Thema 1 - LESS & SASS

Vorteile von CSS Präprozessoren

- CSS Code ist bei großen Projekten ewig lang und unübersichtlich
- Präprozessoren helfen den CSS Code eleganter zu gestallten und zu organisieren
- Bei Farbänderungen müssen semtliche Stellen gesucht und geändert werden, mit Präprozessoren definiert man eine Variable. Diese muss nur einmal geändert werden.

Gründe für Präprozessoren:

- die Definierung 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 sie einmal definieren und können durch einen einzelnen Befehl aufgerufen werden und wiederverwendet werden
- Farbverläufe können sehr lange werden, speziell für ältere Browser müssen mehrere Zeilen Code geschrieben werden
- Style sheets 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 Präprozessoren
    - Boostrap -> source code
        - Bei Customize werden die LESS Dateien bearbeitet
    - Foundation 6 source code
- Mehr für die Spaßigen sachen an CSS

Grundprinzip von Präprozessoren

- Ohne Präprozessoren: Die Seite wird angeschaut, das **CSS File** wird geändert und aktualisert.
- Sobald mit Präprozessoren gearbeiter 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 compiler

```
$meinefarbe: red;
```

```
@mixin uebergang {
  -webkit-transition: 1s all ease;
  transition: 1s all ease;
}

p, a {
  @include uebergang;
}

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

less compiler

```
@meine-farbe: red;
@breite: 10px;
@rahmen: 10px solid blue;
@schrift: Helvetica, Arial, sans-serif;
@wichtige-farbe: orange;
@zweite-farbe: blue;

body {
  color: @meine-farbe;
  font-family: @schrift;
  padding: @breite;
  border:@rahmen;
}

footer {
  color : @zweite-farbe;
  background-color: @meine-farbe;
}
```

LESS & SASS

less
sass

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 Framwork 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?

- Boostrap baut auf LESS auf

Was soll ich verwenden?
Was wird in der Umgebung eingesetzt? Wenn in der **Arbeitsumgebung** verschiedene Präprozessoren eingesetzt werden kommt es zu Fehlern.

LESS Getting Started

LESS kann auch ohne Präprozessor direkt im Browser angezeigt werden.

```
<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>
```

Zusammenfassung
LESS dateien können auch ohne Präprozessoren im Browser verarbeitet werden, jedoch muss dazu die .js Datei von LESS direkt eingebunden werden.

1. Die eigene LESS Datei einbinden **bei rel(relationship) muss /less angefügt werden**
2. Die JS Datei von LESS direkt einbinden

CLI

Die LESS Dateien zuerst compilieren und anschließend an den Browser zu übergeben.

> node muss 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 compilierung wird der command **lessc (less compile)** verwendet

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

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

Übung 1

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

GUIs for less

Vorteil von GUIs viele Compiler minify'n die finalen CSS Dateien von alleine.

DRY

(don't repeat yourself)

```
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;
}
```

Wenn man sich entscheide 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;
@breite: 10px; // Werte
@rahmen: 10px solid purple; // Complexere Ausdrücke
@wert: true; // Wird interessant wenn mit Bedingungen gearbeitet wird
@schrift: Helvetic, Arial, sans-serif; // Listen
```

Die Zuweisung der Variablen erfolgt gleich wie in CSS.

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

Übung 2

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 Selectoren 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 Selectoren schreiben.

Übung 3

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;
      }
    }
}
```

Übung 4

Warum "&"

Beim compilieren 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 selectoren

Mixins

Mixins erlauben Code wieder zu verwenden welche aus mehr als einer Zeile bestehen. Für einfache Fälle werden Variablen genommen welche an underschiedlichen Stellen eingesetzt werden können. Mixins schauen in LESS genau so aus wie Klassendefinitionen

```
@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 m
it der Verärbung hat, es wird immer das Root-Element genommen
}
.uebergang {
  -webkit-transition: 1s all ease;
  transition: 1s all ease;
}

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

nav {
  a {
    .uebergang;
  }
}
```

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

Übung 5

Parameter

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

```
.uebergang_angepasst(@dauer: 1s, @eigenschaft: all, @art: ease) {
  -webkit-transition: @dauer @eigenschaft @art;
  transition: @dauer @eigenschaft @art;
}

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

Mixins ohne Parameter weden 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 unnötg oft zu wiederholen. Mit Parametern werden Mixins deutlich flexiebler. 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 Beipiel aussehen?

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

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

Zusammenfassung
Rechnungen können nützlich sein für die definiereung 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 mehere 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 berüchtsichtigt werden soll kann sie mit einem // auskommentiert werden. Wenn man den CSS üblichen Komentar verwendet, bekommt man im Output File diesen Code /* import file.css */

Übung 6

SASS

Install SASS

```
npm install -g sass
```

SASS Dokumentation

Eine Funktion die bei SASS ohne Zusatz vorhanden ist, ist die Watch Funktion

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

Wenn ein gesammte Ordner überwacht werden solle kann man schreiben

```
sass --watch .
```

Übung 7

Zusammenfassung
Im gegensatz zu LESS bietet SASS out of the box die Watch funktion. Es muss nicht jedesmal der Befehl in die Comando Zeile eingegeben werden, sass überacht 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 ähnlich wie in LESS.

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

Übung 8

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 Hierachie, ähnlich wie HTML.

Übung 9

Mixins

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

@mixin transition {
  -webkit-transition: 1s all ease;
  transition: 1s all ease;
}
```

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, $property, $type){
  -webkit-transition: $duration $property $type;
  transition: $duration $property $type;
}

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

Übung 10

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 diese
r Datei steht in die app.css datei eingefügt
@import "style";
Diese Zeile wird übersprungen.
//@import "style";
```

Übung 11