# LU07: Lösungen – Accordion debuggen

Vergleichen Sie die Lösungen mit Ihrem Debugging-Protokoll:

- Was haben Sie richtig erkannt?
- Welchen Bug haben Sie nicht gefunden – und warum?
- Welches Werkzeug hätte Ihnen dort geholfen?

## Bugs 1 und 2 (identisch in allen Varianten)

### Bug 1 – Fehlende Dateiendung in index.html

```
<!-- ❌ Dateiendung fehlt – Browser kann die Datei nicht
finden -->
<link href="style" rel="stylesheet">

<!-- ✅ Fix -->
<link href="style.css" rel="stylesheet">
```

Dieser Bug erzeugt **keinen Fehler in der Console**. Der Browser lädt die Seite trotzdem – nur ohne Styling. Der Editor (WebStorm / HTMLHint) unterstreicht den Pfad in Rot. Alternativ zeigt der **Network-Tab** in den DevTools einen 404-Fehler für die CSS-Datei.

Falls Sie diesen Bug nicht gefunden haben: Öffnen Sie beim nächsten Mal zuerst den Editor und suchen Sie nach roten Unterwellungen, bevor Sie die DevTools öffnen.

### Bug 2 – Script im ''<head>'' statt vor ''</body>''

```
<!-- ❌ Script wird ausgeführt bevor der Browser das HTML
gelesen hat -->
<!-- → document.querySelectorAll(...) findet keine Buttons,
weil sie noch nicht existieren -->
<head>
  <script src="script.js"></script>
</head>

<!-- ✅ Fix: Script am Ende von <body> -->
<body>
  <!-- ... ganzer HTML-Inhalt ... -->
  <script src="script.js"></script>
</body>
```

JavaScript läuft sofort, wenn der Browser die Zeile einliest. Steht das Script im <head>, existieren die Buttons im HTML noch nicht – `querySelectorAll` gibt `null` zurück. Die Fehlermeldung (`TypeError: Cannot read properties of null`) erscheint deshalb nicht beim Script-Bug selbst, sondern beim nächsten Schritt, der auf das Ergebnis zugreift.

CSS-Dateien gehören in den <head>, Script-Dateien ans Ende des <body>.

# Variante A

### Bug 3 – Fehlender Punkt vor dem Klassennamen

```
// ❌ Sucht nach einem HTML-Tag <accordion-btn> – gibt eine
leere NodeList zurück
const buttons = document.querySelectorAll('accordion-btn');

// ✅ Fix: Punkt vor dem Klassennamen nicht vergessen
const buttons = document.querySelectorAll('.accordion-btn');
```

Ohne den Punkt sucht der Browser nach einem HTML-Element mit dem Tag-Namen `accordion-btn` – so etwas existiert nicht. Das Ergebnis ist eine leere NodeList: `forEach` läuft zwar, aber über 0 Elemente. Es passiert schlicht nichts, und kein roter Fehler erscheint.

Der entscheidende Schritt: `console.log('Buttons:', buttons.length)` – ein Wert von `0` zeigt sofort, dass der Selector nichts findet.

### Bug 4 – Tippfehler ''heihgt'' in style.css

```
/* ❌ 'heihgt' statt 'height' – Browser ignoriert diese
Eigenschaft */
.panel {
  heihgt: 0;
  opacity: 0;
  overflow: hidden;
}

/* ✅ Fix */
.panel {
  height: 0;
  opacity: 0;
  overflow: hidden;
}
```

Der Browser ignoriert unbekannte CSS-Eigenschaften kommentarlos – kein Console-Fehler, keine Meldung. `height` bleibt auf dem Browser-Standard (`auto`), die Panels sind deshalb immer sichtbar. Der Editor markiert `heihgt` sofort als unbekannte Eigenschaft. Im **Elements-Tab** sieht man die Zeile durchgestrichen in den Styles.

## Bug 5 – Doppeltes ''classList.add'' ausserhalb des ''if''-Blocks

```
// ❌ classList.add('open') steht ausserhalb UND innerhalb
des if-Blocks
// → Das Panel wird immer geöffnet – egal ob es vorher offen
oder geschlossen war
buttons.forEach((button) => {
  button.addEventListener('click', (e) => {
    const panelElement = e.target.nextElementSibling;
    const panelIsOpen =
panelElement.classList.contains('open');

    buttons.forEach((andererButton) => {
andererButton.nextElementSibling.classList.remove('open');
      andererButton.setAttribute('aria-expanded', 'false');
    });

    panelElement.classList.add('open');  // ← diese Zeile
löschen

    if (!panelIsOpen) {
      panelElement.classList.add('open');
      button.setAttribute('aria-expanded', 'true');
    }
  });
});

// ✅ Fix: classList.add('open') nur innerhalb des if-Blocks
buttons.forEach((button) => {
  button.addEventListener('click', (e) => {
    const panelElement = e.target.nextElementSibling;
    const panelIsOpen =
panelElement.classList.contains('open');

    buttons.forEach((andererButton) => {
andererButton.nextElementSibling.classList.remove('open');
      andererButton.setAttribute('aria-expanded', 'false');
    });

    if (!panelIsOpen) {
      panelElement.classList.add('open');
      button.setAttribute('aria-expanded', 'true');
    }
  });
```

```
});
```

Kein Console-Fehler – das Verhalten ist subtil. Was passiert Schritt für Schritt beim Klick auf ein **offenes** Panel:

1. `panelIsOpen` wird auf `true` gesetzt ✓
2. Die `forEach`-Schleife entfernt `open` von allen Panels ✓
3. `panelElement.classList.add('open')` fügt `open` sofort wieder hinzu ← das ist das Problem
4. Die `if (!panelIsOpen)`-Bedingung ist `false`, läuft also nicht ✓

Das Panel wird also immer geöffnet – der Close-Mechanismus funktioniert nie.

# Variante B

### Bug 3 – Tippfehler ''opacty'' in style.css

```
/* ❌ 'opacty' statt 'opacity' – Browser ignoriert diese
Zeile */
.panel.open {
  height: auto;
  opacty: 1;
}

/* ✅ Fix */
.panel.open {
  height: auto;
  opacity: 1;
}
```

Das Accordion funktioniert scheinbar – der Button-Pfeil dreht sich, der Bereich klappt auf. Aber der Inhalt bleibt unsichtbar, weil `opacity: 0` aus `.panel` nie durch `opacity: 1` überschrieben wird. `opacty` ist eine unbekannte Eigenschaft und wird ignoriert. Im **Elements-Tab** sieht man `opacty` durchgestrichen in den Styles der `.panel.open`-Regel.

### Bug 4 – ''querySelector'' statt ''querySelectorAll''

```
// ❌ querySelector gibt nur das erste gefundene Element
zurück – kein Array, keine NodeList
const buttons = document.querySelector('.accordion-btn');

// ✅ Fix: querySelectorAll gibt alle gefundenen Elemente als
```

```
NodeList zurück
const buttons = document.querySelectorAll('.accordion-btn');
```

querySelector gibt ein einzelnes Element zurück – das erste, das dem Selector entspricht. Das hat zwei Konsequenzen: Der erste Button funktioniert korrekt, weil forEach auf einem einzelnen Element trotzdem läuft. Die anderen drei Buttons reagieren gar nicht.

Falls Sie nur den ersten Button getestet haben, schien alles zu funktionieren. console.log(buttons) zeigt den Unterschied: ein Element statt einer NodeList.

### Bug 5 – Fehlendes ''e'' als Parameter im Callback

```
// ❌ 'e' wird in der Funktion verwendet, aber nicht als
Parameter deklariert
buttons.forEach((button) => {
  button.addEventListener('click', () => {           // ← ()
ohne 'e'
    const panelElement = e.target.nextElementSibling; // →
ReferenceError
  });
});

// ✅ Fix: 'e' als Parameter ergänzen
buttons.forEach((button) => {
  button.addEventListener('click', (e) => {          // ←
(e) als Parameter
    const panelElement = e.target.nextElementSibling;
  });
});
```

Der Fehler erscheint erst beim Klick, nicht beim Laden der Seite – weil der Callback erst dann ausgeführt wird. Die Console zeigt Uncaught ReferenceError: e is not defined mit einer Zeilennummer. Ein Klick auf die Zeilennummer führt direkt zur Stelle mit e.target.

Das Event-Objekt e wird vom Browser automatisch an den Callback übergeben – aber nur, wenn Sie es als Parameter deklarieren. Fehlt der Parameter, existiert e im Funktionskontext nicht.

## Variante C

### Bug 3 – Syntaxfehler in forEach: fehlende Klammer

```
// ❌ Öffnende Klammer '(' fehlt vor 'andererButton'
// → Console: Uncaught SyntaxError: Unexpected token '=>'
```

```
buttons.forEach(andererButton) => {
  andererButton.nextElementSibling.classList.remove('open');
});

// ✅ Fix: Klammern um den Parameter ergänzen
buttons.forEach((andererButton) => {
  andererButton.nextElementSibling.classList.remove('open');
});
```

Ein `SyntaxError` ist besonders einschneidend: Der Browser kann das gesamte Script nicht parsen und führt **gar nichts** davon aus – nicht nur die fehlerhafte Zeile. Deshalb passiert beim Klick absolut nichts, obwohl die anderen Teile des Codes korrekt sind. Die Console zeigt den Fehler sofort beim Laden der Seite, noch bevor Sie einen Button angeklickt haben.

## Bug 4 – Tippfehler ''heigth'' in style.css

```
/* ❌ 'heigth' statt 'height' – Browser ignoriert diese Zeile */
/* → height bleibt auf dem Browser-Standard (auto), Panels sind immer sichtbar */
.panel {
  heigth: 0;
  opacity: 0;
  overflow: hidden;
}

/* ✅ Fix */
.panel {
  height: 0;
  opacity: 0;
  overflow: hidden;
}
```

Dieser Bug ist das Gegenteil von Variante A: Dort öffneten sich die Panels nicht, weil `height: 0` nicht gesetzt war. Hier sind die Panels immer sichtbar, weil `height` nie auf `0` gesetzt wird – der Browser-Standard ist `auto`. Der Editor markiert `heigth` als unbekannte Eigenschaft, im Elements-Tab erscheint die Zeile durchgestrichen.

## Bug 5 – ''nextSibling'' statt ''nextElementSibling''

```
// ❌ nextSibling gibt den unsichtbaren Text-Node zwischen den Tags zurück
```

```
// → kein DOM-Element → TypeError: Cannot read properties of
undefined (reading 'classList')
const panelElement = e.target.nextSibling;

// ✅ Fix: nextElementSibling überspringt Text-Nodes
const panelElement = e.target.nextElementSibling;
```

Im HTML-Code sieht man zwischen `<button>` und `<div class=„panel“>` nichts – aber der Browser erzeugt aus dem Zeilenumbruch und den Einrückungen einen unsichtbaren **Text-Node**. `nextSibling` gibt diesen Text-Node zurück, kein DOM-Element – und ein Text-Node hat kein `.classList`.

Öffnen Sie den **Elements-Tab** und klappen Sie die Accordion-Struktur auf: Sie sehen zwischen den Tags `#text`-Einträge. Das sind diese Text-Nodes. `nextElementSibling` überspringt sie automatisch und gibt das nächste echte HTML-Element zurück.