafiken liegen im selben Ordner (`img`) und sollen verschiedenen CSS-Klassen als `background` zugewiesen werden. Dazu schreiben wir alle Icons in eine Liste und arbeiten die Liste anschließend der Reihe nach ab.

Die Variable `$icon-list` beinhaltet die mit Komma getrennte Liste der Icons. Mit `@each` teilen wir Sass mit, dass nur eine `@each`-Schleife beginnt. Die Variable `$icon` ist ein Platzhalter und repräsentiert der Reihe nach die Elemente innerhalb der Liste `$icon-list`. Beim ersten Schleifendurchlauf steht `$icon` für `home`, beim zweiten Durchlauf für `rss` usw.

Zwischen den geschwungenen Klammern steht der Code, den wir ausgeben möchten. hier wird die Variable `$icon` zweimal verwendet. Achtet darauf, dass sie interpoliert werden muss (`#{}`)

```
$icon-list: home, rss, google, facebook, twitter;  
@each $icon in $icon-list {  
  .icon-#{$icon} {  
    background: url('img/icon-#${$icon}.png');  
  }  
}  
  
// kompiliert zu  
.icon-home {  
  background: url('img/icon-home.png');  
}  
.icon-rss {  
  background: url('img/icon-rss.png');  
}  
.icon-google {  
  background: url('img/icon-google.png');  
}  
.icon-facebook {  
  background: url('img/icon-facebook.png');  
}  
.icon-twitter {  
  background: url('img/icon-twitter.png');  
}
```

MEHRFACHZUORDNUNG

```
// kompiliert zu
.icon-home {
  height: 64px;
  width: 64px;
  background: url('img/icon-home.svg');
}
.icon-rss {
  height: 16px;
  width: 16px;
  background: url('img/icon-rss.gif');
}
.icon-google {
  height: 32px;
  width: 32px;
  background: url('img/icon-google.png');
}
.icon-facebook {
  height: 32px;
  width: 32px;
  background: url('img/icon-facebook.png');
}
.icon-twitter {
  height: 32px;
  width: 32px;
  background: url('img/icon-twitter.png');
}
```

```
$icon-list: (home svg 64px),
(rss gif 16px),
(google png 32px),
/facebook png 32px),
(twitter png 32px);
```

```
@each $icon in $icon-list {
  $icon-name:  nth($icon, 1);
  $icon-format: nth($icon, 2);
  $icon-size:   nth($icon, 3);

  .icon-#{$icon-name} {
    height: $icon-size;
    width: $icon-size;
    background: url('img/icon-#{$icon-name}.#{$icon-format}');
  }
}
```

Es ist auch möglich, eine Liste von Objekten mit mehreren Eigenschaften abzuarbeiten. Ich habe die Liste mit Icons dazu um ein Dateiformat und um die Größe der Grafik in Pixel erweitert.

Zunächst wird die Liste der Icons definiert. Im Gegensatz zum letzten Beispiel besitzt jedes Icon nun mehrere Eigenschaften. Die Eigenschaften eines Icons wurden mit Leerzeichen getrennt in runden Klammern notiert.

Es folgt die bereits bekannte `@each`-Schleife. Da jedes Icon nun mehrere Eigenschaften besitzt, sind weitere Variablen notwendig (`$icon-name`, `$icon-format`, `$icon-size`). Mit Hilfe der Sass-Funktion `nth()` wird definiert, welcher Wert eines Icons den Variablen zugewiesen werden soll. `nth($icon, 2)`; steht demnach für den zweiten Wert eines Icons, also für das Dateiformat.

Anschließend folgt der CSS-Code, den wir später mit Hilfe der Schleifen kompilieren möchten. innerhalb des Code-Abschnitts werden die soeben definierten Variablen verwendet (Achtung, Interpolation!).

Mit `@while` könnt ihr eine Schleife so lange ausführen lassen, bis eine Bedingung nicht mehr zutrifft.

Das folgende Beispiel zeigt, wie die While-Schleife eingesetzt werden kann um Offset-Klassen im Abstand von 10 Pixeln zu erzeugen. Offset-Klassen werden in vielen Gestaltungsrastern genutzt um Elemente einzurücken.

Die While-Schleife wird mit `@while` eingeleitet. Es folgt die Bedingung: Die Schleife soll so lange durchlaufen bis `$i` den Wert 10 erreicht hat. Damit die Variable `$i` nicht undefiniert ist, wurde vor dem Beginn der Schleife `$i` der Wert 1 zugewiesen.

Zwischen den runden Klammern folgt der CSS-Code, der innerhalb der Schleife ausgegeben werden soll. Wir erstellen eine Klasse namens `.offset-` und erweitern sich um den Wert von `$i` (Achtung: Interpolation!). Anschließend ergeben wir `margin-left` und multiplizieren hier den Wert von `$i` mit 10 Pixel um eine Einrückung von 10-Pixel-Schritten zu erreichen.

Als letztes erhöhen wir mit `$i: $i + 1;` den Wert von `$i` bei jedem Schleifendurchlauf um 1. Andernfalls würden wir eine Endlosschleife erzeugen.

```
$i: 1;

@while $i <= 10 {
  .offset-#${$i} {
    margin-left: $i * 10px;
    $i: $i + 1;
  }
}

//kompiliert zu
.offset-1 { margin-left: 10px; }
.offset-2 { margin-left: 20px; }
.offset-3 { margin-left: 30px; }
.offset-4 { margin-left: 40px; }
.offset-5 { margin-left: 50px; }
.offset-6 { margin-left: 60px; }
.offset-7 { margin-left: 70px; }
.offset-8 { margin-left: 80px; }
.offset-9 { margin-left: 90px; }
.offset-10 { margin-left: 100px; }
```

@FUNCTION

Individuelle Sass-Funktionen mit **@function** sind mixins sehr ähnlich, mit einem entscheidenden Unterschied: Mixins geben einen Codeabschnitt aus, Funktionen hingegen einen Rückgabewert. Dieser Wert kann jedem Sass-Datentyp entsprechen.

Datentypen in Sass

- Zahlenwerte (**25, 1.5, 30px**)
- Text-Strings ("**foo**", '**bar**', **baz**)
- Farben (**#ff00ff, rgba(255,0,123,0.5)**)
- Boolesche Wert (**true** oder **false**)
- Werte-Listen (**10px 20px** oder **Helvetica, Arial, sans-serif**)
- Maps ((**schluessel1: wert1, schlüssel2: wert2**))

Funktionen werden mit **@function** eingeleitet. Es folgt der Name der Funktion, im folgenden Beispiel **multiply-a-with-b**. Zwischen den runden Klammern werden – wie bei Mixins – die Argumente notiert. Zwischen den geschwungenen Klammern folgt der Inhalt der Funktion, hier werden die Argumente benutzt. Mit **@return** wird der Wert der Funktion zurückgegeben. Je nachdem was in einer Funktion geschieht, kann **@return** unterschiedliche Werte annehmen.

```
@function multiply-a-with-b($a, $b) {
  @return $a * $b;
}

div {
  width: multiply-a-with-b(2px,7);
}

//kompiliert zu
div {
  width: 14px;
}
```

```
@function color-theme($color) {
  @if $color == spring {
    @return lime;
  }
  @else if $color == summer{
    @return tomato;
  }
  @else if $color == autumn {
    @return gold;
  }
  @else if $color == winter {
    @return cyan;
  }
}
div {
  background: color-theme(autumn);
}

// kompiliert zu
div {
  background: gold;
}
```

FRAMEWORKS FÜR SASS

Für Sass existieren verschiedene Erweiterungen, allen voran das populäre CSS-Framework Compass. Compass kann im Umfeld von Sass mit jQuery im Umfeld von JavaScript verglichen werden.

Bei jQuery besteht das Problem dass viele Webdesigner die JavaScript-Grundlagen nicht verstehen und JavaScript und jQuery nicht unterscheiden können. Ihnen ist nicht klar, wo das eine anfängt und das andere aufhört. Diese Gefahr gibt es auch bei Sass, da viele Bücher und Online-Tutorials von vorne herein auf Compass basieren, ohne klar herauszustellen, welche Funktionen Sass-Standards und welche spezielle Compass-Funktionen sind.

COMPASS

Compass stellt sehr viele hilfreiche Tools und Snippets (z.B. Mixins) bereit, die die Arbeit mit CSS enorm erleichtern. Das Framework war das erste Sass-Framework überhaupt und wurde von Christopher Eppstein entwickelt, der wiederum auch im Core-Entwicklerteam von Sass mitwirkt. Diese Voraussetzung kann sicherlich als optimal bezeichnet werden

Compass stellt unter anderem Funktionen für den browserübergreifenden Einsatz von (experimentellen) CSS-Funktionen bereit. Auch die Erstellung von CSS-Sprites und Data URIs wird durch das Framework enorm erleichtert. Compass erstellt auf Wunsch auch eine Standard-Ordnerstruktur und richtet Basis-Dateien ein. Auch die Arbeit mit wiederverwertbaren und darstellungsunabhängigen Modulen wird erleichtert. Für Compass wiederum existieren diverse Drittanbieter-Tools und Plugins. Das Framework nimmt im Umfeld von Sass daher eine sehr wichtige Rolle ein, steht allerdings auch aufgrund seiner angeblichen "Schwerfälligkeit" in der Kritik.

Compass ist Open Source und kostenlos, allerdings eine Charity-Software.



Susy ist ein Grid-System für Sass und Compass. Im Gegensatz zu anderen populären Frameworks, zwingt Susy dem Webdesigner kein bestehendes System (z.B. 12 Spalten) auf oder reicht zusätzliche Flexibilität nur mit zusätzlichem Code

Susy kann als Grid Calculator verstanden werden. Mit Susy lässt sich ein Gestaltungsraster entwickeln, dass für das jeweilige Projekt maßgeschneidert ist, aber nicht den berüchtigten Overhead (zu viel Code) produziert.

Susy übernimmt die Berechnung des Rasters mit Hilfe komplexer Sass-Funktionen. Auf Basis einiger Variablen (z. B. der Anzahl an Spalten, dem Spaltenabstand, der Spaltenbreite, gewünschten Breakpoints etc.) erzeugt Susy ein individuelles Framework, dass der Compass-Logik sowie der Susy-Namensgebung folgt. Susy unterstützt euch also dabei, auf Basis von Sass und Compass mit individuellen Gestaltungsrastern zu arbeiten.

BOURBON

Bourbon ist eine Zusammenstellung von Sass-Mixins. Die Library kann als Alternative zu Compass verstanden werden. Bourbon ist weniger umfangreich, dafür allerdings auch schlanker als Compass. Für Bourbon existieren ebenfalls einige Erweiterungen.

BOURBON NEAT

Bourbon Neat ist ein Gestaltungsraster für Bourbon. Ähnlich wie bei Susy wird über globale Variablen das Verhalten des Rasters gesteuert.

Bourbon und Bourbon Neat können zusätzlich über die Erweiterungen Bourbon Bitters und Bourbon Refills erweitert werden. Bourbon Bitters stellt zusätzliche Variablen, Styles und Strukturelemente für Bourbon-Projekte bereit. Bourbon Refills ist eine Pattern-Library: Über Bourbon Refills können funktionale Website-Module hinzugeladen werden.

FOUNDATION, BOOTSTRAP & GUMBY

Foundation, Bootstrap und Gumby sind weitere populäre CSS-Frameworks bei denen ihr ebenfalls mit Sass arbeiten könnt. Alle drei Frameworks stellen in unterschiedlichen umfang ein Raster, verschiedene UI-Elemente, Design-Pattern und Komponenten für Responsive Websites bereit. Die Möglichkeit, Sass zu verwenden wurde bei allen drei Tools erst mit späteren Versionen nachträglich integriert.



CODERS.BAY

QUELLE

"WEB DESIGN MIT SASS
EINE EINFÜHRUNG IN MODERNE STYLESHEETS"
VON JONAS HELLWIG - KULTURBANAUSE