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) ein erstes responsives Verhalten via Media Query umsetzen.

Lernziele

- Sie können eine Google Font per `<link>` einbinden und sinnvoll als `font-family` verwenden.
- Sie verstehen den Unterschied zwischen `px`, `em` und `rem` und setzen Schriftgrössen konsistent um.
- Sie können CSS-Variablen in `:root` definieren und mit `var(…)` wiederverwenden.
- Sie verstehen die CSS-**Cascade** (Reihenfolge/Überschreiben/Vererbung).
- Sie können Media Queries für Responsive Design nutzen und mit DevTools testen.

A) Fonts im Web

A1) Schnellstart: Google Fonts via CDN

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

Typischer Aufbau im <head> (Performance-Tipp: `preconnect`):

```
<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:wght@400;500;600;700&
display=swap" rel="stylesheet" />
```

Dann im CSS verwenden:

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

Hinweis zur Praxis: In professionellen Projekten werden Fonts häufig **lokal** im Projekt gespeichert (Datenschutz/Performance/Offline-Fähigkeit). Für dieses

Modul genügt zu Beginn das Einbinden via `<link>`.

A2) Besser in der Praxis: Fonts lokal speichern (kurz)

Mit `@font-face` können Sie eine Schrift von Ihrer eigenen Projekt-URL laden und einen Namen vergeben. :contentReference[oaicite:0]{index=0}

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

A3) Wichtige Font-Eigenschaften (Repetition)

- `font-family`: Schrift-Familie (immer mit Fallbacks arbeiten, z.B. `sans-serif`)
- `font-weight`: Dicke (z.B. 400 normal, 700 bold – nicht jede Schrift hat alle Stufen) :contentReference[oaicite:1]{index=1}
- `font-style`: normal/italic (oblique selten sinnvoll) :contentReference[oaicite:2]{index=2}
- `text-transform`: Darstellung (uppercase/lowercase/capitalize) – gehört zur Präsentation, nicht zum Inhalt :contentReference[oaicite:3]{index=3}
- `line-height`: Zeilenhöhe (Best Practice: **ohne Einheit**, z.B. 1.4)

A4) Einheiten bei Fonts: px vs em vs rem

- **px** = absolute Einheit (kann bei Accessibility/Zoom/Default-Einstellungen unflexibler sein)
- **em** = relativ zur Schriftgrösse des *Eltern-Elements*
- **rem** = relativ zur Schriftgrösse des `<html>` (Root) – dadurch konsistenter im ganzen Projekt :contentReference[oaicite:4]{index=4}

Best Practice (für dieses Modul):
- Nutzen Sie für Schriftgrössen bevorzugt **rem**.
- Nutzen Sie für `line-height` einen **einheitenlosen Wert** (z.B. 1.4).

B) CSS-Variablen (Custom Properties)

B1) Was sind CSS-Variablen?

CSS-Variablen (korrekt: Custom Properties) sind benannte Werte, die Sie **einmal zentral** definieren und dann überall mit `var(…)` verwenden. Das spart Copy/Paste und macht Anpassungen viel einfacher. :contentReference[oaicite:5]{index=5}

Beispiel:

```
:root {
  --primary: #263fa9;
  --text: #223344;
}
body {
  color: var(--text);
}
button {
  background: var(--primary);
}
```

B2) Warum ist das nützlich?

- **Wiederverwendbarkeit**: Farben/Grössen müssen nicht überall kopiert werden. :contentReference[oaicite:6]{index=6}
- **Wartbarkeit**: Eine Änderung in `:root` wirkt im ganzen Projekt. :contentReference[oaicite:7]{index=7}
- **Semantik**: –`main-text-color` ist verständlicher als #223344. :contentReference[oaicite:8]{index=8}
- **Reagiert auf Media Queries und DOM** (im Gegensatz zu reinen Preprocessor-Variablen). :contentReference[oaicite:9]{index=9}

> **Vergleich zu Print-Design (z.B. InDesign):**
> CSS-Variablen sind ähnlich wie **Absatz-/Zeichenstile** oder ein **Designsystem**: Sie definieren Regeln einmal zentral und wenden sie überall an.

C) CSS Cascade (Kaskade) – kurz & wichtig

Die Kaskade bestimmt, **welche CSS-Regel am Ende gewinnt**.

C1) Grundprinzip

- Viele Eigenschaften werden **vererbt** (z.B. `color`, `font-*`, `line-height`, `text-transform`). :contentReference[oaicite:10]{index=10}

- Andere werden **nicht vererbt** (z.B. viele `background-*` Eigenschaften). :contentReference[oaicite:11]{index=11}
- Wenn Sie eine Eigenschaft «weiter innen» oder «später» neu setzen, **überschreiben** Sie nur diese Eigenschaft – der Rest wird weiter vererbt. :contentReference[oaicite:12]{index=12}

Mini-Beispiel:

```
body { color: #223344; }
p { color: #223344; }
p span { color: #263fa9; } /* überschreibt nur die Farbe im span */
```

C2) Reihenfolge, die Sie sich merken sollen

1. Spezifischere Selektoren gewinnen meist gegen allgemeine.
2. Bei gleicher Spezifität gewinnt die **Regel, die später im CSS steht**.
3. Media Queries wirken wie „zusätzliche Regeln“, die nur aktiv sind, wenn die Bedingung stimmt.

D) Media Queries & Responsive Design

D1) Was macht eine Media Query?

Media Queries erlauben, dass sich das Layout an verschiedene Bildschirmgrössen anpasst, ohne den Inhalt zu ändern – nur die Darstellung wird angepasst. :contentReference[oaicite:13]{index=13}

D2) Breakpoints: nach Inhalt, nicht nach Geräten

Setzen Sie Breakpoints dann, wenn Ihr Layout **„kaputt“** aussieht – nicht weil ein bestimmtes Handy angeblich 390px breit ist. :contentReference[oaicite:14]{index=14}

D3) Beispiel aus unserem Projekt

Unser Desktop-Layout ist 2-spaltig (Text links, Bild rechts). Auf Mobile wird es 1-spaltig, mit dem Bild zuerst:

```
@media screen and (width < 1024px) {
  #hero-section {
    flex-direction: column-reverse; /* Bild oben, Text unten */
  }
}
```

D4) Media Queries und Kaskade

- Innerhalb der Media Query überschreiben Sie **nur** die Eigenschaften, die sich ändern sollen.
- Wenn ein Selektor gleich bleibt (z.B. `#hero-section`), gewinnt bei aktiver Media Query die Regel, die **später** kommt (also im `@media`-Block).

E) Responsive Testing mit DevTools

- Öffnen Sie DevTools (Chrome/Edge: F12)
- Aktivieren Sie den **Device Toolbar** (Handy/Tablet Symbol)
- Testen Sie:
  1. Breitenwechsel über/unter **1024px**
  2. verschiedene Presets (z.B. iPhone, iPad)
  3. Landscape/Portrait
- Prüfen Sie auch:
  1. ob Elemente verschwinden/überlappen
  2. ob Text zu klein/gross wirkt
  3. ob es horizontales Scrollen gibt (meist ein Zeichen für zu grosse Breiten)