CSS Präprozessoren

BACHELORARBEIT 2

StudentIn Barbara Huber, 1010601010
BetreuerIn Hannes Moser

Kuchl, 25.02.2015

Eidesstattliche Erklärung

Hiermit versichere ich, Barbara Huber, geboren am **17.01.1991** in **Kitzbühel**, dass ich die Grundsätze wissenschaftlichen Arbeitens nach bestem Wissen und Gewissen eingehalten habe und die vorliegende Bachelorarbeit von mir selbstständig verfasst wurde. Zur Erstellung wurden von mir keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Ich versichere, dass ich die Bachelorarbeit weder im In- noch Ausland bisher in irgendeiner Form als Prüfungsarbeit vorgelegt habe und dass diese Arbeit mit der den BegutachterInnen vorgelegten Arbeit übereinstimmt.

Kuchl, am **25.02.2015**

Unterschrift

Barbara Huber 1010601010

Kurzfassung

Vor- und Zuname:	Barbara HUBER
Institution:	FH Salzburg
Studiengang:	Bachelor MultiMediaTechnology
Titel der Bachelorarbeit:	Sass vs. Less vs. Stylus
Begutachter:	Hannes Moser

Schlagwörter:

Abstract

Keywords:

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Forschungsfrage . . . 1
1.2 Aufbau . . . 1

2 CSS-Präprozessoren **2**
2.1 Software Komponenten . . . 6
2.2 Less . . . 8
2.3 Sass . . . 9
2.4 Stylus . . . 11
2.5 Erweiterungen von CSS Präprozessoren . . . 12
2.5.1 Variablen . . . 12
2.5.2 Mixin . . . 13
2.5.3 Funktionen . . . 16
2.5.4 Vererbung . . . 16

3 Superset **18**

4 Lexer/Parser **18**

5 Implementierugen von CSS Präprozessoren **22**
5.1 Variablen . . . 22
5.1.1 Less . . . 22
5.1.2 Sass . . . 22
5.1.3 Stylus . . . 23
5.2 Mixins . . . 23
5.2.1 Less . . . 24
5.2.2 Sass . . . 24
5.2.3 Stylus . . . 25
5.3 Funktionen . . . 26
5.3.1 Less . . . 26
5.3.2 Sass . . . 28
5.3.3 Stylus . . . 29

5.4 Vererbung 29
5.4.1 Less 29
5.4.2 Sass 30
5.4.3 Stylus 30

6 Praktische Anwendung und Vergleiche 31

7 Ergebnisse 31

8 Schluss 31

Abbildungsverzeichnis 33

Tabellenverzeichnis 34

Literaturverzeichnis 35

1 Einleitung

1.1 Forschungsfrage

1.2 Aufbau

derzeitiges Literaturverzeichnis:
(Bracey 2014)
(Coyier 2012)
(Croom 2012)
(Firdaus 2014)
(Hixon 2011)
(Page 2013)
(Zing Design 2014)

2 CSS-Präprozessoren

Präprozessoren sind Computerprogramme, welche Daten vorbereiten und zur Weiterverarbeitung an ein anderes Programm weitergeben. In den meisten fällen, wird ein Präprozessor dazu verwendet, Eingabedaten, im Falle dieser Arbeit CSS-Styles, zu konvertieren.(Peter 2012)
Präprozessoren werden verwendet um beispielsweise Variabilität zu schaffen. In dieser Arbeit geht es um die CSS-Präprozessoren Sass, Less und Stylus. Diese Präprozessoren werden verwendet, um das Schreiben des Codes zu erleichtern. Die Präprozessoren erleichtern die Syntax und stellen Funktionen und Variablen zur Verfügung.
Mithilfe der genannten Präprozessoren können Aufgaben automatisiert werden. Es gibt die Möglichkeit Variablen zu erstellen und so die Bearbeitung von Stylings wie z.B. der Farbe, um ein Vielfaches zu erleichtern. Möchte man in einem CSS file die Farbe der Schrift verändern, muss man alle Stellen suchen, an denen diese Farbe zugeordnet wird. Verwendet man beispielsweise mit less eine Variable für die Farbe, so muss nur an einer Stelle, dort wo die Farbe der Variablen zugewiesen wird, die Farbe geändert werden.
Nicht nur Variablen sondern auch Funktionen und Mixins werden von CSS-Präprozessoren bereit gestellt. Im Gegensatz zu einfachem CSS kann man also mit den genannten Präprozessoren Mixins erstellen, die dabei helfen den Code übersichtlicher zu Gestalten. Mit einem Mixin kann z.B. ein Clearfix erstellt werden. Hierfür wird folgender Code, in scss syntax, siehe Kapitel 2.3 was scss ist, erstellt:

Listing 1: erstellen eines Mixins

```
1  @mixin .clearfix {
2   content: ".";
3    display: block;
4    height: 0;
5    clear: both;
6    visibility: hidden;
7  }
8
9  .mainContent {
10   @include: .clearfix;
11 }
```

Die Zeilen 1 bis 7 in Listening 1 erstellen das Mixin *clearfix* und in Zeile 10 wird gezeigt, wie dieses dann aufgerufen werden kann. Mit diesem Mixin kann vermieden werden, dass die in dem Mixin enthaltenen Stylings jedes mal erneut erstellt werden müssen. Stattdessen kann an jeder gewünschten Stelle im Code mit dem Befehl ”include:clearfix"der gesamte Codeabschnitt eingebunden werden.

Ein weiterer Vorteil von CSS-Präprozessoren ist, dass der Code verschachtelt werden kann. Folgendes Listing zeigt, wie sich das verschachteln des Codes auf die Lesbarkeit auswirken

kann. Listing 2 zeigt den Code ohne Verschachtelung und Listing 3 mit Verschachtelung:

Listing 2: Code ohne Verschachtelung

```
1  #content .text .title{
2    font-size: 12px;
3    color: #dedede;
4    font-weight: normal;
5  }
6  #content .text .title h1{
7    font-size: 18px;
8    color: #ffffaa;
9    font-weight: bold;
10 }
11 #content .text .title h3{
12   color: #efefef;
13 }
```

Listing 3: Code mit Verschachtelung

```
1  #content .text{
2    .title{
3      font-size: 12px;
4      color: #dedede;
5      font-weight: normal;
6      h1{
7        font-size: 18px;
8        color: #ffffaa;
9        font-weight: bold;
10     }
11     h3{
12       color: #efefef;
13     }
14   }
15 }
```

Wie in den 2 Listings gut zu sehen, ist der Sinn von CSS-Präprozessoren nicht immer, den code zu verkürzen. Jedoch kann man hier gut erkennen, dass die Schreibweise und Lesbarkeit des Codes vereinfacht wird. In Listing 2 wird Zeile 1 in Zeile 6 und 11 exakt wiederholt. Dies ist in Listing 3 nicht der Fall. Durch die Verschachtelung können beliebig viele Klassen, die sich innerhalb der Klasse text befinden, angesprochen werden, ohne dass man immer alle darüberliegenden Klassen aufrufen muss.

Man muss jedoch darauf achten, dass es nicht immer sinnvoll ist, den Code zu verschachteln. Hat man z.B. einen Button, der auf der ganzen Seite immer grün sein soll, macht es keinen Sinn, dem Button die Styles in einer Verschachtelung zu geben, da man dann immer darauf achten muss, dass der Button auch wirklich in diesem Abschnitt des Quellcodes auftritt. Stattdessen ist es hier sinnvoller, den Button separat zu stylen und bei etwaigen Änderungen, diese dann in der Verschachtelung anzuführen. So kann vermieden werden, dass der Code unübersichtlich wird, und der Button wird nur an bestimmten Stellen geändert.

Neben den genannten Erweiterungen Less, Sass und Stylus zu CSS kann man zusätzlich den Workflow verbessern. Werden die Präprozessoren richtig verwendet kann der Code sauberer und auch kürzer gehalten werden.
Durch den Einsatz der Erweiterungen, kann ein komplexer Code vereinfacht werden, wodurch der Code wartbarer und übersichtlicher wird. Auch kann der Zeitaufwand für die Erstellung der Stylesheets verkürzt werden, da Beispielsweise Code, der häufig verwendet wird, mithilfe eines Mixins nur einmal definiert werden muss.

Der Begriff Workflow beschreibt eine Abfolge von Atkivitäten. In Zusammenhang mit dieser Arbeit, bedeudet Workflow die Ausführung verschiedener Arbeitsschritte in einer definierten Abfolge.
Der erste Arbeitsschritt bei der Erstellung eines Webseitenstylings ist die Erstellung eines Stylesheets. In dieser Arbeit werden less, sass und stylus behandelt. Es wird also entweder eine style.less, eine style.sass bzw. style.scss oder eine style.styl erstellt.
Um diese Styles richtig auf der Webseite ausgeben zu können, folgt der nächste Arbeitsschritt. Die Kompilierung der Dateien in CSS.
Für die Kompilierung gibt es für jeden der genannten CSS-Präprozessoren einige Third-PartyLibraries wie z.B. Koala für Sass und Less oder CodeKit, welche less, stylus und sass kompiliert.
Wie im vorherigen Absatz erwähnt, kann durch die Verwendung der Erweiterungen der Workflow verbessert werden, da der erste Arbeitsschritt schneller und verbessert durchgeführt werden kann.
Die 2 genannten Libraries Koala und CodeKit ermöglichen es, die erstellten Stylesheets in Echtzeit zu kompilieren, so kann der Workflow ein weiteres mal verbessert werden, da die Erstellung des Stylesheets und dessen Kompilierung parallel ablaufen.

Der Code von CSS-Präprozessoren ist immer valides CSS, da die Präprozessoren Erweiterungen von CSS darstellen. Die am Beginn des Kapitels genannten Präprozessoren sind drei der bekanntesten CSS-Präprozessoren, es gibt jedoch noch sehr viele andere, wie zum Beispiel Turbine oder Switch CSS (Jung 2010).

Um zu überprüfen, ob das geschriebene stylesheet valide ist und gute Qualität hat, gibt es Linting. Linting ist die Überprüfung von CSS auf Validität und Codequalität. Es gibt einige OpenSource Tools, die die Überprüfung des Stylesheets einfach und schnell ermöglichen.

Ein gutes OpenSource Tool ist z.B. CSSLint von Nicolas C. Zakas und Nicole Sullivan[1] Neben der Überprüfung der Codequalität werden mit Linting auch die Browserperformance und viele weitere Punkte wie z.B.

- Parsing Fehler
- leere Anweisungen
- Nullwerte benötigen keine Einheiten
- keine IDs in Selektoren
- nicht zu viele Floats verwenden
- nicht zu viele Schriften verwenden

gecheckt.

Wie zuvor beschrieben, bieten CSS-Präprozessoren einige Vorteile, können jedoch auch Nachteile haben, wenn sie falsch verwendet werden. Diese Arbeit vergleicht die 3 bekanntesten CSS-Präprozessoren, aber nicht mit dem Ziel herauszufinden, welche Variante die Beste ist. Es gibt keinen “Besten„ Präprozessor.
Welchen Präprozessor man verwendet bzw. benötigt liegt im Interesse jedes einzelnen. Hier einige Meinungen, von Programmierern, die mit Less, Sass bzw. Scss oder Stylus arbeiten.

> “Less, because it is intuitive and also is the engine inside Twitter Bootstrap. So if you want to edit Bootstrap CSS you edit using Less."(Stephanie Hughes, zitiert nach psdtowp 2014)

> “I use LESS as a CSS preprocessor. ... I like it because it’s closest to vanilla CSS. This way, if you find yourself in a situation where you have to fix something in pure CSS, you haven’t forgotten how to do so, by working in LESS.
>
> In general, I’m a big fan of preprocessors. They allow you to programmatically style your site or app, and they make writing clean, DRY, Object-Oriented CSS much easier."(Jamie Marcus, zitiert nach psdtowp 2014)

> “I have surrendered to SCSS . . . it has the best tooling support (including Chrome) and the most devs using it. I *really* like the others too and have zero qualms about them."(Brian Leroux, zitiert nach psdtowp 2014)

> “Yeah, I use Sass for Css3 and Coffee for JS. Stylus is good if you work with Jade project. SCSS and Less its basicly same, but I prefer Sass, it’s more clean and powerfull."(Gustavo Fernandes, zitiert nach psdtowp 2014)

1. open source Tool zur Überprüfung von CSS: https://github.com/CSSLint/csslint/wiki/About.

> "Yes I do - I'm on the SCSS train for about a year now. It actually made writing stylesheets fun again. I think there is no reason to not use a CSS Preprocessor language since the entry barrier is low and the amount of extra features one can use flexible and optional. I got used to the major features such as nesting, variables, extends or mixins with conditionals etc. and can't image for a second going back to plain old CSS."(Fabricio Marques, zitiert nach psdtowp 2014)

> "I use Scss because most projects use it and Stylus on my own projects. I'm really fond of the stylus syntax, the fact that it makes my stack 100

Aus den Zitaten geht klar hervor, dass jeder seine eigenen Gründe hat, warum er sich für einen CSS-Präprozessor entschieden hat.
Welchen CSS-Präprozessor man verwendet hängt auch sehr davon ab, für welches Projekt man ihn verwendet und wie man sonst arbeitet.

Das gelesene Kapitel beschreibt den Nutzen und die Verwendung von CSS-Präprozessoren und soll erläutern, welche Vorteile und Nachteile so ein Präprozessor bietet.

2.1 Software Komponenten

Software Komponenten (im weiteren Verlauf als Komponenten bezeichnet) sind wiederverwendbare Bausteine einer Applikation, welche aus Software Code bestehen. Komponenten implementieren spezifische Funktionalitäten gemeinsam mit vordefinierten Schnittstellen. Da es sich um wiederverwendbare Teile von Code handelt, sind Komponenten sogenannte Software-Bausteine, die einen bestimmten Bereich eines Geschäftsfeldes kapseln, jedoch keine abgeschlossene Applikation darstellen, und nicht für sich alleine ablaufen können. (Andresen 2003, 1)
In Bezug auf diese Arbeit, sind bspw. Grids Komponenten von CSS-Präprozessoren. Grids sind eine Layout-Komponente von CSS-Präprozessoren. Diese Komponente kann wiederverwendet werden und bildet einen eigenschändigen Bereich des CSS-Präprozessors. Wie beschrieben, ist sie jedoch keine abgeschlossene Applikation und kann nicht alleine Ablaufen.

Im Buch von Andreas Andresen, Komponentenbasierte Softwareentwicklung, (Andresen 2003) werden auf Seite 2 einige Vorteile von Komponenten aufgezählt. Die wichtigsten für diese Arbeit sind folgende:

- überschaubare Größenordnung
- trennen Zuständigkeiten
- einfach einsetzbar und kombinierbar

- einfach wiederverwendbar
- fördern eine schnelle Applikationsentwicklung
- einfach austauschbar

Neben Vorteilen gibt es auch noch wichtige Anforderungen an die Architektur von Komponenten, welche anhand des genannten Beispiels von Grids, näher erklärt werden. Die wichtigesten Anforderungen, welche wie erwähnt noch näher erklärt werden, sind:

- die Ermöglichung einer einfachen Kommunikation von System und Komponente untereinander
- komplexe Zusammenhänge innerhalb von Systemen und zwischen Komponenten und Systemen müssen auf übersichtliche und einfache Weise strukturiert werden können.
- Komponenten und Systeme müssen effizient dimensioniert werden
- bestehende Komponenten müssen einfach integrierbar sein
- eine einfach Wiederverwendung der Komponenten muss ermöglicht werden
- die Zuständigkeit muss klar getrennt sein
- das System muss die Anforderungen in Bezug auf Robustheit, Zuverlässigkeit, Performance, Siherheit und Skalierbarkeit erfüllen

(Andresen 2003, 6)

Ein Grid ist eine Layout Komponente. Das heißt, das Grid kümmert sich um die visuelle Anordnung von Komponenten in einem übergeordneten Container.

2.2 Less

Less ist einer der genannten CSS-Präprozessoren und wird in diesem Kapitel genauer erklärt.
Entwickelt wurde und wird Less von Alexis Sellier. Die Entwicklung begann im Jahr 2010 und geht immer weiter.
Less ist wie auch die Syntax-Variante SCSS von Sass ein Superset und kann somit ohne Probleme in eine CSS datei eingefügt werden. Verwendet man Less, kann man also bei einer CSS Datei einfach die Endung in .less umändern und man hat eine funktionstüchtige Less-Datei in der man die zusätzlichen Eigenschaften von Less einbauen und verwenden kann.
Less verwendet dieselbe Syntax wie css und bietet wie auch Stylus und Sass die Möglichkeit der Verschachtelung und der Verwendung von Features wie Variablen, Mixins, Vererbung und Funktionen.
Im Gegensatz zu Sass, ist Less eine Javascript Library, die wie jede andere Javascript Library im HTML Head eingebunden wird. Zu Beachten ist, dass vor der Einbindung der js Datei die Stylesheets geladen werden.
Um mit Less arbeiten zu können benötigt man also weder eine Commandozeile noch irgendwelche Tools wie Rhino oder Nodejs. Beispiel Einbindung:

Listing 4: Einbindung Less

```
1 //erst das Stylesheet, dann die js datei.
2 <link rel="stylesheet/less" type="text/css" href="styles.less"
     >
3 <script src="less.js" type="text/javascript"></script>
```

Diese Einbindung ist die Einfachste Art Less zu verwenden, jedoch gilt das nur für die Client-seitige Verwendung und mit dieser Einbindung, läuft Less nur mit modernen Browsern.
Um die Less Datein automatisch laden zu können, gibt es ein Client-seitiges Feature, den Watch mode. Um diesen zu aktivieren, kann man entweder “#!watch„ an die URL im headbereich schreiben und die Datei neu laden, oder in der Konsole “less.watch()„ aufrufen.
Für eine Serverseitige Verwendung wird less mit nodejs installiert. Somit kann nach der installation kann die Datei styles.less mit dem Compiler in Node in eine css datei kompiliert werden.(Sellier 2014)

Für die Kompilierung der Less Dateien gibt es einige andere Third Party Tools, wie zum Beispiel SimpLESS oder WinLESS, welche beide Gratis sind.
In einem späteren Kapitel wird noch auf die Verwendung der genannten Erweiterungen mit Less eingegangen.

2.3 Sass

Sass ist eine Erweiterung von CSS3, welche Variablen, Mixins, Selectoren, Funktionen und andere Erweiterungen anbietet. Somit ist Sass ein Präprozessor, welcher schon in einem früheren Kapitel erklärt wurde, von CSS.
Einer der größten Vorteile eines CSS-Präprozessors ist, dass man in der Entwicklung verschiedene Dateien haben kann, ohne Performance einzubüßen.
Um Sass in CSS umzuwandeln, verwendet man entweder die Kommandozeile oder eines von verschiedenen web-frameworks welche die notwendigen Funktionen bereitstellen. Darauf wird später noch genauer eingegangen.
Für Sass gibt es zwei verschieden Syntaxen. Die Ursprüngliche Syntax verwendet die Dateiendung .sass und verwendet Einrückungen statt der geschwungenen Klammern um die Verschachtelung der Selektoren anzuzeigen und Zeilenumbrüche statt eines Semicolons um die Eigenschaften zu trennen.
Die neuere Syntax verwendet die Dateiendung .scss. Hier werden im Gegensatz zur alten Syntax wieder Klammern und Semicolon verwendet. (Yard 2014)
Folgender Codeausschnitt zeigt eine CSS-Datei mit der ursprüngliche Sass Syntax:

Listing 5: Code in ursprünglicher Syntax

```
#content .text
    .title
        font-size: 12px
        color: #dedede
        font-weight: normal
        h1
           font-size: 18px
           color: #ffffaa
           font-weight: bold

    h3
        color: #efefef
```

Nachfolgene Listing zeigt nun einen Codeausschnitt einer CSS-Datei mit der scss Syntax:

Listing 6: Code mit in scss Syntax

```
#content .text{
  .title{
    font-size: 12px;
    color: #dedede;
    font-weight: normal;
    h1{
      font-size: 18px;
      color: #ffffaa;
      font-weight: bold;
```

```
10      }
11      h3{
12        color: #efefef;
13      }
14    }
15  }
```

Wie schon erwähnt bietet Sass die Möglichkeit von Variablen, Funktionen, Mixins und vielem Mehr. Im nachfolgenden Abschnitt wird näher darauf eingegangen.

2.4 Stylus

Stylus ist ein CSS-Präprozessor der mit Nodejs läuft. Die Dateiendung von Stylus ist .styl. Mit dem Befehl “npm install stylus -g„ lässt sich stylus über nodejs installieren und verwenden.
Die Syntax von Stylus ist ähnlich der von Sass. Wie auch bei Sass, können bei Stylus die Klammern und die Semicolons weggelassen werden.
Verwendet man diese Syntax ist Stylus kein Superset von CSS und nicht rückwärtskompatibel.
Die Schreibweise der Syntax ohne Klammern und Semicolons ist optional und es kann auch die normale CSS-Syntax angewandt werden.
Es ist auch möglich, die Syntaxen zu vermischen. Folgender Code zeigt beide Syntaxen in einer Datei:

Listing 7: style.styl

```
1  //Mixin in Stylus
2  border-radius()
3     -webkit-border-radius: arguments;
4     -moz-border-radius: arguments;
5     border-radius: arguments;
6  //Verwendung der normalen CSS-Syntax
7   body a {
8     font: 12px/1.4 "Lucida Grande", Arial, sans-serif;
9     background: black;
10    color: #ccc;
11  }
12 //Verwendung der Stylus Syntax ohne Klammern, aber mit
     Semicolons
13  form input
14    padding: 5px;
15    border: 1px solid;
16    border-radius: 5px;
```

(LearnBoost 2010)

In diesem Codebeispiel wird zuerst ein Mixin erstellt. Hier ist zu sehen, dass dies mit Stylus anders funktioniert als mit Less oder Sass. Darauf wird aber später noch genauer eingegangen.
Die codezeilen 8 bis 12 sind normales CSS und der letzte Abschnitt ist sowohl Stylus-Syntax als auch CSS-Syntax. Es werden keine Klammern gemacht, jedoch Semicolons.
Wie der Code aus Listing 7 zeigt, ist Stylus in der Benützung sehr einfach, da es egal ist, ob nun Klammern und Strichpunkte gemacht werden oder nicht.

2.5 Erweiterungen von CSS Präprozessoren

In diesem Kapitel wird auf Variablen, Mixins, Funktionen und Vererbung von Css-Präprozessoren eingegangen. Es wird erklärt wie diese Komponenten generell verwendet werden. Bei den einzelnen Codebeispielen wird angegeben mit welchem Css-Präprozessor dieser umgesetzt wurde. In einem späteren Kapitel wird die Verwendung jeder Komponente für jeden CSS-Präprozessor im detail beschrieben.

2.5.1 Variablen

Variablen, sind eine Erweiterung der CSS-Präprozessoren gegenüber CSS3 und sind gänzlich gleich wie Variablen in anderen Programmiersprachen. Bei einem großen Projekt mit einigen Grundfarben oder Schriftarten, welche an vielen Stellen im Code verwendet werden, wird somit ermöglicht am Beginn der Datei oder in einer separaten Datei die Variable zu erstellen und ihr einen Wert zuzuweisen. Somit muss in der restlichen Datei nur die Variable aufgerufen werden und bei einer Änderung muss diese nur bei der Zuweisung der Variable geschehen und nicht wie bei CSS an allen Stellen, wo diese verwendet wird.(Yard 2014)
Im Zusammenhang der Erstellung einer Variablen in einer eigenen Datei, ist die Variable **@import**, welche es ermöglicht, in der Entwicklung so viele Datein zu haben, wie man möchte und diese dann in der Produktion zu einer einzigen Datei zusammen zu fügen, sehr interessant(Giraudel 2014).
In der Entwicklung ist es sehr hilfreich die CSS-Datein aufzuteilen, um einen guten Überblick zu schaffen und auch die Größe der Datei beschränken zu können.

> “Multiple files in dev, a single file in prod."(Bruce Lee, zitiert nach Giraudel 2014)

Nachfolgene Listings zeigen eine Datei für die Variablen und eine Datei, in der die Variablen verwendet werden. Die Listings werden in scss syntax geschrieben.

Listing 8: variables.scss

```
1 $primaryFont: normal 13px 'condensed light';
2 $primaryColor: #efefef;
```

Listing 9: style.scss

```
1 @import 'variables.scss';
2
3 #content .text{
4   .title{
5     font: $primaryFont;
6     color: $primaryColor;
7   }
8 }
```

2.5.2 Mixin

Ein Mixin ist eine Classe in CSS, welche viel Ähnlichkeit mit einer Funktion in einer anderen Programmiersprach, z.B. PHP, hat.
In dieser Verwendung ist ein Mixin eine Gruppe von CSS Anweisungen in einer Klasse. Mixins erlauben es, sämtliche Eigenschaften der erstellten Klasse in einer anderen Klasse aufzurufen.
Beispielsweise hat man ein Mixin mit dem Namen RoundBorders, welches die Klasse .RoundBorders erstellt. Diese Klasse .RoundBorders kann man nun ganze einfach in einer anderen Klasse oder auch einer id, z.B. #menu, aufrufen.(Gerchev 2012)
Folgender Code veranschaulicht das aufrufen eines Mixins (alle Folgenden Codebeispiele zum Mixin sind in Less geschrieben):

Listing 10: Mixin

```
1  //Mixin RoundBorders
2  .RoundBorders {
3    border-radius: 5px;
4    -moz-border-radius: 5px;
5    -webkit-border-radius: 5px;
6  }
7
8  #menu {
9    color: gray;
10   .RoundBorders;
11 }
```

In Listing 8 wird in den Zeilen 2 bis 6 das Mixin RoundBorders erstellt. In Zeile 9 wird dieses Mixin aufgerufen. Somit erhält die ID menu die Eigenschaften aus der Klasse RoundBorders.
Die Ausgabe von Listing 8 wird in Listing 9 dargestellt:

Listing 11: Mixin Ausgabe

```
1  //Mixin RoundBorders
2  .RoundBorders {
3    border-radius: 5px;
4    -moz-border-radius: 5px;
5    -webkit-border-radius: 5px;
6  }
7
8  #menu {
9    color: gray;
10   border-radius: 5px;
11   -moz-border-radius: 5px;
12   -webkit-border-radius: 5px;
```

```
13 }
```

Wenn das Mixin in der Ausgabe nicht angezeigt werden soll, kann man dies mit Klammern bewerkstelligen, wie in folgender Listing, in Zeile 2, gezeigt.(the core less team 2014)

Listing 12: Mixin und Ausgabe ohne Mixin

```
1  //Mixin RoundBorders
2  .RoundBorders() {
3    border-radius: 5px;
4    -moz-border-radius: 5px;
5    -webkit-border-radius: 5px;
6  }
7
8  #menu{
9    color: gray;
10   .RoundBorders;
11 }
12
13 //Ausgabe:
14 #menu {
15   color: gray;
16   border-radius: 5px;
17   -moz-border-radius: 5px;
18   -webkit-border-radius: 5px;
19 }
```

Das erstellte Mixin kann somit im gesamten Code verwendet werden. Werden die im Mixin festegelegten Anweisungen in einem Projekt oft benötigt, können damit viele Codezeilen und vorallem Codeduplikationen vermieden werden.

Mixins können auch Argumente oder Selectoren beinhalten.
Soll beispielsweise in einem Mixin für einen abgerundeten Rahmen der Radius variabel bleiben, kann dieser als Argument übergeben werden. Der Code für das Mixin und für die Einbindung in eine Klasse sieht in Less folgendermaßen aus:

Listing 13: Mixin mit Argument

```
1  //Mixin
2  .border-radius(@radius) {
3    -webkit-border-radius: @radius;
4       -moz-border-radius: @radius;
5            border-radius: @radius;
6  }
7
8  .button {
9    .border-radius(6px);
10 }
```

Wie in Listing 11 zu sehen, wird in Zeile 9 der Radius von 6px mitübergeben. (the core less team 2014)
Eine besondere Variable im Zusammenhang mit Mixins ist @arguments. Mit dieser Variable werden alle Argumente, die dem Mixin mitgegeben werden, angewendet. (Gerchev 2012)

Listing 14: Mixin mit @arguments

```
1  //Mixin
2  .BoxShadow(@x: 0, @y: 0, @blur: 1px, @color: #000) {
3    box-shadow: @arguments;
4    -moz-box-shadow: @arguments;
5    -webkit-box-shadow: @arguments;
6  }
7
8  .BoxShadow(2px, 5px);
9
10 //Ausgabe
11 box-shadow: 2px 5px 1px #000;
12 -moz-box-shadow: 2px 5px 1px #000;
13 -webkit-box-shadow: 2px 5px 1px #000;
```

Wie schon erwähnt können Mixins auch Selectoren beinhalten. Das heißt, es kann in einem Mixin auch ein hover Effekt oder ein Aktivstatus angegeben werden.

Listing 15: Mixin mit Selector

```
1  //Mixin
2  .my-hover-mixin() {
3    &:hover {
4      border: 1px solid red;
5    }
6  }
7  button {
8    .my-hover-mixin();
```

```
9  }
10
11 //Ausgabe
12 button:hover {
13   border: 1px solid red;
14 }
```

Mixins können bei jedem der, in dieser Arbeit vorgestellten, CSS-Präprozessoren verwendet werden, die Schreibweise unterscheidet sich jedoch. Darauf wird später noch genauer eingegangen.

2.5.3 Funktionen

Funktionen bei CSS-Präprozessoren sind denen in anderen Programmiersprachen sehr ähnlich. Mit Funktionen, kann man beispielsweise zwei Pixelwerte addieren, subtrahieren, dividieren und multiplizieren. Wie die Implementierung der Funktionen bei den, in dieser Arbeit verwendeten, CSS-Präprozessoren funktioniert, wird in einem späteren Kapitel erklärt.

2.5.4 Vererbung

Verberbung bedeuted, dass die Eigenschaften einer Klasse in einer anderen Klasse vererbt werden können. Im Falle dieser Arbeit hat zum Beispiel die Klasse .message dieselben Eigenschaften wie die Klasse .warning mit ein paar zusätzlichen Eigenschaften. Durch die Möglichkeit der Vererbung bei CSS-Präprozessoren kann nun vermieden werden den Code zu duplizieren. Folgender Code zeigt, wie mit der Variable extend eine Klasse innerhalb einer anderen Klasse aufgerufen und so deren Eigenschaften vererbt werden können:

Listing 16: Vererbung mit extend

```
1 .message {
2     padding: 10px;
3     border: 1px solid #eee;
4   }
5
6   .warning {
7     @extend .message;
8     color: #E2E21E;
9   }
```

Der Code in Listing 16 ist in SCSS syntax geschrieben. In Zeile 7 werden mit @extend die Eigenschaften der Klasse .message vererbt. In allen vorgestellten CSS-Präprozessoren wird zur Vererbung dieselbe Variable verwendet, jedoch unterscheidet sich die Schreibweise in

Stylus und Sass bzw. Scss von der Schreibweise in Less. Darauf wird in einem späteren Kapitel noch eingegangen.

3 Superset

Ein Superset ist eine sogenannte Obermenge. Der Begriff Superset kommt aus der Mengenlehre und bedeutet, dass B ein Superset von A ist, sobald A in B enthalten ist. Bezogen auf diese Arbeit heißt das, dass Beispielsweise Scss, eine Syntax-Variante von Sass, ein Superset von CSS3 ist.
Sobald das CSS3 valide ist, hat man auch ein valides SCSS. Umgekeht gilt das jedoch nicht. Vorteilhaft ist, dass schon vorhandener CSS3-Code einfach weiterverwendet werden kann und mit SCSS erweiterbar ist. Um also eine vorhandene CSS3-Datei in SCSS umzuwandeln, muss nur die Dateiendung von .css auf .scss umgeschrieben werden.

Von den, in dieser Arbeit genannten, Programmiersprachen Less, Sass und Stylus sind nur die Syntax-Variante SCSS, von Sass, und Less jeweils ein Superset von CSS.
Der Begriff Superset darf nicht mit dem Begriff Erweiterung verwechselt werden.
Bei einer Obermenge ist dessen Untermenge immer Valide für die Obermenge, das heißt bei einem Superset wie SCSS ist das CSS, die Untermenge, immer valides SCSS.
Bei den Erweiterungen Sass und Stylus gilt dies jedoch nicht, da die Syntax dieser Präprozessoren nicht dieselbe von CSS3 ist. Auf die Syntax und die Unterschiede wird in einem späteren Kapitel noch näher eingegangen.

4 Lexer/Parser

Ein Parser ist ein Programm zur Zerlegung und Umwandlung einer beliebigen Eingabe, welche zur Weiterverarbeitung in ein brauchbares Fromat umgewandelt wird. Ein Parser erzeugt zusätzliche Strukturbeschreibungen. Parsing bedeuted Syntaxanalyse.
Der Parser verwendet zur Analyse von Text einen lexikalischen Scanner, auch Lexer genannt. Ein Lexer zerlegt die Eingabe in sogenannte Tokens (beispielsweise Wörter oder Eingabesymbole) die der Parser versteht.
Bei einem HTML Code würde der Lexer die Datei ind HTML-Tags und Fließtext zerteilen und so an den Parser weiterleiten. Den Lexer interessiert nur das Aussehen der Syntaxelemente wie z.b. die spietzen Klammern eines Tags. Der Parser verarbeitet dann die syntaktischen Zusammenhänge. Er untersucht also, welche Paar von Tags zusammengehören oder wie diese verschachtelt sind. Die inhaltliche Bedeutung der Tags interessiert den Parser nicht, dafür ist dann die Weiterverarbeitung zuständig.

Im Falle dieser Arbeit würde also der Lexer die Datei welche in Less, Stylus oder Sass bzw. Scss geschrieben wurde, zerlegen. In diesem Fall interessieren den Lexer beispielsweise die Klammern und die Strichpunkte oder wie in Sass die Abstände und Einrückungen.

Auch einige Präprozessoren verwenden einen Parser um die erstellten Dateien in CSS umzuwandeln. Nachfolgende Abschnitte erläutern, wie die beschriebenen Präprozessoren in CSS umwandeln.

Less

Less verwendet beim Parsen keinen Tokenizer, also keinen Lexer. Der Parser von Less durchläuft die Eingabe, den Less-Code, einmal und parsed alles. Das heißt es gibt eine Function in der für alle spezialfälle wieder eigene Funktionen geschrieben werden. Beispielsweise folgender Codeausschnitt aus dem Parser:

Listing 17: Less Parser

```
var Parser = function Parser(context, imports, fileInfo) {
  var parsers,
    parserInput = getParserInput();
    // The Parser
    return {

      parsers: parsers = {

        primary: function () {
          var mixin = this.mixin, root = [], node;

          while (true)
          {
            while (true) {
              node = this.comment();
              if (!node) { break; }
              root.push(node);
            }
            // always process comments before deciding if
               finished
            if (parserInput.finished) {
              break;
            }
            if (parserInput.peek('}')) {
              break;
            }

            node = this.extendRule();
            if (node) {
              root = root.concat(node);
              continue;
            }

            node = mixin.definition() || this.rule() || this.
               ruleset() ||
            mixin.call() || this.rulesetCall() || this.
               directive();
```

```
35              if (node) {
36                root.push(node);
37              } else {
38              if (!(parserInput.$re(/^[\s\n]+/) || parserInput.
                   $re(/^;+/))) {
39                break;
40              }
41            }
42          }
43
44          return root;
45        },
46
47        entities: {
48          variable: function () {
49            var name, index = parserInput.i;
50
51            if (parserInput.currentChar() === '@' && (name =
                 parserInput.$re(/^@@?[\w-]+/))) {
52              return new(tree.Variable)(name, index, fileInfo);
53            }
54          },
55
56          // A variable entity useing the protective {} e.g. @{var
               }
57          variableCurly: function () {
58            var curly, index = parserInput.i;
59
60            if (parserInput.currentChar() === '@' && (curly =
                 parserInput.$re(/^@\{([\w-]+)\}/))) {
61              return new(tree.Variable)("@" + curly[1], index,
                   fileInfo);
62            }
63          },
64
65        },
66
67        variable: function () {
68          var name;
69
70          if (parserInput.currentChar() === '@' && (name =
               parserInput.$re(/^(@[\w-]+)\s*:/))) { return name
               [1]; }
```

```
71        },
72
73      }
74    };
75  };
76  Parser.serializeVars = function(vars) {
77    var s = '';
78
79    for (var name in vars) {
80      if (Object.hasOwnProperty.call(vars, name)) {
81         var value = vars[name];
82         s = ((name[0] === '@') ? '' : '@') + name + ': ' +
              value +
83        ((String(value).slice(-1) === ';') ? '' : ';');
84      }
85    }
86
87    return s;
88  };
89
90  module.exports = Parser;
```

5 Implementierugen von CSS Präprozessoren

5.1 Variablen

Variablen sind wie schon in Kapitel 2.5.1 beschrieben, eine Erweiterung der CSS-Präprozessoren. Mithilfe der Variablen können Werte am Beginn des Stylesheets definiert werden und an jeder Stelle im Code wiederverwendet werden. In diesem Kapitel wird die Verwendung von Variablen mit Less, Stylus und Sass erläutert. Im großen und ganzen ist die Verwendung sehr ähnlich und es werden hier nur die Unterschiede näher betrachtet.

5.1.1 Less

In Less wird die Variable mit dem Zeichen "@„ gekennzeichnet. Folgende Listing zeigt die Initialisierung der Variable und die anschließende Verwendung im Code:

Listing 18: Verwendung Variable in less

```
1 @color: #4D926F;
2
3 #header {
4   color: @color;
5 }
6 h2 {
7   color: @color;
8 }
```

Wie in Zeile 4 zu sehen, muss die Variable wieder mit dem "@„ Zeichen aufgerufen werden. Ohne dem "@„ Zeichen wird keine Farbe mitgegeben.

5.1.2 Sass

Wie auch in Less gibt es für Sass eine eigene Schreibweise für Variablen. Diese Schreibweise gilt für beide Syntaxvarianten von sass.

Listing 19: Verwendung Variable in sass

```
1 $color: #4D926F;
2
3 #header {
4   color: $color;
5 }
6 h2 {
7   color: $color;
8 }
```

Der Code in Listing 18 zeigt die Verwendung einer Variable in Sass. Zeile 1 zeigt, dass die Variable mit “$„ dargestellt werden muss. Auch bei der Verwendung muss, wie bei less das Zeichen “@„, das Zeichen “$„ vorangestellt werden, damit die Variable richtig erkannt wird.

5.1.3 Stylus

Im Gegensatz zu Less und Sass muss bei Stylus kein Sonderzeichen vor die Variable gesetzt werden, jedoch kann die Schreibweise von Sass verwendet und vor die Variable das Zeichen “$„ geschrieben werden. Nachfolgender Code zeigt beide Schreibweisen von Variablen in stylus:

Listing 20: Verwendung Variable in stylus

```
1  \\Initialisierung Variable ohne Sonderzeichen
2  font-size = 14px
3  font = font-size "Lucida Grande", Arial
4  \\Verwendung Variable
5  body
6    font font, sans-serif
7
8  \\Initialisierung Variable mit Sonderzeichen
9  $font-size = 14px
10 \\Verwendung Variable
11 body {
12   font: $font-size sans-serif;
13 }
```

Wie die Listing zeigt, wird in Zeile 2 und 3 die Variable ‚font-size’ bzw. ‚font’ ohne voransetzen eines Sonderzeichens initialisiert. Somit wird auch beim Aufruf der Variable kein Sonderzeichen vorangestellt. Wie in Zeile 3 zu sehen, wird die Variable ‚font-size’ direkt in der Variable ‚font’ aufgerufen. Dadurch muss in Zeile 6 bei der Verwendung nur die Variable ‚font’ aufgerufen werden.
In Zeile 9 wird die Variable wie bei Sass mit dem Zeichen “$„ initialisiert. Dadurch ist auch die Verwendung ident mit der von Sass.

Die letzten 3 Unterkapitel zeigen, dass eine Variable in den 3 beschriebenen CSS-Präprozessoren bis auf die verwendeten Sonderzeichen gleich erstellt und verwendet werden. Im nächsten Kapitel wird die Verwendung von Mixins näher betrachtet.

5.2 Mixins

Mixins sind eigene Klassen, welche am Beginn des Stylesheets erstellt werden und fortlaufend im ganzen Stylesheet wiederverwendet werden können.

In Kapitel 2.5.2 wurde beschrieben, was Mixins sind und wie sie im generellen funktionieren. Im folgenden, wird für jeden Präprozessor im detail erklärt, wie ein Mixin formuliert und angewendet wird.

5.2.1 Less

Mit Less wird eine mixin Klasse erstellt wie jede andere Klasse auch. Folgender Code erstellt ein Mixin und verwendet dieses anschließend:

Listing 21: Verwendung Mixin in Less

```
1  .rounded-corners (@radius: 5px) {
2    border-radius: @radius;
3    -webkit-border-radius: @radius;
4    -moz-border-radius: @radius;
5  }
6
7  #header {
8    .rounded-corners;
9  }
10 #footer {
11   .rounded-corners(10px);
12 }
```

Die Zeilen 1 bis 5 aus Listing 20 erstellen die Mixin-Klasse “rounded-corners„, welche auch die Mitgabe eines Parameter, in diesem Fall den Radius, ermögliicht.
In Zeile 8 und 11 wird die Klasse dann aufgerufen und verwendet. Wie erwähnt, ermöglicht das Mixin auch parameter. Als Defaultwert wird dem Mixin in Zeile 1 ein radius von 5px zugewiesen. In Zeile 11 wird der Defaultwert überschrieben und auf 10px geändert.

5.2.2 Sass

In Sass wird ein Mixin nicht wie in Less definiert. Im Gegensatz zu less, wird bei Sass nicht einfach eine klasse definiert. Listing 21 zeigt, wie mit Sass ein Mixin erstellt wird:

Listing 22: Verwendung Mixin in Sass

```
1  @mixin large-text {
2    font: {
3      family: Arial;
4      size: 20px;
5      weight: bold;
6    }
7    color: #ff0000;
8  }
```

```
9
10 .page-title {
11   @include large-text;
12   padding: 4px;
13   margin-top: 10px;
14 }
```

Der Code zeigt in Zeile 1, dass in Sass für die Erstellung eines Mixins "@mixin„ vor den Namen des Mixins gesetzt werden muss. Trotz der differenten Schreibweise, wird ein Mixin auch in Sass wie eine Klasse behandelt.
Für die Verwendung des Mixins wird in Zeile 11 mit "@include„ und dem Mixin-Namen ‚large-text' aufgerufen.

5.2.3 Stylus

In Stylus ist der Aufbau eines Mixins sehr ähnlich dem mit Less.

Listing 23: Verwendung Mixin in Stylus

```
1 rounded-corners (n)
2   border-radius  n
3   -webkit-border-radius  n
4   -moz-border-radius  n
5
6 #footer
7   rounded-corners(10px)
```

In den Zeilen 1 bis 4 wird das Mixin erstellt. In Zeile 1 wird die Variable n als Parameter mitgegeben, welcher beim Aufruf des Mixins in Zeile 11 ausgefüllt wird. Somit beträgt der Radius 10px.
In Stylus ist die Syntax sehr variabel und es kann von dem Entwickler, der Entwicklerin selbst entschieden werden, ob Klammern, Strichpunkte und Doppelpunkte geschrieben werden oder nicht. Verwendet man die Syntax mit den Zeichen, sieht der Code aus Listing 22 wie folgt aus:

Listing 24: Verwendung Mixin in Stylus

```
1 .rounded-corners (n){
2   border-radius: n;
3   -webkit-border-radius: n;
4   -moz-border-radius: n;
5 }
6
7 #footer{
8   .rounded-corners(10px);
9 }
```

Listing 23 zeigt, dass die Erstellung des Mixins mit dieser Syntax gänzlich der Erstellung mit Less gleicht.

5.3 Funktionen

Dieses Kapitel behandelt die Verwendung von Funktionen und Operatoren mit den CSS-Präprozessoren. Funktionen und Optionen helfen die Strutkur und Lesbarkeit des Stylesheets zu optimieren und es können komplexe Strukturen innerhalb der CSS Datei erstellt werden.
Bei Sass und Stylus können Funktionen eigens definiert werden und im ganzen Stylesheet angewendet werden.
In Less gibt es sehr viele vordefinierte Funktionen, die Beispielsweise Farbmischungen ermöglichen.

5.3.1 Less

Wie erwähnt, gibt es in Less viele verschiedene vordefinierte Funktionen zum Erstellen von komplexen Strukturen. Einige Beispiele, auf die im Anschluss noch genauer eingegangen wird, sind:

- floor
- argb
- saturation
- fadein
- mix
- average

floor
Floor ist eine mathematische Funktion, die zur Berechnung von Integern dient. Mit floor kann ein Integer abgerundet werden. Die Funktion ist wie bei PHP, nur dass hier nicht mitgegeben werden kann, auf wie viele Stellen abgerundet wird. Bei der Funktion in Less wird automatisch auf die nächst niedrigere ganze Zahl abgerundet.

argb
Argb ist wie ‚rgb' eine Farbfunktion und berechnet eine Hexdezimal Version der angegebenen Farbe im #AARRGGBB Format. Anders als bei rgba wird hierbei der Alphawert zu Beginn verwendet. Beispielsweise:

Listing 25: ARGB in Less

```
1  rgb(90, 23, 148);
2  //Ausgabe:
3  #5a1794
4
5  rgba(90, 23, 148, 0.5)
6  //Ausgabe:
7  #5a179480
8
9  argb(rgba(90, 23, 148, 0.5));
10 //Ausgabe
11 #805a1794
```

Wie die Listing zeigt, wird der Alphawert, welcher in der rgba-Funktion als letzter Parameter mitgegeben wird, bei der Berechnung des Hexadezimalwertes am Beginn verwendet.

saturation
Saturation ist wie ‚argb' eine Farbfunktion, die jedoch nicht wie argb die übergebenen Parameter in einen Hexadezimalwert umrechnet sondern den Sättigungskanal aus dem Farbobjekt extrahiert. Die Ausgabe dieser Funktion ist ein Prozentwert zwischen 0 und 100 und wird aus einem hsl-Wert ermittelt.

Listing 26: Saturation in Less

```
1 saturation(hsl(90, 100%, 50%))
2 //Ausgabe
3 100%
```

fadein
Die Farboperation ‚fadein' erhöht die Deckkraft der übergebenen Farbe. Als mitgabeparameter stellt die Funktion einen hsla-wert und eine Prozentzahl zwischen 0 und 100 bereit. Nachfolgender Code zeigt, wie der übergebene Farbwert mit der übergebenen Prozentzahl berechnet wird.

Listing 27: Fadein in Less

```
1 fadein(hsla(90, 90%, 50%, 0.5), 10%)
2 //Ausgabe
3 rgba(128, 242, 13, 0.6) (hsla(90, 90%, 50%, 0.6))
```

Die gegenteilige Funktion zu ‚fadein' ist ‚fadeout', welche die Deckkraft nicht erhöht sondern vermindert.

mix
Mix ist eine Funktion, die, wie der Name vermuten lässt, zwei Farben miteinander vermischt. Hierbei wird auch die Transparenz beachtet. Mitgegeben werden der Funktion 2 Farbwerte und die gewünschte Transparenz.

Listing 28: Mix in Less

```
1 mix(#ff0000, #0000ff, 50)
2 mix(rgba(100,0,0,1.0), rgba(0,100,0,0.5), 50)
3 //Ausgabe
4 #800080
5 rgba(75, 25, 0, 0.75);
```

average
Average ist ebenso, wie die gleichnamige Funktion in PHP, eine Funktion, welche den Durchschnitt berechnet. In Less berechnet die Funktion aber nicht das Mittel von Integerwerten sondern den Durchschnitt von Farbwerten. Die mitgegebenen Parameter sind zwei Farbobjekte, aus denen ein drittes Farbobjekt, der Medianwert ermittelt wird.

Listing 29: Average in Less

```
1 average(#ff6600, #000000);
2 //Ausgabe
3 #ff6600 #000000 #803300
```

(Sellier 2014)

Neben den vorgestellten Funktionen, die von Less zur Verfügung gestellt werden, kann man auch einfache mathematische Funktionen erstellen wie z.B.

Listing 30: Funktionen in Less

```
1 @base: 5%;
2 @filler: (@base * 2);
3
4 height: (100% / 2 + @filler);
```

5.3.2 Sass

In Sass wird eine Funktion auf folgende Weise erstellt:

Listing 31: Verwendung Funktion in Sass

```
1 $grid-width: 40px;
2 $gutter-width: 10px;
3
4 @function grid-width($n) {
5   @return $n * $grid-width + ($n - 1) * $gutter-width;
6 }
7
8 #sidebar { width: grid-width(5); }
```

Wie bei der Verwendung von Variablen beschrieben, werden in Zeile 1 und 2 die Variablen ‚grid-width' und ‚gutter-width' erstellt. Die Zeilen 4 bis 6 erstellen die Funktion, der eine Variable mitgegeben werden kann. In der Funktion wird dann mithilfe des mitgegebenen Parameters und der zuvor definierten Werte ‚grid-width' und ‚gutter-width' die Breite berechnet, welche in Zeile 8 aufgerufen wird.
Die Funktion in Listing 30 zeigt, wie Funktionen in Sass im Generellen erstellt und verwendet werden.

5.3.3 Stylus

Wie in Sass, sieht in Stylus die Erstellung einer Funktion sehr ähnlich aus. Da jedoch bei Stylus auf Klammern, Strichpunkte und Doppelpunkte verzichtet werden kann, wird hierbei eine Funktion wie in Listing 31 gezeigt, formuliert:

Listing 32: Verwendung Funktion in Stylus

```
1 grid-width: 40px;
2 gutter-width: 10px;
3
4 function grid-width(n)
5   return n * grid-width + (n - 1) * gutter-width
6
7 #sidebar
8    width: grid-width(5)
```

Wie schon bei der Erstellung einer Variablen mit Stylus erklärt wurde, wird auch hier kein Sonderzeichen benötigt, um eine Variable zu definieren. Ebenso wird für die Funktion nichts dergleichen benötigt.

5.4 Vererbung

Kapitel 2.5.4 beschreibt, was Vererbung bei CSS-Präprozessoren bedeutet und wie diese im generellen funktioniert. Dieses Kapitel beschreibt für jeden behandelten CSS-Präprozessor, wie Vererbung zustande kommt und verwendet wird.

5.4.1 Less

In Less funktioniert die Vererbung einfach durch Verschachtelung.

Listing 33: Vererbung in Less

```
1 #header {
2   h1 {
```

```
3       font-size: 26px;
4       font-weight: bold;
5     }
6     p { font-size: 12px;
7       a { text-decoration: none;
8         &:hover { border-width: 1px }
9       }
10    }
11  }
```

Auch mithilfe von Mixiins kann in Less vererbt werden. Ein Mixin ist wie erwähnt eine eigene Klasse und diese kann in jeder anderen Klasse durch das Aufrufen vererbt werden. Wie ein Mixin erstellt und aufgerufen wird, wurde bereits in Kapitel 5.2.1 erläutert.

5.4.2 Sass

Sass behandelt Vererbung anders als Less. Sass verwendet für die Vererbung von Klassen die Anweisung ‚@extend', welche an jeder Stelle im Code aufgerufen werden kann. Listing 33 zeigt ein kurzes Beispiel, wie die Vererbung in Sass funktioniert.

Listing 34: Vererbung in Sass

```
1 .error {
2   border: 1px #f00;
3   background-color: #fdd;
4 }
5 .seriousError {
6   @extend .error;
7   border-width: 3px;
8 }
```

Die Listing zeigt, wie in der Klasse ‚.seriousError' die Klasse '.error' vererbt wird.

5.4.3 Stylus

Stylus vererbt Klassen auf genau dieselbe Weise wie Sass.

6 Praktische Anwendung und Vergleiche

Ich denke hier werde ich versuchen einen Parser zu schreiben. Zusätzlich möchte ich einen Generator machen, bei dem ich aussuchen kann, welchen Präprozessor ich in CSS parsen möchte. (oder so ähnlich)

7 Ergebnisse

hier möchte ich die Ergebnisse aus dem praktischen Teil zusammenfassen: Installation(dauer, Schwierigkeiten ...), Codequalität, Codekomplexität, Browserkompatibilität (vielleicht fällt mir bei der Recherche noch mehr ein, was ich vorher vergleiche kann und hier präsentieren)

8 Schluss

ist eh klar, was hier kommt ;)

Abkürzungsverzeichnis

Abb.	Abbildung
z.B.	zum Beispiel
ca.	cirka
bzw.	Beziehungsweise

Abbildungsverzeichnis

Listings

1 erstellen eines Mixins . . . 2
2 Code ohne Verschachtelung . . . 3
3 Code mit Verschachtelung . . . 3
4 Einbindung Less . . . 8
5 Code in ursprünglicher Syntax . . . 9
6 Code mit in scss Syntax . . . 9
7 style.styl . . . 11
8 variables.scss . . . 12
9 style.scss . . . 12
10 Mixin . . . 13
11 Mixin Ausgabe . . . 13
12 Mixin und Ausgabe ohne Mixin . . . 14
13 Mixin mit Argument . . . 15
14 Mixin mit @arguments . . . 15
15 Mixin mit Selector . . . 15
16 Vererbung mit extend . . . 16
17 Less Parser . . . 19
18 Verwendung Variable in less . . . 22
19 Verwendung Variable in sass . . . 22
20 Verwendung Variable in stylus . . . 23
21 Verwendung Mixin in Less . . . 24
22 Verwendung Mixin in Sass . . . 24
23 Verwendung Mixin in Stylus . . . 25
24 Verwendung Mixin in Stylus . . . 25
25 ARGB in Less . . . 27
26 Saturation in Less . . . 27
27 Fadein in Less . . . 27
28 Mix in Less . . . 28

29 Average in Less 28
30 Funktionen in Less 28
31 Verwendung Funktion in Sass 28
32 Verwendung Funktion in Stylus 29
33 Vererbung in Less 29
34 Vererbung in Sass 30

Tabellenverzeichnis

Literaturverzeichnis

Andresen, Andreas. 2003. *Komponentenbasierte Softwareentwicklung: mit MDA, UML und XML.* München und Wien: Carl Hanser. ISBN: 3-446-22282-0.

Bracey, Kezz. 2014. *Why I Choose Stylus.* Besucht am 26. Oktober 2014. http://webdesign.tutsplus.com/articles/why-i-choose-stylus-and-you-should-too--webdesign-18412.

Coyier, Chris. 2012. *Sass vs. Less.* Besucht am 26. Oktober 2014. http://css-tricks.com/sass-vs-less/.

Croom, Johnathan. 2012. *Sass vs. Less vs. Stylus: Preprocessor Shootout.* Besucht am 26. Oktober 2014. http://code.tutsplus.com/tutorials/sass-vs-less-vs-stylus-preprocessor-shootout--net-24320.

Firdaus, Thoriq. 2014. *CSS Preprocessors Compared: Sass vs. LESS.* Besucht am 26. Oktober. http://www.hongkiat.com/blog/sass-vs-less/.

Gerchev, Ivaylo. 2012. *A Comprehensive Introduction to Less: Mixins.* Besucht am 27. November 2014. http://www.sitepoint.com/a-comprehensive-introduction-to-less-mixins/.

Giraudel, Hugo. 2014. *What's the Difference Between Sass and SCSS?* Besucht am 2. November 2014. http://www.sitepoint.com/whats-difference-sass-scss/.

Hixon, Jeremy. 2011. *An Introduction to Less, and Comparison to Sass.* Besucht am 26. Oktober 2014. http://www.smashingmagazine.com/2011/09/09/an-introduction-to-less-and-comparison-to-sass/.

Jung, Jean-Baptiste. 2010. *8 CSS preprocessors to speed up development time.* Besucht am 21. Dezember 2014. http://www.catswhocode.com/blog/8-css-preprocessors-to-speed-up-development-time#disqus_thread.

LearnBoost. 2010. *Stylus: Expressive, dynamic, robust CSS.* Besucht am 21. Dezember 2014. http://learnboost.github.io/stylus/.

Page, Luke. 2013. *Less vs. Sass vs. Stylus.* Besucht am 26. Oktober 2014. http://www.scottlogic.com/blog/2013/03/08/less-vs-sass-vs-stylus.html.

Peter, Christian. 2012. *C-Präprozessoren.* http://wr.informatik.uni-hamburg.de/_media/teaching/wintersemester_2012_2013/epc-1213-peter-praeprozessor-praesentation.pdf.

psdtowp. 2014. *CSS preprocessors.* Besucht am 21. Dezember 2014. https://psdtowp.net/css-preprocessors.html.

Sellier, Alexis. 2014. *Less die dynamische stylesheet sprache.* Besucht am 21. Dezember. http://www.lesscss.de/.

the core less team. 2014. *Language Features: Features of the Less language.* Besucht am 27. November. http://lesscss.org/features/.

Yard. 2014. *Sass: Documentation.* Besucht am 24. November 2014. http://sass-lang.com/documentation/file.SASS_REFERENCE.html#syntax.

Zing Design, Sam. 2014. *Less vs. Sass.* Besucht am 26. Oktober 2014. http://www.zingdesign.com/less-vs-sass-its-time-to-switch-to-sass/.

Anhang