Modul 293 - Lösungsblatt 08

von Lukas Meier

Unterricht vom 10.02.2026

Lösungsblatt: Repetition HTML und CSS

Thema 1: HTML Grundlagen

Lösung 1.1 – Begriffe sicher unterscheiden

- Ein **HTML-Tag** ist eine Markierung in spitzen Klammern, die dem Browser sagt, welche Rolle ein Inhalt hat, z. B. `<p>` für einen Absatz.
- Ein **Attribut** liefert zusätzliche Informationen zu einem Tag, z. B. `href` bei einem Link oder `alt` bei einem Bild.
- **Start-Tag**: öffnet das Element (z. B. `<h1>`), **End-Tag**: schliesst es (`</h1>`), **Inhalt**: steht dazwischen (z. B. `Titel`).

Beispiel:

```
<h1 class="titel">Willkommen</h1>
```

Tag: `h1`, Attribut: `class="titel"`, Inhalt: `Willkommen`.

Lösung 1.2 – Tags erkennen und einordnen

Code:

```
<a href="https://example.com" target="_blank">Zur Website</a>
<img src="bild.jpg" alt="Profilbild">
<section class="intro">Willkommen</section>
```

- Tags: `<a> ... </a>`, `<img>`, `<section> ... </section>`.
- Attribute: `href`, `target`, `src`, `alt`, `class`.

- Selbstschliessendes Element in diesem Ausschnitt: `<img>`.

Lösung 1.3 – Selbstschliessende Tags

Eine mögliche Lösung (nur Body-Inhalt):

```
<h1>Mein Lerntagebuch</h1>
<p>Heute wiederhole ich HTML und CSS.<br />Ich übe Schritt für Schritt.</p>
<img src="lernen.jpg" alt="Schreibtisch mit Laptop und Notizen">
<hr />
```

Selbstschliessende Tags hier: `<br />`, `<img />`, `<hr />`.

Lösung 1.4 – Semantik verbessern

Eine mögliche semantische Struktur:

```
<header>
  <h1>Meine Lernseite</h1>
</header>
<nav>
  <a href="#">Start</a>
  <a href="#">Kontakt</a>
</nav>
<main>
  <section>
    <p>Heute üben wir HTML.</p>
  </section>
</main>
```

Lösung 1.5 – Fehleranalyse

Korrigierter Code:

```
<section>
  <h1 class="titel">Wochenplan</h1>
  <p id="text" class="wichtig">Montag: CSS wiederholen</p>
  <img src="plan.png" alt="Wochenplan als Bild" />
</section>
```

Vier Fehler und Begründung:

- `<h1>` wurde nicht geschlossen: Überschriften mit Inhalt brauchen ein End-Tag.

- `"wichtig"` war ein ungültiges freies Attributfragment: für Klassennamen braucht es `class="..."`.
- Bei `<img>` war `source` falsch: korrekt ist `src`.
- Der Selbtschliessende `img`-Tag wurde am Ende nicht mit einem `/` geschlossen.

Lösung 1.6 – Transferaufgabe HTML

Eine vollständige Mini-Seite:

```
<body>
  <header>
    <h1>Lektion 08 Repetition</h1>
  </header>

  <nav>
    <a href="thema1.html">Thema 1</a>
    <a href="thema2.html">Thema 2</a>
    <a href="kontakt.html">Kontakt</a>
  </nav>

  <main>
    <section id="thema1">
      <h2>HTML</h2>
      <p>Hier wiederholen wir Tags und Attribute.</p>
      <hr>
    </section>

    <section id="thema2">
      <h2>CSS</h2>
      <p>Hier wiederholen wir Selektoren und Layout.</p>
      <img src="css.png" alt="CSS Logo">
    </section>
  </main>

  <footer id="kontakt">
    <p>copyright 2026 Modul 293</p>
  </footer>
</body>
```

Thema 2: CSS Selektoren

Lösung 2.1 – Grundprinzipien

- Tag-Selektor: direkt mit dem Tag-Namen, z. B. `p { color: blue; }`.
- Klassen-Selektor: mit Punkt, z. B. `.info { color: green; }`.
- ID-Selektor: mit Raute, z. B. `#title { color: red; }`.

Mini-Beispiel:

```
<h1 id="title">Titel</h1>
<p class="info">Text</p>

h1 { font-size: 32px; }
.info { color: green; }
#title { text-transform: uppercase; }
```

Lösung 2.2 – Selektoren zuordnen

Gegebenes HTML:

```
<h1 id="title">Modul 293</h1>
<p class="info">Einführung</p>
<p class="info">Repetition</p>
<button class="info" id="start-btn">Start</button>
```

Selektierte Elemente:

- `p` selektiert die beiden Absatz-Elemente.
- `.info` selektiert beide `p` und den `button`.
- `#title` selektiert nur das `h1`-Element.
- `button.info` selektiert nur den `button` mit Klasse `info`.

Lösung 2.3 – Mehrere Elemente in einem Selektor

Mögliche CSS-Lösung:

```
h1, h2, h3 {
  color: #1f4e79;
}

p, li {
```

```
  font-size: 18px;
}

.note, #warning {
  background-color: #fff3cd;
}
```

Lösung 2.4 – Kombinierte Selektoren

Mögliche CSS-Regeln:

```
p.info {
  color: blue;
}

#menu {
  border: 2px solid #333;
}

.nav li {
  font-weight: bold;
}
```

Warum `.info` zu ungenau ist:

- `.info` trifft alle Elemente mit dieser Klasse (z. B. `p`, `div`, `button`).
- Mit `p.info` wird gezielt nur der Absatz mit dieser Klasse angesprochen.

Lösung 2.5 – Fehler finden und korrigieren

Korrigierte Variante:

```
#title {
  color: red;
}

.intro {
  font-size: 20px;
}

p.hinweis, #wichtig {
  background: yellow;
```

```
}
```

Kurzbegründung pro Block:

- `title` würde das HTML-`<title>`-Element meinen; gemeint war hier typischerweise die ID `title`.
- `#.intro` ist ungültig, da `#` und `.` nicht gleichzeitig für denselben einfachen Selektor stehen.
- `p, .hinweis #wichtig` Logikfehler! Id's sollen immer nur einmal im HTML vorkommen. Ein Selektor für alle Elemente mit der ID wichtig in allen Klassen hinweis macht also keinen Sinn.

Lösung 2.6 – Transferaufgabe Selektoren

Möglicher HTML-Ausschnitt:

```
<h1 id="page-title">Repetition</h1>
<h2 class="subtitle">Selektoren</h2>

<p class="info">Absatz 1</p>
<p>Absatz 2</p>
<p class="info important">Absatz 3</p>

<ul class="nav">
  <li>Home</li>
  <li>HTML</li>
  <li>CSS</li>
  <li>Kontakt</li>
</ul>

<button id="start">Start</button>
```

Passendes CSS:

```
/* Tag-Selektor */
p {
  line-height: 1.5;
}

/* Klassen-Selektoren */
.info {
  color: #0a4;
```

```
}

.subtitle {
  letter-spacing: 0.5px;
}

/* ID-Selektor */
#start {
  border: 1px solid #333;
}

/* Gruppierte Selektoren */
h1, h2 {
  font-family: Arial, sans-serif;
}

ul, p {
  margin-bottom: 12px;
}

/* Kombinierter Selektor */
p.info {
  font-weight: 600;
}
```

Thema 3: CSS Box-Modell

Lösung 3.1 – Theorie verstehen

Reihenfolge von innen nach aussen:

- **Content** (Inhalt)
- **Padding** (Innenabstand)
- **Border** (Rahmen)
- **Margin** (Aussenabstand)

Erklärungen:

- `padding` ist der Abstand zwischen Inhalt und Rahmen.

- `margin` ist der Abstand eines Elements nach aussen zu anderen Elementen.
- Verwendete Properties: `padding`, `padding-top/right/bottom/left`, `margin`, `margin-top/right/bottom/left`.

Lösung 3.2 – Box aufbauen

CSS:

```
.card {
  width: 280px;
  padding: 16px;
  border: 2px solid black;
  margin: 20px;
}
```

Skizze (textuell):

- Mitte: Inhaltstext
- Um den Inhalt: 16px Padding
- Danach: 2px Border
- Aussen: 20px Margin zur Umgebung

Lösung 3.3 – Margin vs. Padding gezielt testen

Beobachtung:

- Wenn nur `padding` grösser wird, wächst der innere Abstand zwischen Inhalt und Rahmen; die Box wirkt innen luftiger.
- Wenn nur `margin` grösser wird, bleibt das Innere gleich, aber der Abstand zu anderen Elementen wird grösser.
- `padding` beeinflusst den Innenraum der Box, `margin` den Aussenraum im Layout.

Lösung 3.4 – Block und Inline vergleichen

- `div` ist standardmässig `display: block`.
- `span` ist standardmässig `display: inline`.

- Bei `span` wirken `width`, `height` und vertikale `margin` in der Regel nicht wie bei Block-Elementen.
- Mit `display: inline-block` akzeptiert `span` Breite/Höhe und bleibt trotzdem im Textfluss.

Lösung 3.5 – Rechnen mit dem Box-Modell

Gegeben: `width: 300px`, links/rechts `padding: 20px`, links/rechts `border: 3px`.

content-box:

- Gesamtbreite = $300 + 20 + 20 + 3 + 3 = 346$px

border-box:

- Gesamtbreite = 300px (Padding und Border sind in den 300px enthalten)

Unterschied: Bei `content-box` kommt Padding/Border zur Breite dazu, bei `border-box` nicht.

Lösung 3.6 – Transferaufgabe Box-Modell

Mögliche Lösung:

```
<section class="cards">
  <article class="card card-a">
    <h3>Karte A</h3>
    <p>Mehr Padding, kleine Margin.</p>
  </article>

  <article class="card card-b">
    <h3>Karte B</h3>
    <p>Kleine Padding, grosse Margin.</p>
  </article>

  <article class="card card-c">
    <h3>Karte C</h3>
    <p>Inline-block Karte.</p>
  </article>
</section>
```

```
.card {
```

```
  box-sizing: border-box;
  width: 260px;
  border: 2px solid #222;
}

.card-a {
  padding: 24px;
  margin: 8px;
}

.card-b {
  padding: 10px;
  margin: 24px;
}

.card-c {
  padding: 16px;
  margin: 12px;
  display: inline-block;
}
```

Kurze Dokumentation:

- Die Karten bleiben übersichtlich, weil `box-sizing: border-box` die Breite stabil hält.
- Unterschiedliche Margin/Padding-Werte verändern gezielt den Innen- und Aussenabstand.

Thema 4: CSS Positionierung

Lösung 4.1 – Positionierungsarten beschreiben

- `static`: Standard, Element bleibt normal im Dokumentfluss, `top/right/bottom/left` wirken nicht.
- `relative`: Element bleibt im Fluss, kann aber visuell relativ zu seiner Ausgangsposition verschoben werden.
- `absolute`: Element wird aus dem normalen Fluss genommen und relativ zum nächsten positionierten Vorfahren platziert.
- `fixed`: Element wird relativ zum Viewport fixiert und bleibt beim

Scrollen sichtbar.

Einsatzbeispiele:

- `static`: normale Text- und Inhaltsblöcke
- `relative`: kleine Korrekturpositionen bei Badges/Labels
- `absolute`: Icons oder Marker in einer Karte
- `fixed`: Sticky-Topbar oder dauerhaft sichtbarer Hilfe-Button

Lösung 4.2 – Relative Positionierung anwenden

Beispiel:

```
.container {
  width: 400px;
  border: 1px solid #888;
}

.box {
  position: relative;
  top: 10px;
  left: 20px;
}
```

Beschreibung:

- Verschoben wird vom ursprünglichen Platz der Box aus.
- Der ursprüngliche Platz im Dokumentfluss bleibt reserviert.

Lösung 4.3 – Absolute Positionierung

Beispiel:

```
.container {
  position: relative;
  width: 500px;
  height: 220px;
  border: 2px solid #333;
}

.marker-a {
  position: absolute;
```

```
  top: 10px;
  left: 10px;
}

.marker-b {
  position: absolute;
  bottom: 10px;
  right: 10px;
}
```

Unterschied beim Entfernen von `position: relative` am Container:

- Die Marker orientieren sich nicht mehr am Container, sondern am nächsten anderen positionierten Vorfahren (oder am Viewport).

Lösung 4.4 – Fixed und Scrollverhalten

Beispiel:

```
.notice {
  position: fixed;
  top: 12px;
  right: 12px;
}
```

Analyse:

- Das Element bleibt beim Scrollen sichtbar an derselben Stelle im Fenster.
- Es verbessert die Sichtbarkeit wichtiger Infos, darf aber Inhalte nicht verdecken.

Lösung 4.5 – Display und Dokumentfluss

Beobachtung:

- `display: block`: jedes Element startet in neuer Zeile, nimmt standardmässig volle Breite ein.
- `display: inline`: Elemente liegen in einer Zeile, Breite/Höhe nur eingeschränkt steuerbar.
- `display: inline-block`: Elemente bleiben in einer Zeile, Breite/Höhe sind aber steuerbar.

- `position: relative`: Element bleibt im Fluss, nur visuell verschoben.
- `position: absolute`: Element verlässt den normalen Fluss und überlagert ggf. andere Elemente.

Lösung 4.6 – Transferaufgabe Positionierung

Mögliche Mini-Webseite:

```
<header class="topbar">Modul 293</header>

<main class="page-content">
  <section class="board">
    <span class="pin pin-a">A</span>
    <span class="pin pin-b">B</span>
    <article class="text-block">
      Dieses Element bleibt normal im Dokumentfluss.
    </article>
  </section>
</main>
```

```
.topbar {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  height: 52px;
}

.page-content {
  margin-top: 70px;
}

.board {
  position: relative;
  min-height: 220px;
  border: 2px solid #444;
}

.pin {
  position: absolute;
  width: 28px;
```

```
  height: 28px;
  display: inline-block;
  text-align: center;
}

.pin-a { top: 12px; left: 12px; }
.pin-b { bottom: 12px; right: 12px; }
```

Reflexion (Beispiel):

- Absolute Positionierung ist sinnvoll für dekorative Marker, Overlays und exakt platzierte UI-Elemente.
- Sie sollte vermieden werden, wenn Inhalte dynamisch wachsen oder stark responsiv sein müssen.
- Zu viele absolute Elemente führen schnell zu überlappenden Inhalten.
- `display` steuert weiterhin, wie ein Element grundsätzlich im Layout erscheint (Block, Inline, Inline-Block).
- `position` steuert dagegen, wie es im oder ausserhalb des Dokumentflusses platziert wird.
- Erst das Zusammenspiel beider Eigenschaften ergibt ein stabiles Layout.

Lösung Abschlussaufgabe (optional)

Eine mögliche kompakte Lernseite erfüllt alle Kriterien durch:

- Semantik: `header`, `nav`, `main`, `section`, `footer`
- Selektoren: Kombination aus `h1`, `.info`, `#menu`, `p.info`, `h1, h2`
- Box-Modell: Karten mit `padding`, `margin`, `border`, global `box-sizing: border-box`
- Positionierung: fixe Kopfzeile plus absolut positionierte Marker in einem relativ positionierten Container