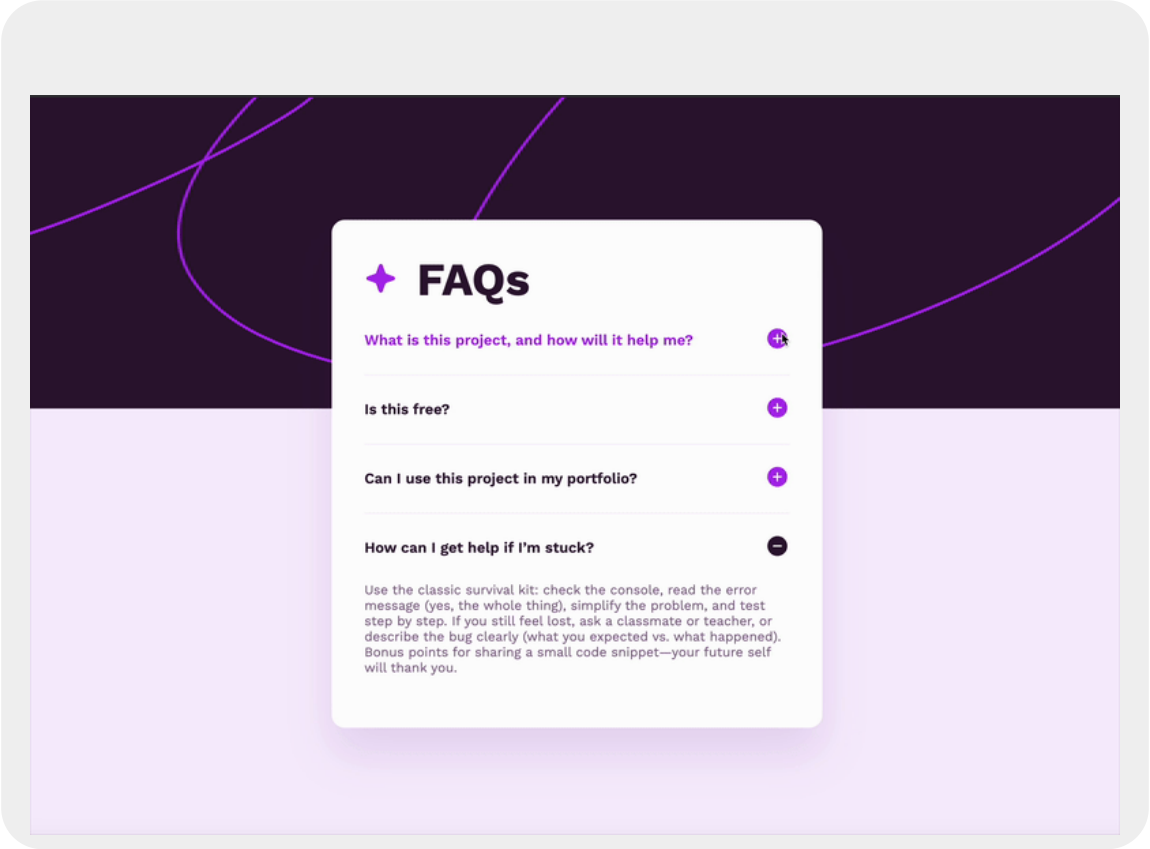
LU05a – DOM Traversal & NodeLists

Ausgangslage: Das Accordion-Projekt

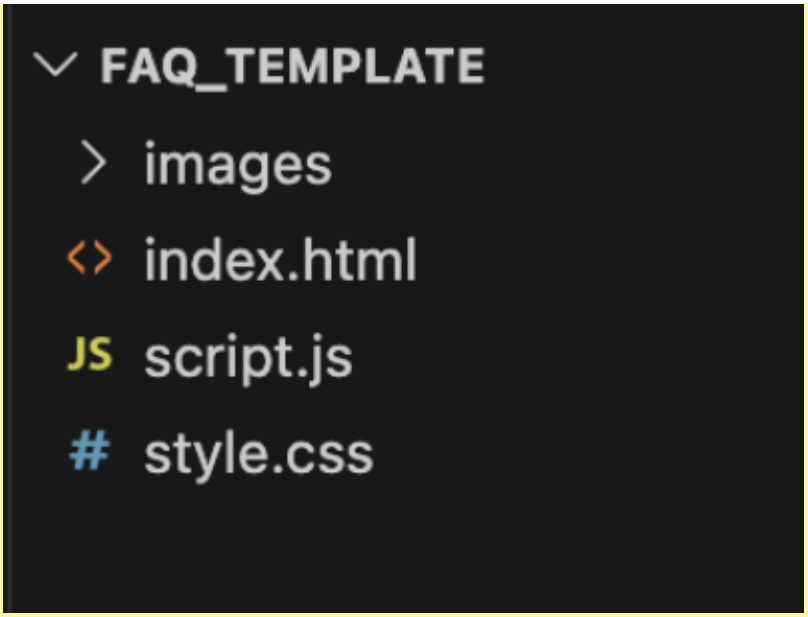
In diesem Block bauen Sie ein interaktives FAQ Accordion.



Laden Sie hier das Figma-File, die Start-HTML und das Readme herunter:

Accordion Starter

Projektaufbau: Erstellen Sie drei Dateien: `index.html`, `script.js` und



`style.css`.

Verlinken Sie CSS- und JavaScript-File im HTML:

1. **CSS:** im <head> mit

   ```
   <link href="style.css" rel="stylesheet">
   ```

2. **JS-Script:** am Ende des <body> mit

   ```
   <script src="script.js"></script>
   ```

   .

Testen Sie zuerst, ob die Verlinkung klappt – dieser Code gehört in `script.js`:

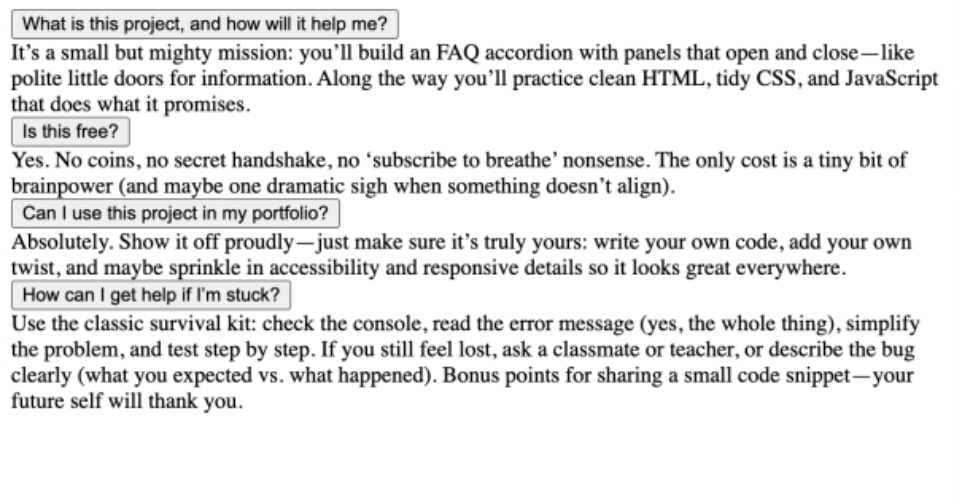
```
console.log('Hello World!'); // erscheint das in der
DevTools-Console?
```

Der Aufbau beginnt bewusst **ohne Styling** – Funktionalität zuerst. Das **HTML-Grundgerüst** sieht so aus:

```
<div class="accordion">
  <h1>FAQs</h1>

  <div class="accordion-item">
    <button class="accordion-btn" aria-
expanded="false">
      What is this project, and how will it help me?
    </button>
    <p class="panel">
      It's a small but mighty mission: you'll build
```

```
an FAQ accordion...
    </p>
  </div>

  <div class="accordion-item">
    <button class="accordion-btn" aria-
expanded="false">
      Is this free?
    </button>
    <p class="panel">
      Yes. No coins, no secret handshake...
    </p>
  </div>

  <!-- weitere .accordion-item ... -->
</div>
<script src="script.js"></script>
```

Anschliessend sollte es so aussehen im Browser:



Schritt 1: Panels (Antworten) mit CSS ein- und ausblenden

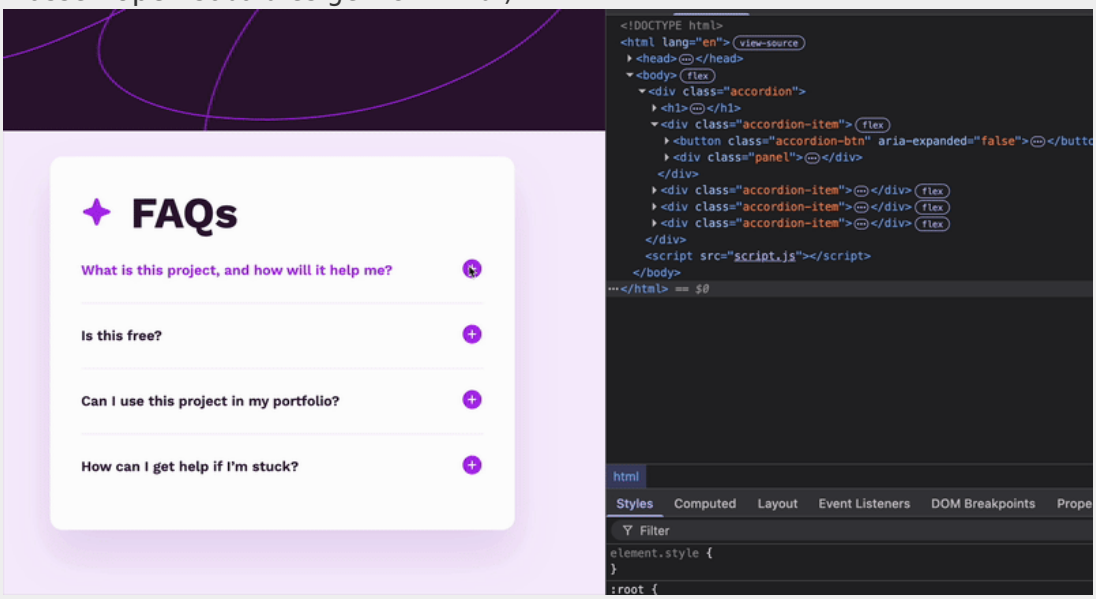
Bevor wir JavaScript schreiben, definieren wir die zwei möglichen Zustände im CSS. Die Klasse open wird später per JS gesetzt oder entfernt – das CSS übernimmt das An- und Ausblenden:

Dieser Code kommt ins `style.css`.

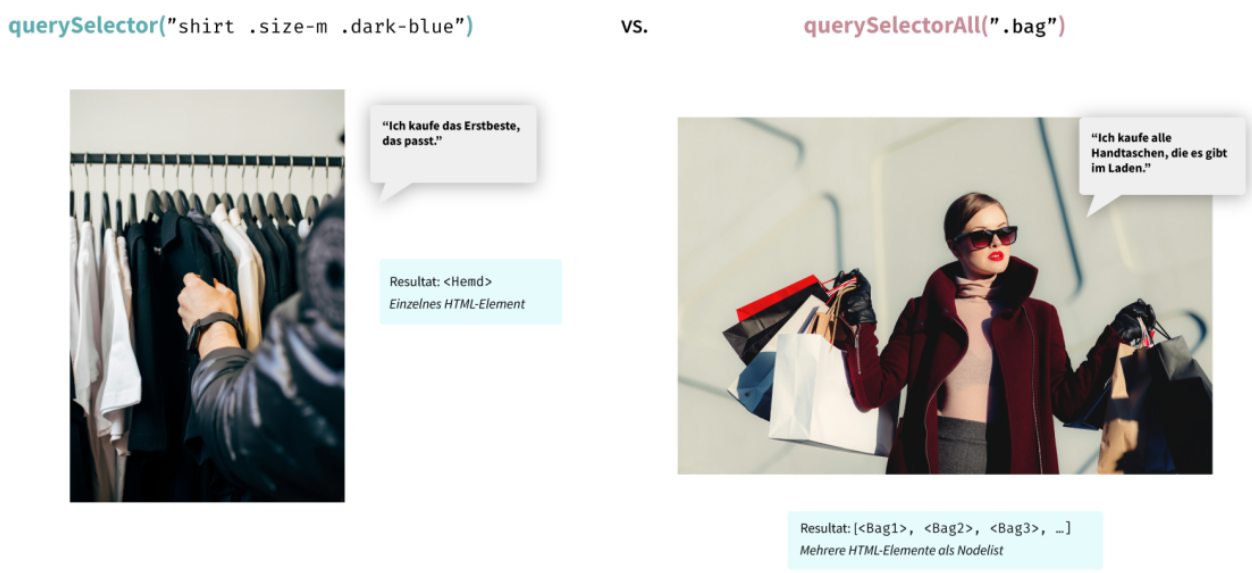
```
/* Standardmässig werden die Antworten versteckt */
.panel {
  display: none;
}
```

```css
/* Sichtbar werden sie nur, wenn Klasse 'open' gesetzt */
.panel.open {
  display: block;
}
```

Was macht dieser Code (später)? Er blendet nicht aktive Antworten aus und blended das aktive Element ein (div mit Klasse `.panel` bekommt eine zweite Klasse `.open` sobald es geklickt wird.):



Schritt 2: querySelector vs. querySelectorAll



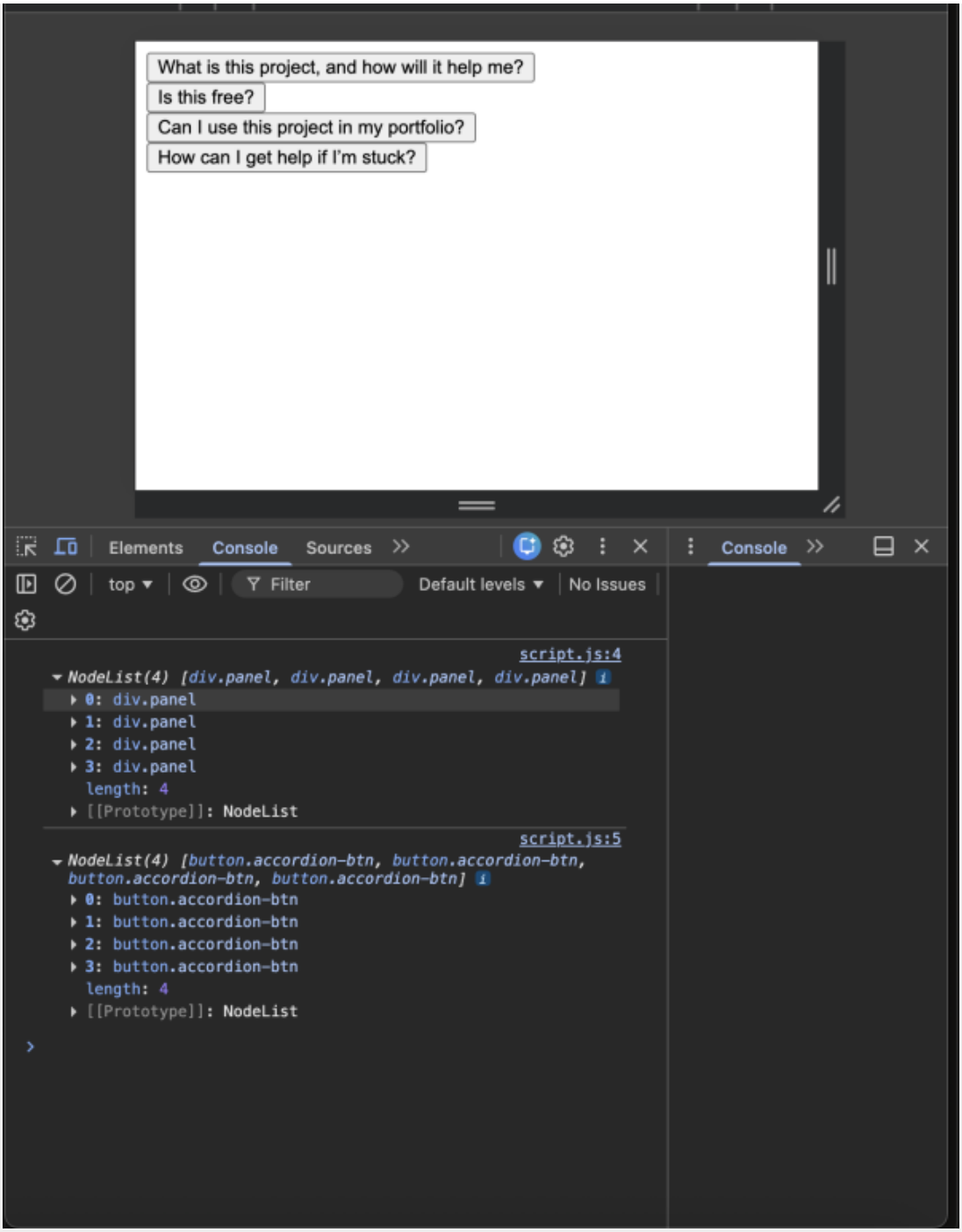
Als Erstes selektieren wir die Elemente im DOM. Was passiert, wenn wir `querySelector()` verwenden?

```
const panel = document.querySelector('.panel');
const btn   = document.querySelector('.accordion-btn');

console.log(panel); // → <p class="panel">...</p>     NUR
das erste!
console.log(btn);   // → <button class="accordion-
btn">...</button>   NUR das erste!
```

`querySelector()` gibt immer nur das **erste** passende Element zurück. Für ein Accordion mit vier Fragen reicht das nicht – wir brauchen alle.

Die Lösung: `querySelectorAll()` gibt **alle** passenden Elemente zurück:

```
const panels  = document.querySelectorAll('.panel');
const buttons = document.querySelectorAll('.accordion-btn');

console.log(panels);  // → NodeList(4) [p.panel, p.panel,
p.panel, p.panel]
console.log(buttons); // → NodeList(4) [button.accordion-
btn, ...]
```



Was ist eine NodeList?

`querySelectorAll()` gibt kein gewöhnliches Array zurück, sondern eine **NodeList** – eine listenartige Sammlung von DOM-Elementen.

Was eine NodeList **kann:**

- Per Index auf eine einzelnes Element zugreifen: `list[0]`, `list[1]` ...
- Wieviele Elemente hat es davon?: `list.length`

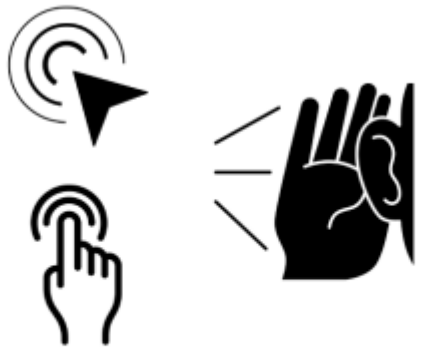
- `forEach()` – über alle Elemente iterieren

Was eine NodeList standardmässig **nicht** kann (anders als ein echtes Array):

- `.map()`, `.filter()`, `.reduce()`

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

console.log(buttons.length); // → 4
console.log(buttons[0]);     // → <button class="accordion-
btn">...</button>
```

Schritt 3: forEach() – EventListener auf alle Buttons setzen



```
btn.addEventListener("click", () => { … })
```

Auf einen **einzelnen Button** einen EventListener setzen würde so aussehen:

```
button.addEventListener('click', () => {
  console.log('Geklickt!');
});
```

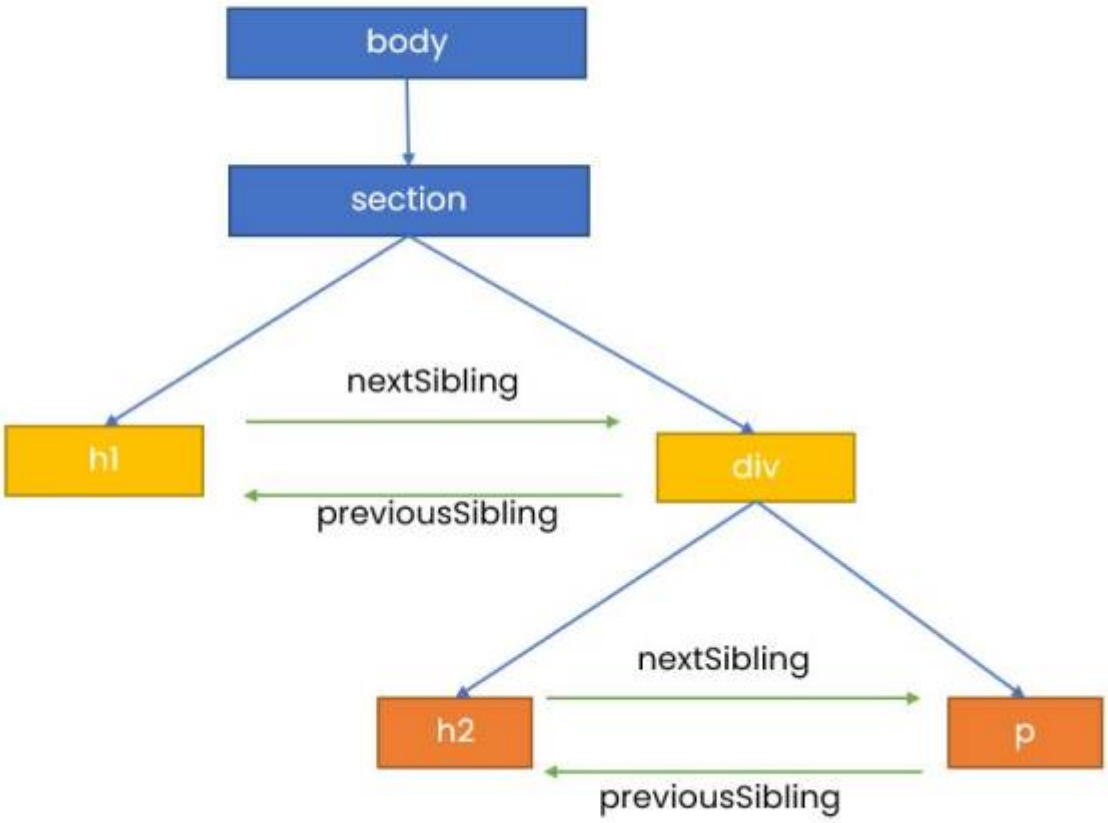
Da wir aber eine NodeList haben, iterieren wir mit `forEach()` über alle Elemente und setzen auf jedem einen Listener:

```
buttons.forEach(b => {
  b.addEventListener('click', (e) => {
```

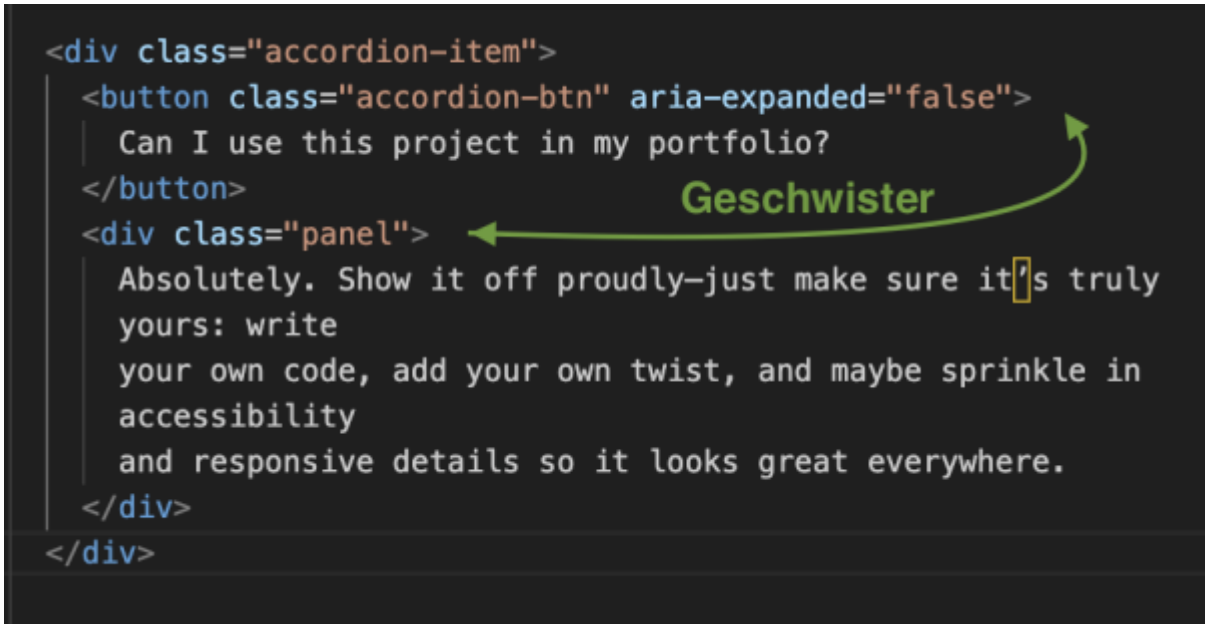
```
    console.log('Geklickt:', e.target);
  });
});
```

Öffnen Sie die DevTools-Console und klicken Sie auf verschiedene Buttons. `e.target` sollte immer den jeweiligen geklickten Button ausgeben.

Schritt 4: Das richtige Panel finden – DOM Traversal



Wir haben jetzt EventListener auf allen Buttons. Wenn ein Button geklickt wird, müssen wir das zugehörige <p class=„panel“> öffnen. Die Frage ist: **Wie wissen wir, welches Panel zu welchem Button gehört?** –> Button und Panel sind Geschwister im HTML:



DOM Traversal mit nextElementSibling

Die sauberere Lösung nutzt die **Struktur des DOMs** selbst. Im HTML ist jedes Panel das direkte Geschwister-Element des zugehörigen Buttons:

```
<div class="accordion-item">
  <button class="accordion-btn">...</button>   ← Button
  <p class="panel">...</p>                     ← direkt
daneben: das Panel
</div>
```

Mit `nextElementSibling` navigieren wir direkt vom geklickten Button zu seinem Panel:

```
buttons.forEach((b) => {
  b.addEventListener('click', (e) => {
    const panelElement = e.target.nextElementSibling; // 
direkt das zugehörige Panel
    panelElement.classList.add('open');
  });
});
```

Warum nicht einfach nextSibling?

Der erste Versuch mit `nextSibling` scheitert:

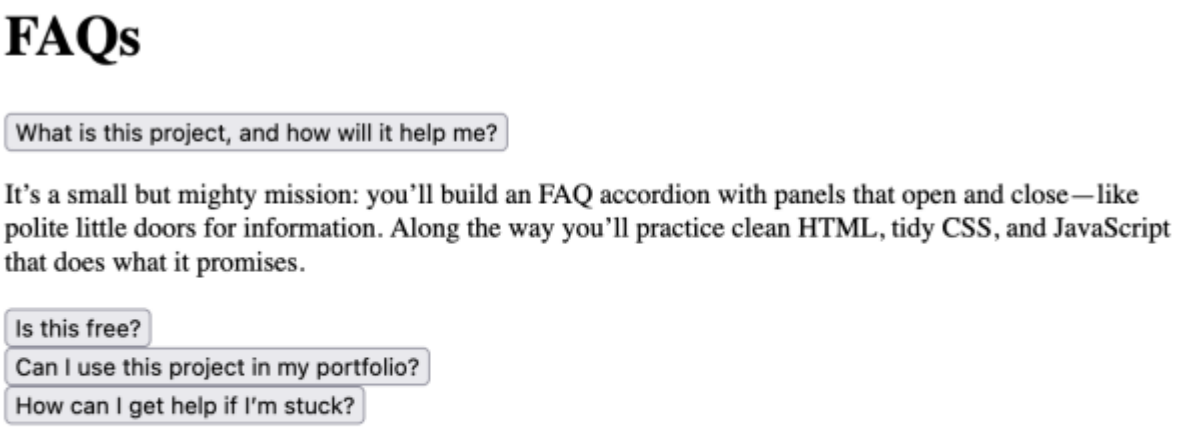
```
buttons.forEach((b) => {
  b.addEventListener('click', (e) => {
    const panelElement = e.target.nextSibling;
    panelElement.classList.add('open'); // □ Fehler in der
Console!
    console.log(e.target.nextSibling);  // → #text  (ein
Zeilenumbruch!)
  });
});
```

Warum? Im HTML-Quellcode sind Zeilenumbrüche und Leerzeichen zwischen Tags eigenständige DOM-Nodes (sog. **Text-Nodes**). `nextSibling` gibt den allernächsten Node zurück – das ist der Zeilenumbruch nach <button>, kein HTML-Element. `nextElementSibling` überspringt Text-Nodes und gibt immer das nächste **HTML-Element** zurück.

Eigenschaft	Beschreibung
`element.nextElementSibling`	Nächstes Geschwister-Element
`element.previousElementSibling`	Vorheriges Geschwister-Element
`element.parentElement`	Elternelement
`element.firstElementChild`	Erstes Kind-Element
`element.lastElementChild`	Letztes Kind-Element

Testen Sie: Alle Panels sollten sich jetzt öffnen lassen – aber noch nicht schliessen.

Schritt 5 & 6: Zum finalen Script



Jetzt bauen wir den Code Schritt für Schritt zum finalen Script aus. Drei Dinge müssen noch gelöst werden:

1. Alle anderen Panels schliessen, wenn eines geöffnet wird
2. Ein geöffnetes Panel beim erneuten Klick wieder schliessen (Toggle)
3. Sicherstellen, dass wir immer den `<button>` ansprechen – nicht ein Kind-Element davon

Problem: e.target kann das falsche Element sein

Wenn der Button ein Icon enthält (bei uns: das + und – via `::after`), kann ein Klick genau auf dieses Icon landen. Dann zeigt `e.target` auf das Icon – nicht auf den Button selbst.

Eigenschaft	Beschreibung
`event.target`	Das Element, das das Event **ursprünglich ausgelöst** hat – kann ein Kind-Element (wie ein Icon) sein
`event.currentTarget`	Das Element, an dem der **EventListener registriert** wurde – immer der Button

Die Lösung: Wir lesen den Button immer via `e.currentTarget` aus und speichern ihn in einer Variable. Damit haben wir auch die stabile Basis für `nextElementSibling`.

Das finale Script

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

buttons.forEach((button) => {
  button.addEventListener('click', (e) => {
    const btn          = e.currentTarget;           // immer
der <button>
    const panelElement = btn.nextElementSibling;    //
direkt das zugehörige Panel
    const panelIsOpen  =
panelElement.classList.contains('open'); // aktuellen
Zustand lesen

    // Alle Panels schliessen
    buttons.forEach((andererBtn) => {
andererBtn.nextElementSibling.classList.remove('open');
      andererBtn.setAttribute('aria-expanded', 'false');
    });

    // Dieses Panel öffnen – aber nur wenn es vorher
geschlossen war
    if (!panelIsOpen) {
      panelElement.classList.add('open');
      btn.setAttribute('aria-expanded', 'true');
    }
```

```
    });
});
```

Was passiert bei jedem Klick?

1. `e.currentTarget` gibt uns sicher den <button>, egal ob auf Text oder Icon geklickt wurde
2. `btn.nextElementSibling` gibt das direkt zugehörige Panel zurück
3. `classList.contains('open')` liest den aktuellen Zustand, bevor wir alles schliessen
4. Das innere `forEach()` schliesst alle Panels und setzt `aria-expanded` auf `false`
5. Die `if`-Bedingung öffnet das geklickte Panel nur dann, wenn es vorher zu war – so funktioniert auch der Toggle (zweimal klicken schliesst)

Das `setAttribute('aria-expanded', …)` erklären wir ausführlich in den folgenden Seiten.

Zusammenfassung

Konzept	Merksatz
`querySelector()`	Gibt **ein** Element zurück – das erste passende
`querySelectorAll()`	Gibt eine **NodeList** zurück – alle passenden
`forEach()`	Iteriert über eine NodeList wie über ein Array
`nextElementSibling`	Nächstes Geschwister-**Element** (überspringt Text-Nodes)
`e.target`	Das Element, das das Event ausgelöst hat (kann Kind-Element sein)
`e.currentTarget`	Das Element mit dem registrierten EventListener