LU05b – CSS Transitions

CSS Transitions – Grundprinzip

Eine CSS Transition animiert den Übergang zwischen zwei Zuständen eines Elements. Sie definieren, **welche Eigenschaft** animiert wird, **wie lange** die Animation dauert und nach welcher **Timing-Kurve**.

```
.element {
  background-color: blue;
  transition: background-color 400ms ease-in-out;
}

.element:hover {
  background-color: red;
}
```

Beim Hovern wechselt die Hintergrundfarbe nun nicht abrupt, sondern gleitend über 400 ms von Blau zu Rot.

Mehrere Eigenschaften gleichzeitig animieren:

```
.panel {
  opacity: 0;
  height: 0;
  transition:
    opacity  600ms ease-in-out,
    height   600ms ease-in-out;
}
```

Warum reicht display: none nicht?

Im ersten Schritt haben wir die Panels mit `display: none` / `display: block` ein- und ausgeblendet. Das funktioniert – aber es gibt **keinen Übergang**, das Element erscheint und verschwindet abrupt.

Warum lässt sich `display` nicht animieren? CSS Transitions interpolieren zwischen zwei Zahlenwerten. `display: none` bedeutet: das Element **existiert im Layout nicht**. Zwischen „existiert nicht“ und „existiert“ kann kein Mittelwert berechnet werden – eine Transition ist daher logisch nicht möglich.

```
/* ❌ Kein Übergang möglich */
.panel {
  display: none;
  transition: display 400ms; /* wirkungslos */
}
.panel.open {
  display: block; /* schaltet sofort um */
}
```

Die klassischen Workarounds

Bis vor Kurzem gab es nur unbefriedigende Lösungen für sanfte Auf-/Zuklapp-Animationen:

Workaround 1: opacity + visibility

`opacity` lässt sich animieren, aber das Element nimmt weiterhin Platz im Layout ein.

Workaround 2: JavaScript

Die tatsächliche Höhe per JavaScript berechnen (`element.scrollHeight`) und explizit als Pixel-Wert setzen. Mehr Code, mehr Komplexität, schwerer wartbar.

Das Problem: height von 0 zu auto

Der einfachste und natürlichste Ansatz wäre: `height: 0` → `height: auto` (→ auto: der Browser berechnet selbst die Höhe des Elements anhand der Inhalte). Leider funktioniert das **nicht** mit einer Transition:

```
/* ❌ Funktioniert NICHT – Browser kann nicht interpolieren
*/
.panel {
  height: 0;
  overflow: hidden;
  transition: height 600ms ease-in-out;
}
.panel.open {
  height: auto; /* Browser weiss nicht, was das in px
bedeutet */
```

```
}
```

Der Browser kann zwischen einem fixen Pixelwert (`0`) und dem Schlüsselwort `auto` nicht interpolieren – `auto` ist kein Zahlenwert, sondern eine Anweisung: „berechne die Höhe selbst“.

Die moderne Lösung: interpolate-size: allow-keywords

Seit 2024 unterstützen moderne Browser eine neue CSS-Eigenschaft, die genau dieses Problem löst: `interpolate-size: allow-keywords`.

Sie erlaubt dem Browser, Übergänge zwischen einem Pixel-Wert und einem CSS-Schlüsselwort wie `auto`, `min-content` oder `max-content` zu berechnen.

```
:root {
  interpolate-size: allow-keywords; /* einmal global
definieren */
}
```

Damit funktioniert `height: 0` → `height: auto` mit einer Transition wie erwartet.

CSS-Transitions: Ein- und Ausblenden der Antworten

```
:root {
  interpolate-size: allow-keywords;
}

/* Standardmässig versteckt */
.panel {
  height: 0;
  opacity: 0;
  overflow: hidden;
  transition:
    height  800ms ease-in-out,
    opacity 800ms ease-in-out;
}

/* Sichtbar wenn Klasse 'open' gesetzt */
.panel.open {
  height: auto;
  opacity: 1;
  overflow: clip;
}
```

Das Öffnen und Schliessen wird weiterhin per JavaScript durch `classList.add('open')` / `classList.remove('open')` gesteuert – die Animation läuft vollständig in CSS.

overflow: hidden vs. overflow: clip

Im geschlossenen Zustand verwenden wir `overflow: hidden`, im offenen `overflow: clip`. Der Unterschied:

- `overflow: hidden` erzeugt einen neuen Scroll-Container (Block Formatting Context), was in manchen Layouts zu unerwünschten Nebeneffekten führen kann.
- `overflow: clip` schneidet Inhalte ab, ohne einen Scroll-Container zu erzeugen – und harmoniert besser mit modernen Transitions.

Neueste Entwicklung: @starting-style

In sehr modernen Browsern gibt es eine noch elegantere Variante mit `@starting-style` und `display allow-discrete`. Diese Technik erlaubt Transitionen auch beim **ersten Erscheinen** eines Elements, also direkt von `display: none` zu `display: block`:

```
:root {
  interpolate-size: allow-keywords;
}

/* Startzustand beim ersten Einblenden */
@starting-style {
  .panel {
    opacity: 0;
    height: 0;
  }
}

/* Standardmässig versteckt */
.panel {
  height: 0;
  overflow: hidden;
  opacity: 0;
  transition:
    height  1100ms ease-in-out,
    opacity 1100ms ease-in-out,
    display 0.5s allow-discrete;
}

/* Sichtbar wenn Klasse 'open' gesetzt */
.panel.open {
  display: block;
  height: auto;
  opacity: 1;
```

```
}
```

Browser-Support: `@starting-style` und `display allow-discrete` sind noch **nicht von allen Browsern unterstützt** (Stand 2026: kein Firefox-Support). Für produktive Projekte wird die stabilere Lösung mit `interpolate-size` und `height: 0 → auto` empfohlen, die im Accordion-Projekt eingesetzt wird.

Aktuellen Support prüfen: caniuse.com – @starting-style

Timing-Funktionen

Der dritte Parameter von `transition` steuert die Beschleunigungskurve der Animation:

Wert	Beschreibung
`ease`	Langsam starten, beschleunigen, langsam enden (Standard)
`ease-in`	Langsam starten, schnell enden
`ease-out`	Schnell starten, langsam enden – oft natürlicher
`ease-in-out`	Langsam starten und enden, in der Mitte schnell
`linear`	Konstante Geschwindigkeit
`cubic-bezier(x,x,x,x)`	Vollständig selbst definierte Kurve

Im Accordion-Projekt verwenden wir `ease-in-out` – das Panel öffnet und schliesst sich jeweils mit einer leichten Verzögerung am Anfang und Ende, was natürlicher wirkt.