Tag 7 CSS: CSS-Präprozessoren - Sass & Less

1. Was sind CSS-Präprozessoren?

Ein CSS-Präprozessor ist ein Tool, das eine eigene Stylesheet-Sprache (z. B. Sass oder Less) in normales CSS „übersetzt". Damit können Entwickler:innen **Variablen**, **Funktionen**, **Verschachtelung**, **Vererbung** und **Logik** verwenden – Features, die in reinem CSS nicht oder nur begrenzt verfügbar sind.

> Sass und Less vereinfachen die Entwicklung großer CSS-Projekte und verbessern die Wartbarkeit des Codes.

---

2. Sass vs. Less – Überblick

Feature	Sass	Less
Syntax	`.scss` oder `.sass`	`.less`
Installation	via npm, Dart Sass, VS Code Plugin	via npm, Less Compiler
Beliebt bei	Bootstrap, Foundation	Ant Design, Semantic UI
Compiler benötigt?	✅ Ja	✅ Ja
Syntaxstil	SCSS ist CSS-kompatibel	ähnlich zu CSS

> Hinweis: Im Unterricht und in Edube wird **Sass im SCSS-Syntax** bevorzugt.

---

3. Installation (Beispiel mit Sass)

a) Sass global installieren (via npm)

```
npm install -g sass
```

b) Kompilieren

```
sass styles.scss styles.css
```

> Alternativ: VS Code Plugin „Live Sass Compiler" benutzen (empfohlen für Einsteiger)

---

4. Grundsyntax (Sass / SCSS)

a) Variablen

```
$primary-color: #3498db;

body {
  background-color: $primary-color;
}
```

b) Verschachtelung (Nesting)

```
nav {
  ul {
    li {
      a {
```

```
        color: white;
      }
    }
  }
}
```

c) Partials & Import

```
// _buttons.scss
.button {
  padding: 10px;
}

// styles.scss
@import 'buttons';
```

> Sass „mergt“ Partials beim Kompilieren – ideal für große Projekte

d) Mixins (Funktionsähnlich)

```
@mixin flex-center {
  display: flex;
  justify-content: center;
  align-items: center;
}

.box {
  @include flex-center;
}
```

e) Vererbung mit `@extend`

```
%rounded {
  border-radius: 10px;
}

.card {
  @extend %rounded;
}
```

f) Operatoren

```
$base: 16px;

.container {
  padding: $base * 2;
}
```

---

5. Less – Basics im Vergleich

a) Variablen

```
@main-color: #ff6600;
```

```
body {
  background-color: @main-color;
}
```

b) Verschachtelung

```
nav {
  ul {
    li {
      a {
        color: white;
      }
    }
  }
}
```

c) Mixins

```
.flex-center() {
  display: flex;
  justify-content: center;
  align-items: center;
}

.box {
  .flex-center();
}
```

d) Import

```
@import "buttons.less";
```

---

6. Wann Sass oder Less einsetzen?

Sass wird bevorzugt, wenn:

- du SCSS als quasi-Standard in größeren Frameworks verwendest
- du modular arbeitest (Partials, Mixins, Funktionen)

Less eignet sich für:

- Legacy-Projekte mit Less-Abhängigkeit
- Frameworks, die nativ auf Less setzen (z. B. Ant Design)

---

7. Best Practices

- Benenne Dateien konsistent (`_nav.scss`, `_colors.scss`)
- Nutze sinnvolle Variablen für Farben, Abstände, Breakpoints
- Verwende Mixins für wiederkehrende Layouts
- Trenne Struktur (Layout) und Design (Farben, Effekte)
- Kompiliere regelmäßig und prüfe Output auf CSS-Ebene

---

8. CSS Custom Properties (`--var`) und :root – Konkurrenz oder Ergänzung zu Sass?

Seit 2015 gibt es in CSS die sogenannten **Custom Properties**, auch bekannt als **CSS-Variablen**. Diese erlauben eine native Variablenverwendung **ohne Kompiler**, direkt im Browser:

```
:root {
  --primary-color: #3498db;
  --spacing: 1.5rem;
}

.button {
  background-color: var(--primary-color);
  margin-bottom: var(--spacing);
}
```

Unterschiede zu Sass-Variablen

Feature	CSS `--var`	Sass / Less
Zur Laufzeit verfügbar	✅ (mit JavaScript änderbar)	❌ (nur beim Kompilieren)
Theming / Dark Mode	✅	✳️ möglich mit Aufwand
Strukturelle Logik	❌	✅ (Mixins, Funktionen, Schleifen)
Verbreitung / Reife	✅ (ab 2017 voll unterstützt)	✅ (seit Jahren Standard)

Wann CSS Variablen verwenden?

- Für **Farben, Spacing, Schriftgrößen**
- Für **responsive Breakpoints** (mit Media Queries kombinierbar)
- Für **dynamische Themes oder JavaScript-Interaktionen**

Kombinierbar mit Sass!

CSS Variablen lassen sich auch **aus SCSS heraus erzeugen**:

```
$primary: #3498db;

:root {
  --primary-color: #{$primary};
}
```

> So bekommst du das Beste aus beiden Welten: SCSS für Struktur, `--var` für Laufzeit.

Best Practice (2025-ready)

- Verwende **CSS Variablen** für dynamische, browsergesteuerte Werte
- Verwende **Sass** für strukturierte Stylesheets mit Mixins, Modulen, Funktionen
- **Kombiniere beide**, wenn du Flexibilität und Wartbarkeit maximieren willst

---

9. Übungsaufgaben

Aufgabe 1

Erstelle eine `main.scss`-Datei mit Variablen für:

- Primärfarbe
- Schriftgröße
- Containerbreite

Binde diese Variablen in ein einfaches Layout ein (Header, Footer, Content).

Aufgabe 2

Schreibe ein Mixin für ein flexibles Grid (ähnlich wie Flexbox) und wende es auf ein `section`-Element an.

Aufgabe 3

Baue eine Komponente `card.scss`, die `@extend` und Variablen verwendet.

Aufgabe 4

Stelle das Projekt so um, dass du alle Styles in Module zerlegst (z. B. `_layout.scss`, `_colors.scss`) und über `@import` zusammenfügst.
