

# LU02: Fonts, CSS-Variablen, Cascade & Media Queries

**Ziel von LU02:** Sie bringen Ihr Projekt „Landingpage Alarado“ optisch näher ans Figma-Design, indem Sie

1. die Schrift (Google Font) einbinden,
2. Designwerte als CSS-Variablen definieren und
3. mit Media Queries das Responsive-Layout für Mobile umsetzen.

## Lernziele

- Sie können eine Google Font per `<link>` einbinden und als `font-family` verwenden.
- Sie verstehen `px`, `em` und `rem` und setzen Schriftgroessen konsistent um.
- Sie können CSS-Variablen in `:root` definieren und mit `var(...)` einsetzen.
- Sie verstehen die CSS-**Kaskade** (Vererbung, Spezifität, Reihenfolge).
- Sie können Media Queries für Responsive Design nutzen und mit DevTools testen.

---

## 1) Fonts im Web



## Google Fonts via CDN

In vielen Projekten werden Fonts schnell über ein CDN eingebunden (z.B. Google Fonts). Das ist praktisch für Schulprojekte und schnelle Prototypen.

Typischer Aufbau im <head> (hier mit **Poppins**):

```
<link rel="preconnect" href="https://fonts.googleapis.com"
/>
<link rel="preconnect" href="https://fonts.gstatic.com"
crossorigin />
<link
  href="https://fonts.googleapis.com/css2?family=Poppins:ital,
  wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1
  ,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display
  =swap"
  rel="stylesheet"
/>
```

### Dann im CSS verwenden:

```
body {
  font-family: "Poppins", sans-serif;
}
```



**Praxis-Hinweis:** In professionellen Projekten werden Fonts oft **lokal** gespeichert (Datenschutz, Performance, Offline-Fähigkeit). Für dieses Modul genügt zu Beginn das Einbinden via <link>. <sup>1)</sup>

## FONTS LOKAL SPEICHERN

Wenn Fonts lokal im Projekt liegen, werden sie mit @font-face eingebunden.

```
@font-face {
  font-family: "Poppins";
  src: url("./fonts/Poppins-Regular.woff2") format("woff2");
  font-weight: 400;
  font-style: normal;
}
```

<sup>2)</sup>

## WICHTIGE FONT-EIGENSCHAFTEN

- font-family: Schriftfamilie mit Fallback (z.B. sans-serif)
- font-weight: Schriftschnitt (400 normal, 700 bold) <sup>3)</sup>
- font-style: normal oder italic
- text-transform: Darstellung (z.B. uppercase) <sup>4)</sup>
- line-height: Zeilenhöhe (Best Practice: **einheitenlos**, z.B. 1.4)

## EINHEITEN BEI SCHRIFTGROESSEN

**Wichtig:** Bei Schriftgroessen geht es nicht nur um „schön“, sondern auch um **Skalierung, Wartbarkeit und Accessibility**.

Viele Menschen nutzen Browser-Zoom oder haben eine grössere Standard-Schrift eingestellt.

## px

- px ist eine „fixe“ Design-Einheit: 16px bleibt rechnerisch 16px.
- Vorteil: direkt und einfach.
- Nachteil: In grossen Projekten wird es schnell unübersichtlich, weil Anpassungen an vielen Stellen nötig werden.

5)

```
h1 { font-size: 64px; }
p  { font-size: 18px; }
```

## em

- 1em entspricht der **berechneten Schriftgroesse** (font-size) des aktuellen Elements.
- em wirkt wie ein Faktor: 2em ist doppelt so gross wie die Schriftgroesse des Eltern-Elements.

6)

```
body { font-size: 16px; } /* Basis */
h1   { font-size: 2em; }  /* 2 × 16px = 32px */
```

**Die typische EM-Falle** Wenn mehrere Eltern-Elemente die Schrift bereits verändern, multipliziert sich der Effekt:

```
body { font-size: 1.2em; } /* 1.2 × 16px = 19.2px */
div  { font-size: 1.3em; } /* 1.3 × 19.2px = 24.96px */
h1   { font-size: 2em; }  /* 2 × 24.96px = 49.92px */
```

7)

## Wann ist em sinnvoll

- Für Proportionen innerhalb einer Komponente: Padding, Icon-Größen, Abstände, die mit dem Text mitwachsen sollen.

```
button {
  font-size: 1rem;
  padding: 0.8em 1.2em; /* wächst mit der Button-Schrift */
```

{

## rem

- 1rem ist relativ zur Schriftgroesse des Root-Elements html.
- Vorteil: bleibt konsistent, egal wie tief ein Element verschachtelt ist.

```
html { font-size: 16px; }
h1   { font-size: 4rem; }      /* 64px */
p    { font-size: 1.125rem; } /* 18px */
```

8)

## Empfehlung für dieses Modul

- Für Schriftgroessen überwiegend **rem** verwenden.
- **em** gezielt innerhalb von Komponenten einsetzen.
- **px** sparsam einsetzen, z.B. für 1px Linien oder sehr feine Details.

## line-height

Setzen Sie line-height möglichst **einheitenlos**, z.B.:

```
body { line-height: 1.4; }
```

9)

## 2) CSS-Variablen

### Was sind CSS-Variablen

CSS-Variablen sind benannte Werte, die Sie einmal zentral definieren und dann überall wiederverwenden.

```
:root {
  --primary: #263fa9;
```

```
--text: #223344;  
--h1-size: 4rem;  
}  
  
body { color: var(--text); }  
button { background: var(--primary); }  
h1 { font-size: var(--h1-size); }
```

## Warum lohnt sich das

- **Wiederverwendbarkeit:** weniger Copy/Paste, weniger Fehler
- **Wartbarkeit:** 1 Änderung in :root wirkt im ganzen Projekt
- **Lesbarkeit:** Namen sind verständlicher als zufällige Hex-Zahlen

### Vergleich zu Print-Design

CSS-Variablen funktionieren ähnlich wie Absatz- und Zeichenstile: Sie definieren Werte zentral und wenden sie konsistent an.

---

## 3) CSS Cascade



Viele CSS-Regeln funktionieren nach dem Prinzip „Wasserfall“ → Styles, die für Elemente ganz „oben“ im HTML definiert wurden, vererben sich an Elemente weiter „unten“ in der HTML-Struktur (gemeint sind Kinder-Elemente)

### Was ist die Kaskade

Die CSS-Kaskade entscheidet, welche CSS-Regel am Ende wirklich angewendet wird, wenn mehrere Regeln dasselbe Element betreffen.

## Vererbung

Einige Eigenschaften werden von Eltern an Kinder **vererbt**.

- Häufig vererbt: color, font-\*, line-height, text-transform
- Meist nicht vererbt: background-\*, margin/padding, border, width/height, display

```
body {  
    font-family: "Poppins", sans-serif;  
    color: #223344;  
    line-height: 1.4;  
}
```

## Überschreiben

Wenn zwei Regeln dieselbe Eigenschaft setzen, gilt diese Reihenfolge:

1. **Wichtigkeit:** !important gewinnt <sup>10)</sup>
2. **Spezifität:** ID schlägt Klasse, Klasse schlägt Element
3. **Reihenfolge:** bei gleicher Spezifität gewinnt die spätere Regel

```
a { color: black; }  
.menu a { color: gray; }  
.active { color: blue; }  
#special a { color: red; }
```

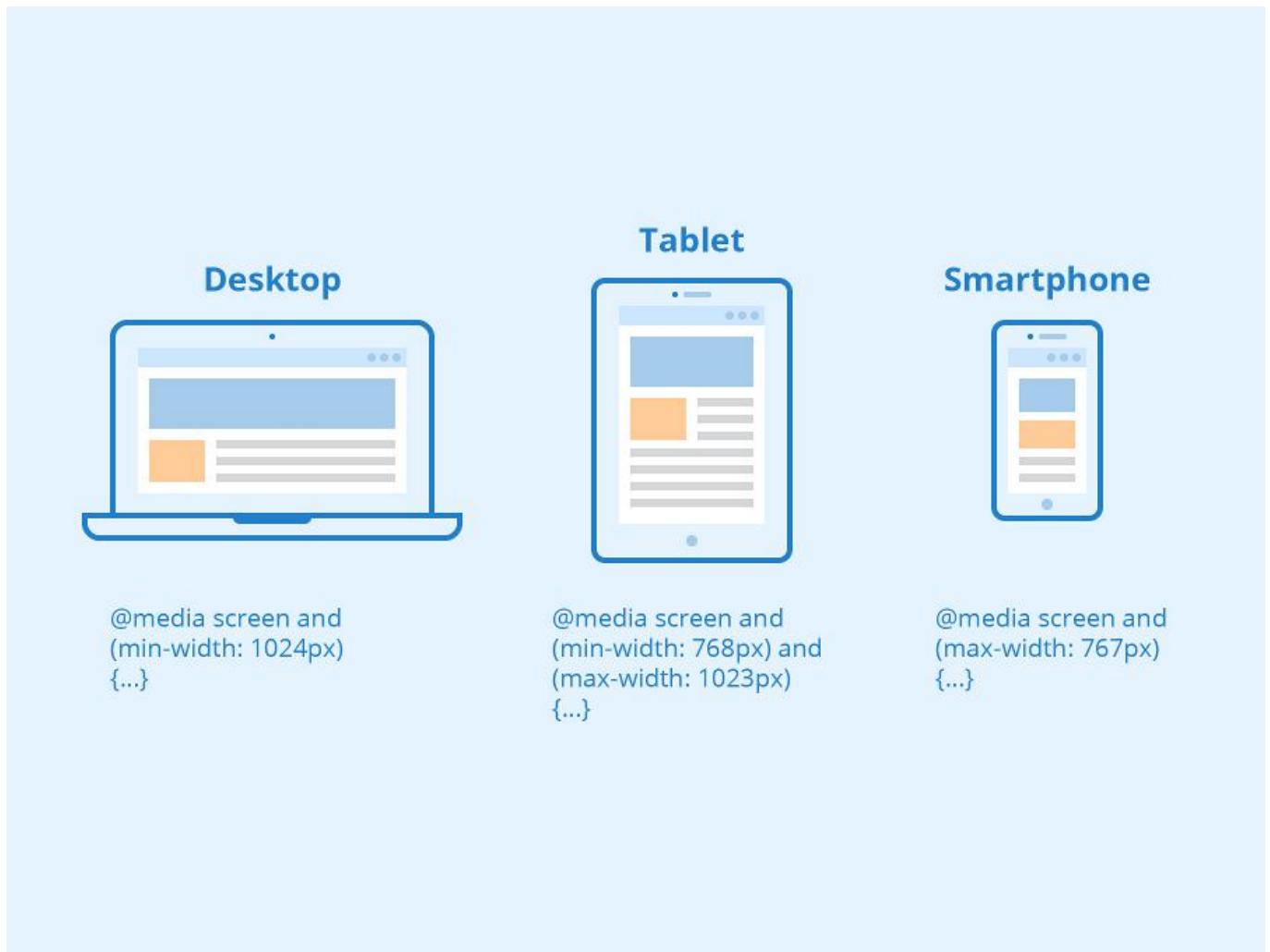
## Reihenfolge im CSS

Bei gleicher Spezifität gewinnt die Regel, die später im CSS steht.

```
.menu a { color: gray; }  
.menu a { color: green; } /* gewinnt, weil später */
```

Für ein vertieftes Verständnis schauen Sie sich diese interaktive Darstellung an: [CSS Cascade](#)

## 4) Media Queries und Responsive Design



### Was macht eine Media Query

Media Queries passen die Darstellung an verschiedene Bildschirmbreiten an. Der Inhalt bleibt gleich, nur das Layout ändert sich.

### Breakpoints

Ein Breakpoint ist eine Breite, bei der Sie das Layout anpassen, weil es sonst nicht mehr sauber wirkt.

```
@media (width < 1024px) { ... }
```

Typische Hinweise, dass ein Breakpoint nötig ist:

- Elemente überlappen oder stossen sich
- Navigation passt nicht mehr

- 2 Spalten werden zu eng
- horizontales Scrollen entsteht



**Merksatz:** Ein Breakpoint ist ein Layout-Umbruchpunkt, nicht ein „Handy-Mass“.

## Beispiel aus unserem Projekt

Desktop: Text links, Bild rechts. Mobile: Bild oben, Text darunter.

```
#hero-section { flex-direction: row; }

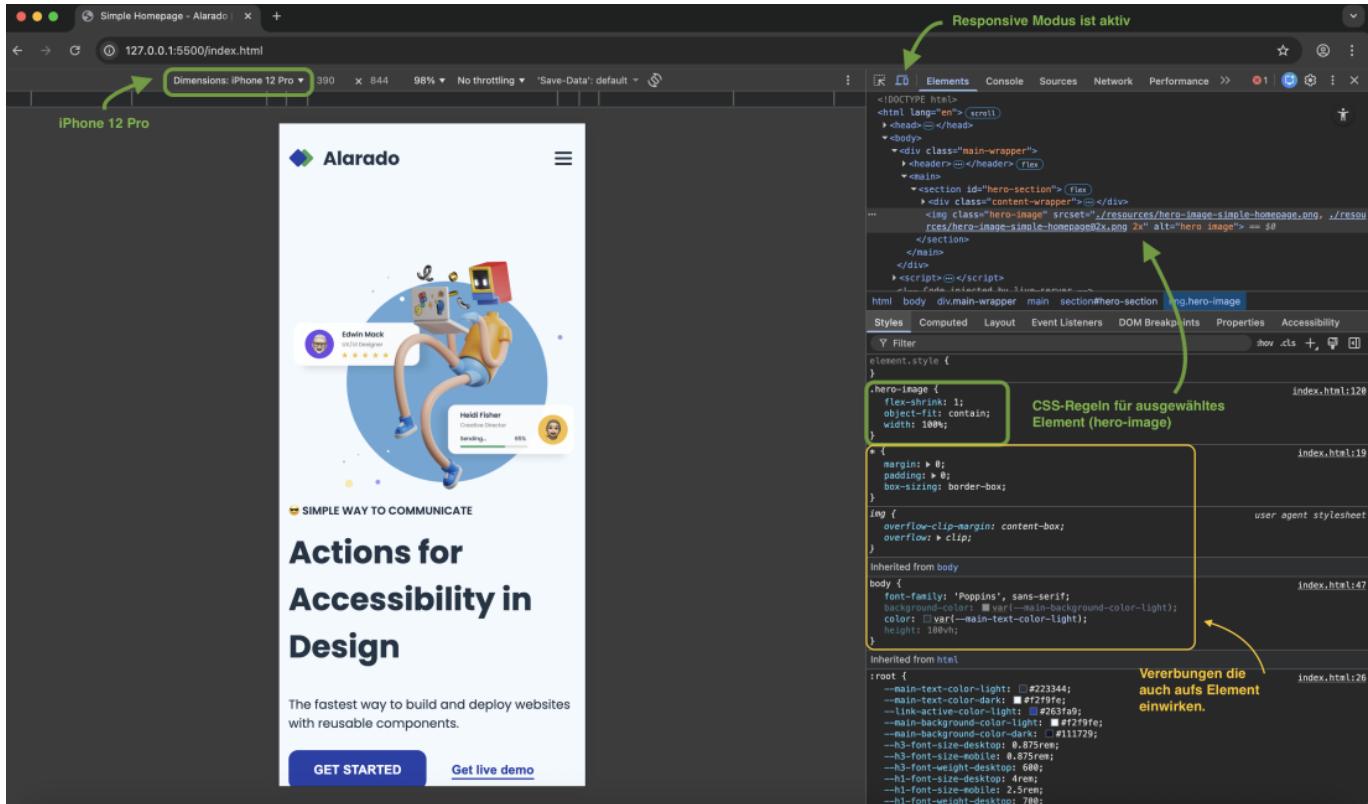
@media screen and (width < 1024px) {
    #hero-section { flex-direction: column-reverse; }
}
```

## Media Queries und Kaskade

Media Queries folgen denselben Regeln wie normales CSS:

- Bei aktivem Breakpoint überschreiben Sie nur die Eigenschaften, die sich ändern.
- Bei gleicher Spezifität gewinnt die Regel, die später kommt.

## 5) Responsive Testing mit DevTools



- Öffnen Sie DevTools (Chrome/Edge: F12)
- Aktivieren Sie Device Toolbar (Handy/Tablet Symbol)
- Testen Sie:
  1. Breitenwechsel über/unter 1024px
  2. Presets (z.B. iPhone, iPad)
  3. Hoch-/Querformat
- Prüfen Sie:
  1. Überlappungen oder abgeschnittene Bereiche
  2. horizontales Scrollen
  3. Lesbarkeit der Schrift

### Praxis-Tipp

Wenn etwas nicht greift: DevTools → „Styles“ öffnen und schauen, welche Regel durchgestrichen ist. Das ist Kaskade in Aktion.

<sup>1)</sup>

CDN-Fonts sind bequem, aber Sie sind abhängig von einem externen Dienst.

<sup>2)</sup>

Für lokale Fonts brauchen Sie die Font-Dateien im Projekt, idealerweise woff2.

<sup>3)</sup>

Nicht jede Font bietet jede Weight-Stufe.

<sup>4)</sup>

Der Inhalt bleibt gleich, nur die Darstellung ändert sich.

<sup>5)</sup>

Browser-Zoom vergrößert zwar alles. Trotzdem ist px oft weniger flexibel, wenn Sie ein Designsystem skalieren oder mehrere Breakpoints sauber pflegen wollen.

<sup>6)</sup>

Historisch stammt „em“ aus der Typografie. In CSS ist em eine Recheneinheit: 1em = aktuelle font-

Last update:

2026/02/02 de:modul:m291:learningunits:lu02:theorie:a\_css-intro https://wiki.bzz.ch/de/modul/m291/learningunits/lu02/theorie/a\_css-intro  
01:29

---

size.

7)

Darum ist es für globale Typografie manchmal schwerer zu kontrollieren, wenn viel verschachtelt wird.

8)

Mit `rem` vermeiden Sie „Schriftgroessenvermehrung“ durch verschachtelte Elemente.

9)

Einheitenlose `line-height` skaliert automatisch mit der Schriftgroesse.

10)

Nur im Ausnahmefall verwenden.

From:

<https://wiki.bzz.ch/> - BZZ - Modulwiki



Permanent link:

[https://wiki.bzz.ch/de/modul/m291/learningunits/lu02/theorie/a\\_css-intro](https://wiki.bzz.ch/de/modul/m291/learningunits/lu02/theorie/a_css-intro)

Last update: **2026/02/02 01:29**