LESS & SASS

#SAE-arbeit/unterrichte/frontend_advanced

Vorteile von CSS Preprozessoren

- CSS Code ist bei großen Projekten ewig lang und unübersichtlich
- Preprozessoren helfen den CSS Code eleganter zu gestallten und zu organisieren
- Durch Variablen sind Änderungen sehr schnell und einfach erledigt

Beispiel Gründe für Preprozessoren:

- die Definition der Schrift erspart einiges an Code Zeilen
- CSS3, eine Angabe mit Präfixen
  - box-sizing muss mit webkit & moz für Firefox definiert werden.
- durch Mixins werden Angaben einmal definieren und können durch einen einzelnen Befehl aufgerufen und wiederverwendet werden
- Farbverläufe können sehr lange werden, speziell für ältere Browser müssen mehrere Zeilen Code geschrieben werden
- Stylesheets können Modular aufgebaut werden, am Ende kann alles zusammen gefasst werden und es wird eine Datei ausgegeben
- schneller erstellen, leichter warten, und besser verändern
- Frameworks verwenden auch Preprozessoren
  - Bootstrap ➜ source code
  - Foundation 6 ➜ source code
- Mehr Zeit für die Spaßigen Sachen an CSS

Grundprinzip von Preprozessoren

- Ohne Preprozessoren: Die Seite wird angeschaut, das **CSS File** wird geändert und aktualisiert.
- Sobald mit Preprozessoren gearbeitet wird, wird nicht mehr das CSS File geändert, sondern die LESS oder SASS Dateien. Schaut ähnlich aus wie CSS, beinhaltet aber Eigenschaften die es in CSS nicht gibt.
- Mit einem Compiler wird aus einer LESS / SASS Datei eine CSS Datei.

Compiler

[sass](https://www.sassmeister.com/)

```
$color: red;

@mixin transition {
  -webkit-transition: background-color 2s ease-out;
  -moz-transition: background-color 2s ease-out;
  -o-transition: background-color 2s ease-out;
  transition: background-color 2s ease-out;
}

p, a {
  @include transition;
}

p:hover {
  background-color: $color;
}
```

[less](http://lesscssismore.com/rek4za/)

```
@color: red;
@padding: 10px;
@border: 10px solid blue;
@font: Helvetica, Arial, sans-serif;
@important-color: orange;
@second-color: blue;

body {
  color: @color;
  font-family: @font;
  padding: @padding;
  border: @border;
}

footer {
```

```
  color : @second-color;
  background-color: @color;
}
```

[LESS](http://lesscss.org/) & [SASS](https://sass-lang.com/)

Mit beiden können schneller CSS Dateien erstellt werden.

Unterschied

Kleinigkeiten auf der Syntaktischen Ebene, von der Grundfunktionalität sind beide gleich, die Umwandlung funktioniert bei beiden gleich.

SASS

- Variablen werden mit einem **$** gesetzt
- es gibt ein @mixing
- es wird mit @include gearbeitet

Wieso SASS?

- Foundation Framework setzt auf SASS
- compass setzt auf sass auf und beinhaltet weiter nützliche Funktionen, es gibt nichts vergleichbares für LESS

LESS

- Variablen werden mit einem **@** gesetzt
- **mixing** wird bei LESS nicht verwendet, es sieht eher aus wie eine Klasse
- LESS braucht kein include es wird wie eine CSS Klasse geschrieben

Wieso LESS?

- Bootstrap baut auf LESS auf
- Kann ohne Preprozessor eingebunden werden

Was soll ich verwenden?

Es kommt auf das Umfeld an. Wenn in der **Arbeitsumgebung** verschiedene Preprozessoren eingesetzt werden kommt es zu Fehlern. Bei Team arbeiten von Beginn an

einigen um Fehler vorzubeugen.

LESS Getting Started

LESS kann auch ohne Preprozessor direkt im Browser angezeigt werden. In diesem Fall muss eine JavaScript Datei eingebunden werden, welche den LESS Code in CSS Code umwandelt. Die JS Datei wird direkt nach dem Stylesheet eingebunden und bei der Link Relation muss der Parameter durch **/less** ergänzt werden. Zu Beachten ist bei dieser Methode, dass mit dem Einbinden einer zusätzlichen Datei ein weiterer Request beim laden der Webseite ausgeführt wird und somit die Performance beeinträchtigt.

```
<link rel="stylesheet/less" type="text/css" href="styles.less" />
<script src="//cdnjs.cloudflare.com/ajax/libs/less.js/3.9.0/less.min.js" ></
script>
```

CLI

Eine weitaus elegantere Methode ist die LESS Dateien zuerst zu kompilieren und anschließend eine CSS Datei dem Browser zu übergeben. Die gängigste Variante ist die Compilierung mit der CLI (Command Line Interface). Um die Compilierung durchzuführen muss node installiert sein.

install node.js und darin beinhalteten Node Package Manager (kurz: npm)

LESS global installieren um es in jedem Projekt verwenden zu können.

```
npm install -g less
```

für die CLI Kompilierung wird der command **lessc (less compile)** verwendet

```
lessc input.less output.css
```

Es muss ein input file (*.less) existieren, eine CSS Datei wird automatisch von dem CLI Tool erstellt.

[Übung 1](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_1_cli.zip)

Zusammenfassung

Für die Installation von LESS über die CLI wird node.js benötigt. Um LESS global erreichbar zu machen wird mit dem **-g** tag installiert. Mit dem Befehl **lessc input.less output.css** wird das LESS File in CSS umgewandelt.

GUIs

Natürlich ist es auch möglich LESS Dateien mit GUIs (graphical user interface) um zu wandeln. Vorteil von GUIs viele Compiler minify'n / uglyfy'n die finalen CSS Dateien um die Performance zu verbessern. Mit der CLI können die fertigen Dateien aber auch uglyfied werden.

[GUIs for less](http://lesscss.org/tools/#guis-for-less)

DRY

(don't repeat yourself)

DRY gilt nicht nur für CSS. Egal ob man PHP, JS, C++, etc. programmiert. Sobald ein Ablauf mehr als einmal vorkommt sollte man mit Variabeln / Funktionen arbeiten.

```
body {
color: #fff;
font-family: Helvetice, Arial, sans-serif;
background-color: #7d9153;
}

footer {
color: #7d9153;
background-color: #fff;
}

nav {
background-color: #d37130;
border-left: 30px solid #fff;
}
```

```
nav a:link, nav a:visited{
color: #fff;
}
```

Farben sind in CSS ein perfektes Beispiel. Sollte man sich entscheiden eine andere Farbe zu benutzen muss man diese an sämtlichen Stellen ändern. Wenn man mit LESS arbeite muss man nur die Variable zu Beginn ändern und sie wird auch an allen anderen Stellen geändert.

Variablen

Variablen können in den unterschiedlichsten Varianten angegeben werden.

```
@color: #000;
@width: 10px; *// Werte*
@border: 10px solid purple; *// Complexere Ausdrücke*
@bool: true; *// Wird interessant wenn mit Bedingungen gearbeitet wird*
@font: Helvetic, Arial, sans-serif; *// Listen*
```

Variablen werden gleich zugewiesen wie normale CSS Werte.

```
body {
  font-family: @font
}
```

[Übung 2](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_2_variablen.zip)

Nesting

Eine übliche Schreibweise für CSS Klassen:

```
nav {
background-color: #d37130;
border-left: 30px solid #fff;
}
```

```
nav ul {
    list-style-type: none;
}

nav a:link, nav a:visited{
color: @color;
}

nav a:hover,
nav a:active,
nav a :focus{
    background-color: orange;
}
```

Jedesmal wird "nav" wiederholt. Es können noch weitaus komplexere Selektoren entstehen und man verliert schnell den Überblick und es entstehen Fehlerquellen.

Ein Beispiel für eine Verschachtelung des nav selectors

```
nav {
background-color: #d37130;
border-left: 30px solid #fff;

    ul {
        list-style-type: none;
    }

    a:link, nav a:visited{
        color: @color;
    }

    a:hover,
    a:active,
    a:focus{
        background-color: orange;
    }
```

```
}
```

Jetzt kann der obersten Selector (root selector) mit nur einem Wort geändert werden und man muss nicht jedes mal nav umschreiben. **DRY** Der CSS code verändert sich nicht, aber man muss deutlich weniger Selektoren schreiben.

[Übung 3](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_3_nesting.zip)

More Nesting

```
nav {
    background-color: #d37130;
    border-left: 30px solid #fff;

    ul {
        list-style-type: none;
    }

    a {
      &:link,
      &:visited
      {
        color: @color;
      }

      &:hover,
      &:active,
      &:focus
      {
         background-color: orange;
      }
    }
}
```

Warum „&“ ?

Beim kompilieren wird automatisch ein Leerzeichen für den Nachfolge Selektor erzeugt damit der Code in CSS auch funktioniert. Mit dem & Zeichen wird nach dem a Selektor kein Leerzeichen eingesetzt.

Das & Zeichen kann noch für weitere Selektoren verwendet werden. Wenn dem Elternelement, z.B. dem body die Klasse „.post" durch js hinzugefügt wird, wird die Farbe der nav rot.

```
nav {
  ...

  .post & {
    color: red
  }
}
```

Zusammenfassung

- Nesting wird genutzt um Selektoren nicht ständig wiederholen zu müssen.
- Das & Zeichen gilt als Platzhalter für die oben genannten Selektoren

Mixins

Wie wir bereits wissen werden für einfache Wiederholungen Variablen verwendet, welche an unterschiedlichen Stellen eingesetzt werden können. Mixins erlauben Mehrzeiligen Code wieder zu verwenden. In LESS schauen Mixins den normalen CSS Klassendefinitionen gleich.

```
@font: Helvetica;

.fontmedium {
  font-size: 14px; *// ältere Browser welche rem nicht verstehen, verwenden die 14px*
  font-size: 0.84rem; *//rem ist eine relativ neue Schreibweise welche keine Probleme mit der Verärbung hat, es wird immer das Root-Element genommen*
}
.transition {
  -webkit-transition: background-color 2s ease-out;
```

```
  -moz-transition: background-color 2s ease-out;
  -o-transition: background-color 2s ease-out;
  transition: background-color 2s ease-out;
}

body {
  color: red;
  font-family: @font;
  .fontmedium;
}

nav {
  a {
    .transition;
  }
}
```

Es ist das gleiche Prinzip wie bei Variablen **don't repeat yourslef** und es hilft euch bei der Codewartung. Wenn etwas geändert werden soll, muss dies nur an einer Stelle gemacht werden.

[Übung 5](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_5_mixins.zip)

Parameter

In LESS kann auch mit Parametern gearbeitet werden. Ähnlich wie in javascript

```
.transition_parameter(@duration: 1s, @property: all, @function: ease) {
  -webkit-transition: @duration @property @function;
  -moz-transition: @duration @property @function;
  -o-transition: @duration @property @function;
  transition: @duration @property @function;
}

body {
  .transition_parameter(2s, background-color, linear);
}
```

Mixins ohne Parameter werden in die CSS Datei übernommen weil sie sich von CSS

Klassen nicht unterscheiden. Mixins mit Parameter werden jedoch nicht übernommen. Wenn man nicht will das die Mixins in der CSS Datei angezeigt wird, fügt man am Ende zwei Klammern hinzu.

```
.fontmedium(){
  ...
}
```

Zusammenfassung

Mixins werden verwendet um den selben Code nicht unnötig oft zu wiederholen. Mit Parametern werden Mixins deutlich flexibler. Mit einer leeren Runden Klammer werden die Mixins nicht in die CSS Datei übernommen.

Operatoren

In LESS kann man gewohnt wie in anderen Programmiersprachen Rechenoperationen durch führen.

```
@base: 5%;
@filler: @base * 2;
@other: @base + @filler;
@base-color: #333;

body {
  color: #888 / 4;
  background-color: @base-color + #111;
  height: 100% / 2 + @filler;
}
```

Das sind die Theoretischen Möglichkeiten für was man Rechenoperationen verwenden kann. Wie würde aber ein reales Beispiel aussehen?

```
@spalten: 12p;
@breite: 70px;
@abstand: 10px;
@element-breite: @spalten * (@breite + @abstand) + @ababstandstand;
```

```
main {
  width: @element-breite;
}
```

Zusammenfassung

Rechnungen können nützlich sein, z.B. für die Definition von Höhen und Breiten, das Rechnen mit Farben wird eher weniger verwendet.

Import

Bei größeren Projekten will man den CSS Code auf mehrere Dateien aufteilen.

```
*@import* "url";
```

import sollte in reinem CSS vermieden werden. Wenn mehrere CSS Dateien über import eingebunden werden, gibt es für jede CSS Datei einen Request der abgewartet wird. Die Performance der Webseite leidet darunter.

```
*@import* 'font'; *// Kein seperates Dokument wird im Browser eingebunden*

*@import* 'font.css' // Ein CSS File wird mit einem Request vom server geladen
```

Wenn man mehrer CSS Dateien hat, am besten in LESS Dateien umwandeln um den Code in eine Datei einzubinden. Wenn eine Datei nicht berücksichtigt werden soll kann sie mit einem // aus kommentiert werden. Wenn man den CSS üblichen Kommentar verwendet, bekommt man im Output File diesen Code /* import file.css */

[Übung 6](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_6_imports.zip)

SASS

Install SASS

```
npm install -g sass
```

SASS Dokumentation

Eine Funktion die bei SASS ohne Zusatz vorhanden ist, ist die Watch Funktion. Watch weist SASS an die Quelldateien auf Änderungen zu überwachen und jedes mal neu zu kompilieren wenn die SASS Dateien gespeichert werden.

```
sass --watch input.scss:output.css
```

Wenn ein gesamte Ordner überwacht werden solle kann man schreiben

```
sass --watch .
```

[Übung 7](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_1_cli.zip)

Zusammenfassung

Im Gegensatz zu LESS bietet SASS out of the box die Watch Funktion. Es muss nicht jedesmal der Befehl in die CLI eingegeben werden, sass überwacht die Dateien auf Änderungen und führt die Umwandlung aus.

SASS GUI

- Scout
- Koala
- Sass Meister

Variablen

werden in SASS mit einem **$** definiert.

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;
```

Die Zuweisung der Variablen erfolgt gleich wie in LESS.

```
body {
  font: $font-stack;
  color: $primary-color;
}
```

[Übung 8](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_2_variablen.zip)

Schreibweisen

Von SASS gibt es 2 verschiedene Schreibweisen. Es gibt die SCSS Schreibweise und die Sass Schreibweise. SCSS ähnelt CSS sehr und es werden {} verwendet. Bei der Sass Schreibweise werden Zuweisungen mit Einrückungen definiert.

- Es gibt keinen Strichpunkt am Ende
- Bei Block Deklarierungen werden keine {} verwendet

Indented Syntax

```
body
  color: red
  padding: 10px
  font-family: Helvetica
```

Es wird die Dateiendung **.sass** verwendet

Nesting

```
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block;
```

```
    a {
      display: block;
      padding: 6px 12px;
      text-decoration: none;
    }
  }
}
```

Sass könnt ihr eure CSS-Selektoren verschachteln. Um eine visuelle Hierarchie, ähnlich wie HTML.

[Übung 9](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_3_nesting.zip)

Mixins

```
*@mixin* border {
  padding: 20px;
  border: 1px solid green
}

*@mixin* transition {
  -webkit-transition: background-color 2s ease-out;
  -moz-transition: background-color 2s ease-out;
  -o-transition: background-color 2s ease-out;
  transition: background-color 2s ease-out;
}
```

Im Gegensatz zu LESS werden Mixins nicht wie CSS Klassen definiert sondern mit dem Zusatz **@mixing**. Zum Aufrufen des Mixins wir @include verwendet.

```
p {
  @include border;
}
```

Gleich wie in LESS können auch Parameter verwendet werden.

```
@mixin transition($duration: 1s, $property: all, $function: ease) {
  -webkit-transition: $duration $property $function;
  -moz-transition: $duration $property $function;
  -o-transition: $duration $property $function;
  transition: $duration $property $function;
}

.box {
  @include transition(2s, all, ease)
}
```

[Übung 10](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_5_mixins.zip)

Operatoren

```
.container {
  width: 100%;
}

article {
  float: left;
  width: 600px / 960px * 100%;
}

aside {
  float: right;
  width: 300px / 960px * 100%;
}
```

Wir haben ein sehr einfaches, Raster auf der Basis von 960px erstellt. Mit den Operationen

in Sass können wir so etwas wie Pixelwerte nehmen und sie ohne großen Aufwand in Prozentwerte umwandeln.

Import

Aus performance Gründen sollten so wenige Dateien wir möglich verwendet werden. Je mehr Dateien über CSS Native eingebunden werden, desto mehr Requests werden an den Server gehen und die Performance beeinflussen.

```
Eine CSS Datei wird Eingebunden und somit ein Request an den Server gestellt
@import "style.css";
Eine sass datei wird beim compilieren eingebunden und der Code welcher in
dieser Datei steht in die app.css datei eingefügt
@import "style";
Diese Zeile wird übersprungen.
//@import "style";
```

[Übung 11](https://marcpeternell.github.io/preprocessors/uebungen/Beispiel_6_imports.zip)

Extend/Inheritance

Das ist eines der nützlichsten Merkmale von SASS. Mit @extend kann ein Code Block von Eigenschaften eines Selektors zu einem weiteren weitergeben. Dadurch bleibt der Code übersichtlich. Im folgenden Beispiel werden Meldungen für Fehler, Warnungen und Erfolge erstellt.

```
*/* This CSS will print because %message-shared is extended. */*
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

*// This CSS won't print because %equal-heights is never extended.*
%equal-heights {
  display: flex;
  flex-wrap: wrap;
```

```
}

.message {
  *@extend* %message-shared;
}

.success {
  *@extend* %message-shared;
  border-color: green;
}

.error {
  *@extend* %message-shared;
  border-color: red;
}

.warning {
  *@extend* %message-shared;
  border-color: yellow;
}
```

Output:

```
*/* This CSS will print because %message-shared is extended. */*
.message, .success, .error, .warning {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}

.warning {
```

```
  border-color: yellow;
}
```

Der Code weist .message, .success, .error und .warning an, sich genau wie %message-shared zu verhalten. Das bedeutet, dass überall, wo %message-shared auftaucht, auch .message, .success, .error und .warning auftaucht. Die Magie geschieht im generierten CSS, wobei jede dieser Klassen die gleichen CSS-Eigenschaften erhält wie %message-shared.

Hinweis: %equal-heights wird nicht generiert, da %equal-heights niemals erweitert wird.