

LANGUAGE EXTENSIONS FÜR CSS3

MMWP2024 - LV10

INHALTSVERZEICHNIS

- Organisation
- Erweiterungssprachen
- SASS und LESS
- Frameworks und Pakete

INHALTSSCHWERPUNKTE

- CSS-PreProcessors
- Integration von CSS-Erweiterungssprachen
- SASS-SCSS PreProcessors
- Aktuelle Entwicklungen von Erweiterungssprachen

VORAUSSETZUNG

Der Ausgangspunkt dieser Vorlesungsreihe ist das Wissen über Themen:

- Wissen über CSS und gängige Konzepte wie MediaQueries
- Funktionsweisen von PreProcessors
- Grundlegende Verständnisse von Compiler- und Interpreter-Konzepten

ZIELE

Vorstellung von:

- SASS/SCSS
- CSS-Variablen und SASS-Funktionen
- Andere CSS-Erweiterungssprachen
- Frameworks welche Erweiterungssprachen selbständig benutzen

ERWEITERUNGSSPRACHEN - ANWENDUNGSGRÜNDE

- Codewiederholung vermeiden (oft müssen n-fach verwendete CSS-Anweisungsblöcke auch n-fach ausgeschrieben werden)
- Browserpräfixe verarbeiten
- Vorberechnung von Werten für die CSS-Definitionen
- Bessere Strukturierung und Hierarchie bei Komplexität erreichen
- Bedingungen und Transitivität bei Vererbung besser verarbeiten
- Schleifen bedingte CSS Regeln automatisiert generieren

ERWEITERUNGSSPRACHEN - STRUKTURIERUNG UND HIERARCHIE

- Angabe von CSS3-Strukturen in verschachtelter Schreibweise (nested properties)
- Mithilfe von Selektorenverschachtelung
- Nutzung von vor- und selbstdefinierten Funktionen (Mixins)
- Verbesserte Lesbarkeiten der CSS-Regeln
- Media-Queries erlauben doppelte Schachtelung durch Angaben in den Media-Queries, was (im Moment) nicht unterstützt wird

ERWEITERUNGSSPRACHEN - BEDINGTHEIT UND VARIABILITÄT

- Der Einsatz von Variablen und bedingten Anweisungen soll möglich werden
- Die Beschränkungen der calc()-Funktion sollen überwunden werden
- Auch hier: Bedingte Anweisungen durch Media-Queries funktionieren (noch) nicht
- Bestimmte Selektoren mit Angabe von Bedingungen bei Elementen, z.B. bestimmte Attributwerte werden kaum unterstützt

ERWEITERUNGSSPRACHEN - KOMPATIBILITÄT

- Angaben zu CSS-Eigenschaften, die Browserpräfixe für die Quer-/Abwärtskompatibilität benötigen
- Durch Angabe des CompileBefehls werden aus Stamm-CSS-Regeln, Browserspezifische CSS-Regeln
- Die Beschränkungen der calc()-Funktion sollen überwunden werden
- Auch hier: Bedingte Anweisungen durch Media-Queries funktionieren (noch) nicht
- Bestimmte Selektoren mit Angabe von Bedingungen bei Elementen, z.B. bestimmte Attributwerte werden kaum Unterstützt
- **Browser Prefixe**

ERWEITERUNGSSPRACHEN - KRITIK

- Großer Aufwand um marginale Probleme zu lösen
- Nicht standardisierte Syntax (jede Erweiterungssprache ist unterschiedlich)
- bei Compilerfehlfunktion wird kein CSS ausgeliefert
- Ähnliche Funktionen lassen sich serverseitig durch Sprachen wie PHP oder NodeJS realisieren
- **CSS Preprocessors**





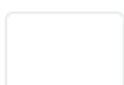
VERBREITETE ERWEITERUNGSSPRACHEN

- LESS: [LESS Seite](#)
- Sass/SCSS: [SASS Seite](#)
- Stylus: [Stylus Seite](#)
- PostCSS: [PostCSS Seite](#)

WEITERE ERWEITERUNGSSPRACHEN

- CSS-crush: [CSSCrush](#)
- Rework (mit node.js zusammen): [Rework](#)
- Myth (auf Basis von Rework): [Myth](#)
- Clay (als Embedded Domain Specific Language, Haskell-basiert): [Clay](#)
- DtCSS (als PHP-Skript): [DTCSS](#)
- CSS-on-diet : [CSS on diet](#)

ERWEITERUNGSSPRACHEN NUTZERANTEIL

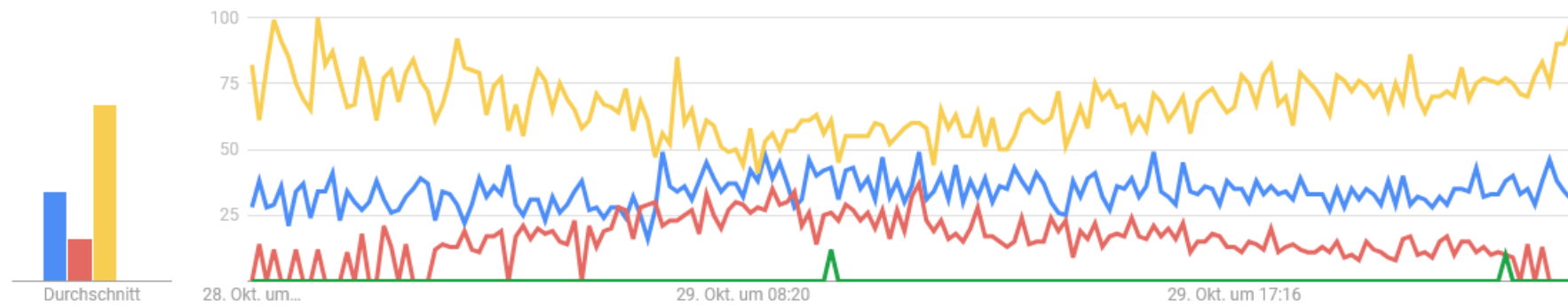
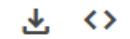
BEST CSS PREPROCESSORS/POSTPRO...		PRICE	LAST UPDATED
85	 Sass	-	Nov 26, 2023
74	 Stylus	-	Oct 4, 2022
68	 Less	-	Feb 15, 2023
60	 PostCSS	-	Sep 5, 2022
--	 cssnext	-	Oct 13, 2020

[↗](#) SEE FULL LIST

Quelle

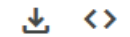
ERWEITERUNGSSPRACHEN SUCHANFRAGEN

Interesse im zeitlichen Verlauf ?



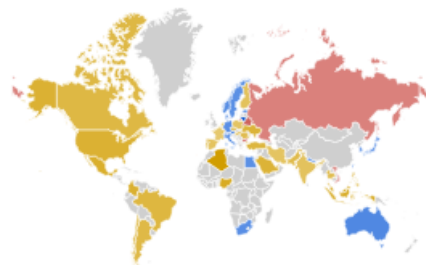
Aufschlüsselung nach Region

Region ▼



● SASS ● SCSS ● Stylus ● PostCSS

Sortieren: Interesse an "SASS" ▼



1	Litauen	<div><div></div></div>
2	Estland	<div><div></div><div></div></div>
3	Nepal	<div><div></div><div></div></div>
4	Australien	<div><div></div><div></div><div></div></div>
5	Schweiz	<div><div></div><div></div><div></div></div>

Quelle

INTEGRATION

- Präprozessor: Konvertiert die Ausgangsdatei in der LE-Sprache nach Validierung in CSS, wobei die Zielfdatei abgespeichert und gegebenenfalls gleich verwendet wird
- Clientseitig: LESS bei Chrome, Firefox, Safari und IE, mit JavaScript-Unterstützung über DOM (live)
- Serverseitig: Sass/SCSS, Stylus und LESS, Abspeicherung im Quellverzeichnis der Website (offline). Eventuell „watchdog“ für Updates mit sofortiger Kompilierung

GRUNDLAGEN - SASS

- Nutzung einer Dart-RTE, einer C/C++-RTE (LibSass) oder Ruby-RTE (wird nicht mehr entwickelt) möglich
- Kompression der CSS-Dateien möglich
- Zusammenfassung der CSS-Dateien als eine Hauptstyle-Datei

GRUNDLAGEN - LESS

- Nutzung von NodeJS
- Kompression der CSS-Dateien möglich (nicht mehr empfohlen → Zusatzbibliotheken nutzen)

GRUNDLAGEN - SASS/SCSS VS LESS

- SASS/SCSS
 - Sass imperativ orientiert
 - SCSS gestaltet die Syntax als CSS-Erweiterung aus
 - Grundphilosophie bei Sass ist die eines klassischen Programmierers
- LESS
 - Deklarativ-funktional orientiert (Schleifen als Rekursionen, if-Anweisungen als Guard-Bedingungen für Mixins)
 - Grundphilosophie dem Designer näher

GRUNDLAGEN - SCHREIBWEISEN

- Prinzip des Einrückens (Strukturierung Sass):
 - Selektoren und jede Eigenschaftsdeklaration stehen in separaten Zeilen, letztere werden gegenüber dem Selektor eingerückt, keine Zeichen am Zeilenende notwendig
 - Nacheinander stehende Zeilenanfänge auf gleicher Einrückungstiefe bilden einen Block
 - Der Doppelpunkt ist das Signal zu Wertzuweisung für Eigenschaften
- LESS
 - Schreibweise (fast) wie in CSS-Dateien
 - @import auch lokal möglich

NESTED PROPERTIES IN SASS

```
1 .selector {  
2   border-width: 10px;  
3   border-color: black;  
4 }  
5 .selector {  
6   border: {  
7     width: 10px;  
8     color: black;  
9   }  
10 }
```

VARIABLEN - 1

```
1 //Sass
2 $color: red; // erste!
3 #menu a {
4   color: $color;
5 }
6 $color: "rot"; // zweite!
7 #menu-{$color} a {
8   border-color: red;
9 }
10
11 //LESS
12 @color: red; //final, global
13 #menu {
14   color: @color;
15   @color: green; // final lokal
```

VARIABLEN - 2

- SASS/SCSS
 - Globale und lokale Variablen (auch mit selben Namen, dann beziehen sich lokale Zuweisungen nur auf die lokale Variable → Shadowing)
 - Neubelegung der Variablen beim Kompilieren zulässig, dabei Typisierung
 - Typ der Variable kann beim Kompilieren wechseln
 - [Quelle](#)
- LESS
 - Globale und lokale Variablen, letzte Belegung zählt
 - Suche nach Variable von innen nach außen, bei mehrfachem Vorkommen mit gleichem Namen
 - Es gibt Variablentypen
 - [Quelle](#)

DATENTYPEN

Sass

- Zahlen (gegebenenfalls mit Einheiten)
- Zeichenketten
- Farben
- Booleans
- NULL
- Listen (von Einzelwerten)
- Maps (d.h. Listen von Paaren Schlüssel+Wert)

Listen und Maps können auch ineinander geschachtelt werden.

LESS

- Zahlen
- Zeichenketten
- Farben
- Booleans
- NULL
- Listen (von Einzelwerten)

MIXINS IN SASS

- Ein Mixin ist in Sass ein Definitionsblock von CSS-Styles, der einen eindeutigen Bezeichner besitzt und vielfach wieder aufgerufen werden kann
- Es kann parametrisiert werden, was beim Aufruf andere Parametersetzungen erlaubt
- Definition: Über @mixin-Kennung
- Aufruf: Über @include-Kennung, mit Setzung eventueller Parameter im Mixin
- [Quelle](#)

MIXIN-BEISPIEL

```
1 @mixin bordered-box($farbe,$breite){  
2   border:{  
3     width: $breite+px;  
4     style: solid;  
5     color: $farbe;  
6   }  
7 }  
8 #red-box {  
9   @include bordered-box(red,4);  
10 }
```

MIXIN IN LESS

- Mixins in LESS sind Definitionsblöcke mit z.B. Stilangaben, die sich wiederverwenden lassen
- De facto sind es künstlich erzeugte Selektoren
- Wenn Klammern hinter dem definierenden Wort des Mixins stehen, so erfolgt eine Ausgabe in die erzeugte CSS-Datei hinein, andernfalls nicht
- Mixins können rekursiv verwendet werden
- Mixin-Guards garantieren die Abbruchbedingungen

MIXIN-GUARDS-BEISPIELE - 1

```
1 .mixin (@farbe) when (lightness(@farbe) >= 50%) {  
2   background-color:black;  
3 }  
4 .mixin (@farbe) when (lightness(@farbe) < 50%) {  
5   background-color:white;  
6 }  
7 .mixin (@farbe) {  
8   color: @farbe;  
9 }  
10 #box {  
11   .mixin(#eeeeee);  
12 }
```

MIXIN-GUARDS-BEISPIELE - 2

```
1 & when (lightness(@farbe) >= 50%) {  
2   background-color:black;  
3   p {  
4     color:@farbe;  
5   }  
6 }
```

KONTROLLIERTER IMPORT

- Wichtig: CSS-Präprozessoren ignorieren beim @import-Befehl Dateien mit einer .css-Endung
- Importieren lassen sich somit nur proprietäre Dateien, die beispielsweise auf .less, .scss oder .sass enden
- Zuerst wird beim Kompilieren der Quelltext aller importierten Dateien mit einbezogen (außer CSS!), und dann entsteht genau eine CSS-Datei

KONTROLLIERTER IMPORT - URSPRUNGSQUELLTEXT

```
1  /* file.{type} */
2  body {
3    background: black;
4  }
5
6  @import "reset.css";
7  @import "file.{type}";
8  p {
9    background: white;
10 }
```

KONTROLLIERTER IMPORT - RESULTAT

```
1 @import "reset.css";
2 /* kompilierter Anteil ... */
3 body {
4     background: black;
5 }
6 p {
7     background: white;
8 }
```


FUNKTIONEN

LESS hat eine Sammlung von Funktionen in seiner Bibliothek. Sass erlaubt es, eigene Funktionen zu erzeugen

```
1 @function column-width ($page-width){  
2   @return $page-width / 3 - 10px;  
3 }  
4  
5 $red: #ff0000;  
6 $dark-red: $red / 2;
```

FUNKTIONEN

Es gibt Bedingungen zur Ablaufsteuerung und es gibt for-, while- und each-Schleifen in Sass

```
1 .selektor {  
2   @if $variable == "wert1" { color: red; }  
3   @else if $variable == "wert2" { color: green; }  
4   @else $variable == "wert3" { color: blue; }  
5 }
```

ABLAUFSTEUERUNG IN SASS - 1

```
1 /* for-Schleife mit 5 Durchläufen */
2 @for $i from 1 through 5 {
3     #objekt-#{ $i } { margin-top: $i * 10px; }
4 }
5
6 /* while-Schleife */
7 $i: 1;
8 @while $i < 10 {
9     .item-#{ $i } { width: 2em * $i; }
10    $i: $i + 2;
11 }
```

ABLAUFSTEUERUNG IN SASS - 2

```
1 /* each-Schleife mit Laufvariablen-Liste */
2 @each $name, $farbe, $abstand in (anton, orange, 100px), (berti, green, 1
3     .#{ $name } {
4         position: absolute;
5         top: $abstand;
6         background-image: url('/images/#{ $name }.png');
7         border: 10px solid $farbe;
8     }
9 }
```

@EXTEND-ERWEITERUNG IN SASS

Erweiterung funktioniert wie Vererbung bei Klassen

```
1 .box { //bleibt unverändert
2   padding: 10px;
3 }
4
5 .box-mit-rahmen {
6   @extend .box; //padding: 10px;
7   border: 1px solid red;
8 }
```

&:EXTEND-ERWEITERUNG IN LESS

In LESS folgt das Thema der Erweiterung eher dem Gedanken der Wiederverwendung z.B. von CSS-Klassen

```
1 nav ul {  
2   &:extend(.inline);  
3   background: blue;  
4 }  
5  
6 .inline {  
7   color: red;  
8 }
```

FRAMEWORKS UND PAKETE

- Sass/SCSS, LESS
 - Bootstrap 5+: [SASS Support](#)
- Sass/SCSS
 - Foundation 5+: [Foundation Seite](#)
 - Bourbon – a mixin library: [Bourbon](#)
 - Susy: [Susy Seite](#)
 - [Sierra](#)
 - [Materializecss](#)
 - [Gridle](#)
- LESS
 - 3L: [Quelle](#)

INSTALLATION UND NUTZUNG

- LESS
- Sass
- Stylus

VERSCHIEDENE COMPILER

CSSPRE Übersicht

WEITERES

- [CSS Pre Prozessor Übersicht](#)
- große Sammlung mit Vergleichen zu Eigenschaften, Compilern, Konvertieren und mehr: [Quelle](#)
- Tutorial zu LESS: [Quelle](#)
- sehr gutes, sechsteiliges Tutorial: [Quelle](#)
- Einführung und Vergleich der Syntax: [Quelle](#)

QUELLEN

- Vergleich der Fähigkeiten von Sass, LESS, Stylus nach Features, aktuell: [Quelle](#)
- Konverter nach CSS oder cross unter den Präprozessorsprachen: [CSS Konvertierung](#)
- Scott Logic, Vergleich: [LESS vs STYLUS vs SASS](#)
- [Mozilla CSS pre processor Übersicht](#)
- [SASS Seite](#)

ABSPANN

Zehntes Level geschafft zwei weitere folgen

Fragen und Feedback?