

Build Tasks Repetition

WebFonts nachladen:

- «CSS Font Loading Module Level 3», *Editor's Draft* • <https://drafts.csswg.org/css-font-loading/>
- Promise-basierende JavaScript-API

oder mit Java Loader funktion.

- «CSS Font Loading Module Level 3», *Editor's Draft* • <https://drafts.csswg.org/css-font-loading/>
- Promise-basierende JavaScript-API

Spezifischen Font laden:

```
var fontFace = new
FontFace( 'MyWebFont' ,
'url( "xy.woff2" )
    format( "woff2" ) ,
url( "xy.woff" )
format( "woff" ) ' );

fontFace.load( ).then( function( loadedFontFace )
{

document.fonts.add( loadedFontFace );
```

```
document.documentElement.  
classList.add( 'font_l  
oaded' );  
} );
```

Beispiel: Klasse
hinzufügen, wenn alle
Fonts geladen wurden

- Demo: <http://codepen.io/backflip/pen/BzbAAP>
- **Achtung:** "Latest version" der W3C erwähnt "ready()" - Methode, inzwischen ist es

im Editor's Draft ein Attribut:

```
document.fonts.ready.then(function()  
{ document.documentElement.classList.add('font  
s_loaded');  
});
```

Was ist ein CSS-Präprozessor?

- Superset von CSS → CSS
- Ansatzweise vergleichbar mit

Typescript

Umwandeln in valides CSS

Vorteile / Möglichkeiten von Präprozessoren

- Nicht an Limitationen von CSS gebunden
- Möglichkeit zu effizienterem und besser wartbarem Code

Vorteile/Möglichkeiten von Präprozessoren

(Auszug)

- Variablen
- Nesting, Parent-Selektor
- Partials/Import
- Mixins (mit Parametern)
- Extend, Placeholder-Selektor
- Funktionen & Operationen
- Loops & Control Directives

Alternative: Variablen
in CSS (mittlerweile in
aktuellen Browser
meist unterstützt)

```
:root {
    --color-main:
#73c92d;
}

a {
    color: var(--color-
main);
}}

.highlight {
    background-color:
var(--color-main);
```

Parametric Mixins

```
@mixin border-radius($radius) {
    -webkit-border-radius: $radius;
    -moz-border-radius: $radius;
    -ms-border-radius: $radius; für alte Browser z.B
    border-radius: $radius;
}

.box {
    -webkit-border-radius: 1rem;
    -moz-border-radius: 1rem;
    -ms-border-radius: 1rem;
    border-radius: 1rem;
}

.box {
    @include border-radius(1rem);
}
```

Parametric Mixins (1)

```
@mixin breakpoint($size) {
  @media(min-width: $size) {
    @content;
  }
}

.conent: für mediaquery praktisch : Content ist background:red

.element {
  @include breakpoint(1000px) {
    background: red;
  }
}
```

→

```
@media(min-width: 1000px) {
  .element {
    background: red;
  }
}
```

Parametric Mixins (2)


```
@mixin triangle($direction:"down", $color:#000, $size:1em, $selector:"after") {
  &:#{ $selector } {
    height: 0;
    width: 0;
    content: "";
    position: absolute;
    border: $size solid transparent;

    @if $direction == "up" {
      border-bottom-color: $color;
    }
    @if $direction == "down" {
      border-top-color: $color;
    }
  }
}

.expand {
  @include triangle;
}

.collapse {
  @include triangle("up");
}
```

→ ein triable machen



@extend

```
%box {
  padding: 1em;
  border: 1px solid #ccc;
}

.message {
  @extend %box;
  width: 100%;
  color: #333;
}

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

.error {
  @extend .message;
  border-color: red;
}
```

einfach erweiter und erweiter und erweiter

→

```
.message,
.success,
.error {
  padding: 1em;
  border: 1px solid #cccccc;
  width: 100%;
  color: #333;
}

.success {
  border-color: green;
}

.error {
  border-color: red;
}
```

@mixins vs @extend

- Generell: Vor- und Nachteile bei allen Ansätzen
- Wichtig: **generiertes CSS im Auge behalten**
 - Risiko: was kommt dabei raus?
- Mixins:
 - Einfach wartbar, Resultat vorhersehbar
 - Resultat kann sehr redundant sein. Aber: **gzip!** → <https://tech.bellycard.com/blog/sass-mixins-vs-extends-the-data/>
- @extend (mit oder ohne %placeholder):
 - Reduktion des generierten Codes
 - Resultat nicht immer vorhersehbar, funktioniert nicht über Media-Queries hinweg
 - funktioniert nicht mit mediaquery

@extend: Caveats

- Position des zu extendenden Selektors relevant für generiertes Resultat
- Jedes Vorkommen wird extended:

```
.message + .message {  
  margin-bottom: .5em;  
}  
...  
.message-error {  
  @extend .message;  
}  
...  
→  
.message + .message,  
.message-error + .message-error,  
.message + .message-error,  
.message-error + .message {  
  margin-bottom: .5em;  
}  
...  
Extend ist also heikler - besser mixins
```

- [Beispiel 1](#), [Beispiel 2](#), Artikel «What Nobody Told You About Sass's @extend» (Hugo Giraudel, 2014)

@extend: Caveats

- Position des zu extendenden Selektors relevant für generiertes Resultat
- Jedes Vorkommen wird extended:

```
.message + .message {  
  margin-bottom: .5em;  
}  
...  
.message-error {  
  @extend .message;  
}  
...  
→  
.message + .message,  
.message-error + .message-error,  
.message + .message-error,  
.message-error + .message {  
  margin-bottom: .5em;  
}  
...  
Extend ist also heikler - besser mixins
```

- [Beispiel 1](#), [Beispiel 2](#), Artikel «What Nobody Told You About Sass's @extend» (Hugo Giraudel, 2014)

```
<a href="#" class="link var_error">error</a>  
</br>  
<a href="#" class="link var_success">success</a>  
</br>  
<a href="#" class="link var_info">info</a>  
</br>  
<a href="#" class="link var_organic">organic</a>
```

```
@mixin typeStyles($type) {  
  &.var_#{$type} {  
    @content;  
  }  
}
```

```
.link {  
  @include typeStyles("error") {  
    color: firebrick;  
  }  
}
```

```
}  
@include typeStyles("success") {  
  color: olivedrab;  
}  
@include typeStyles("info") {  
  color: lightblue;  
}  
@include typeStyles("organic") {  
  color: tomato;  
}  
}
```

error

success

info

organic

Build TASKS:

GRunt : Knotig

Gulp: Code (Code over Konfiguration)

API (Version 3.x)

- **gulp.task:** Definiert Task
- **gulp.src:** Liest Dateien und generiert Stream
- **gulp.dest:** Schreibt Dateien im Stream wieder ins Dateisystem
- **gulp.watch:** Triggert Tasks bei Änderungen an Dateien

```
var gulp = require('gulp'),
    jshint = require('gulp-jshint'),
    concat = require('gulp-concat'),
    uglify = require('gulp-uglify');

gulp.task('js', function() {
    return gulp.src('source/*.js')
        .pipe(jshint())
        .pipe(jshint.reporter('default'))
        .pipe(concat('main.js'))
        .pipe(uglify())
        .pipe(gulp.dest('build/'));
});

gulp.task('default', function() {
    gulp.watch('source/*.js', ['js']);
});
```

Was ist ein "Frontend-Framework"?

- Werkzeugkasten mit Komponenten aus HTML-, CSS- und JS-Bausteinen
- Frontend-Framework ≠ JavaScript-Framework

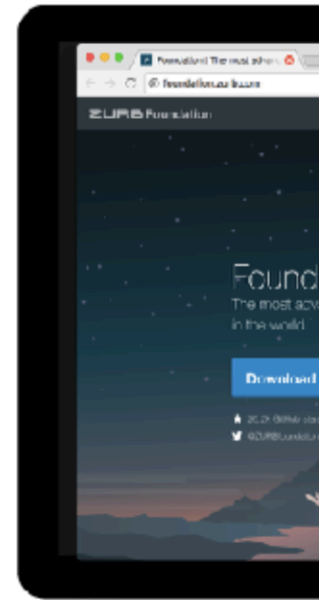
Foundation:

Foundation

«The most advanced responsive front-end framework in the world.»

<http://foundation.zurb.com/>

Standardstyles sind gut konfigurierbar. Mit Sass-Variablen konfigurierbar (mit Bootstrap weniger konfigurierbar - mehr überschrieben nötig)



Kriterien

- Funktionsumfang
- Konventionen / Struktur der Source (Präprozessor, JS-Modul-Pattern)
- Wichtig: **Nur notwendige Komponenten laden!**
- Beispiel: post.ch nutzt Grid-Komponente von Foundation

Übung

- *Aufgabe:* Komponente aus Bootstrap oder Foundation verwenden
- *Ansatz:*
 - Framework mit npm installieren: **npm install bootstrap-sass**
npm install foundation-sites
 - Komponente auswählen: Beispiel "Dropdown"
 - HTML-Seite mit Demo-Markup erstellen
 - "js"-Task anpassen, um notwendiges JS zu laden
 - "css"-Task anpassen, um notwendiges (S)CSS zu laden
- *Lösung:* <https://github.com/cas-fe/12-Gulp-Demo> (Branch **feature/bootstrap**)

```
var gulp = require('gulp'),
    jshint = require('gulp-jshint'),
    concat = require('gulp-concat'),
    uglify = require('gulp-uglify'),

    sass = require('gulp-sass'),

    autoprefixer = require('gulp-autoprefixer'),
    postcss = require('gulp-postcss'),
    cssnano = require('cssnano');

gulp.task('js', function() {
    return gulp.src('source/*.js')
        .pipe(jshint())
        .pipe(jshint.reporter('default'))
        .pipe(concat('main.js'))
        .pipe(uglify())
        .pipe(gulp.dest('build/'));
});

gulp.task('sass', function () {
    return gulp.src('./source/*.scss')
        .pipe(sass().on('error', sass.logError))
        .pipe(autoprefixer({
            browsers: ['last 2 versions'],
            cascade: false
        })))
});
```

```
        .pipe(gulp.dest('build/')));
});

// postcss: mit braucht's keine buffering
// file buffered -> cssnano -> und wieder buffered - sonst
// verwendet man postcss
// postcss auch wegen den ganzen Filemap besser so

gulp.task('autoprefixer', () =>
    gulp.src('build/*.css')
        .pipe(postcss([autoprefixer(), cssnano()])))
        .pipe(uglify())
        .pipe(gulp.dest('dist'))
);

gulp.task('default', function() {
    gulp.watch('./source/*.scss', ['sass']);
    gulp.watch('./build/*.css', ['autoprefixer']);
});
```