4주차 · 3시간 워크숍

JavaScript와 AI로 기능 완성하기

값·변수·조건문·함수·반복문을 직접 실행해 본 뒤, 3주차 상품 페이지의 FAQ가 사용자 행동에 반응하도록 만들고 결과를 검증합니다.

04

이번 주에 할 수 있게 되는 것

- 이번 기능에 필요한 문자열·숫자·불리언 등의 핵심 값과 나머지 참고 자료형을 구분합니다.
- `const`, 비교, 함수, 반복문의 역할을 읽고 짧은 코드를 직접 실행합니다.
- JavaScript가 HTML 요소를 찾고 사용자 이벤트에 반응하는 흐름을 읽습니다.
- FAQ 버튼의 접근성 상태와 실제 화면 상태를 함께 바꿉니다.
- AI에 수정 범위를 제한하고 설명·변경 줄·브라우저 결과를 대조합니다.
- 콘솔의 단서를 따라 세 가지 오류를 찾고 기본 화면·동작 QA를 수행합니다.
- Shopify Liquid와 브라우저 JavaScript의 역할과 실행 시점을 구분합니다.

수업 흐름

1. FAQ 먼저 열어보기와 핵심 문법 35분
2. 요소 선택과 이벤트 연결 25분
3. 휴식 10분
4. FAQ 열기·닫기 구현 50분
5. AI 제한 수정과 변경 비교 25분
6. 휴식 10분
7. 콘솔 디버깅과 최종 QA 25분

교재 목차

실습 파일과 비교용 완성본
내 작업 폴더: 3주차에 작업한 폴더를 VS Code로 엽니다. 그 안의 `index.html`, `style.css`, `script.js`를 이번 주에도 계속 수정합니다.

비교용 파일: 3주차 CSS 완성본 열기 · 4주차 JavaScript 완성본 열기

내 작업 폴더가 없다면 `examples/product-landing/week-3-css` 폴더를 같은 위치에 복사해 `my-product-page`로 이름을 바꿉니다. 비교용 원본 폴더는 직접 수정하지 않습니다. PDF에서도 위 링크를 클릭할 수 있습니다.

오늘의 필수 실습 순서

1. **Console 체험:** 완성본에서 네 줄을 실행해 답변 하나를 잠깐 엽니다. 이 코드는 파일에 저장하지 않습니다.
2. **기본 문법:** 예제1~3과 주문 계산기 문제1~3을 Console에서 실행합니다. 예제4~5와 문제4~5는 선택 복습입니다.
3. **실제 파일 구현:** 5장 FAQ 실습에서 `index.html`, `style.css`, `script.js`를 차례로 저장합니다.
4. **완료 확인:** 마우스와 키보드로 FAQ를 열고 닫은 뒤 Console 오류와 390px 화면을 확인합니다.

// 1. JavaScript가 하는 일

HTML은 읽을 정보의 구조를 만들고 CSS는 그 구조의 모양과 배치를 정합니다. JavaScript는 브라우저가 사용자의 클릭 같은 사건을 감지했을 때 실행할 동작을 연결합니다.

HTML

FAQ 질문을 실제 `button`으로 바꾸고 질문과 답변의 연결 정보를 둡니다.

CSS

FAQ 버튼의 기본 모양과 키보드 초점 표시를 추가합니다.

JavaScript

클릭된 버튼과 답변을 찾아 열린 상태와 숨김 상태를 갱신합니다.

실제 3주차 → 4주차 차이

`index.html`

질문 제목 안에 `button`이 생기고, `aria-controls`와 답변 `id`가 연결됩니다. 답변에는 `hidden`이 추가됩니다.

`style.css`

FAQ 버튼이 기존 제목처럼 보이도록 만들고, `:focus-visible`로 키보드 초점 테두리를 표시합니다.

`script.js`

비어 있던 파일에 요소 찾기, 클릭 이벤트, `aria-expanded`와 `hidden` 갱신 코드가 들어갑니다.

먼저 눈으로 비교

두 로컬 예제를 각각 열어 보세요. 3주차는 답변이 항상 보이고, 4주차 기본 완성본은 답변이 닫힌 상태로 시작해 질문을 눌렀을 때 열립니다.

초기 상태: 두 답변이 닫혀 있고 버튼의 `aria-expanded`는 `false`입니다.

클릭 후: 첫 답변이 보이고 해당 버튼의 `aria-expanded`가 `true`로 바뀝니다.

5분 먼저 해보기: FAQ 답변 하나 열기

문법을 모두 배우기 전에 JavaScript가 화면을 바꾸는 결과부터 확인합니다. 4주차 완성 예제를 열고 개발자 도구의 Console에 아래 네 줄을 한 번에 실행하세요.

개발자 도구는 상품 제목을 오른쪽 클릭해 **검사**를 선택하거나, macOS **Option+Command+I**, Windows **Ctrl+Shift+I** 또는 **F12**로 엽니다.

Console에만 실행 아래 코드는 결과를 먼저 체험하는 임시 실험입니다. `script.js` 에는 저장하지 않습니다.

```
const firstButton = document.querySelector('[data-faq-button]');
const firstAnswer = document.getElementById('shipping-answer');
firstButton.setAttribute('aria-expanded', 'true');
firstAnswer.hidden = false;
```

성공 기준: 첫 번째 배송 답변이 보입니다. 지금은 각 문법을 외우지 말고 “요소를 찾는다 → 상태를 바꾼다 → 화면이 달라진다”는 흐름만 확인합니다. 실험을 마치면 페이지를 새로고침해 원래 상태로 돌립니다.

// 2. 기본 문법 익히기

FAQ 완성 코드를 바로 복사하기 전에 작은 JavaScript를 직접 실행해 봅니다. 아래 예제는 서로 독립적입니다. **예제1~3은 필수**로 Console에 한 묶음씩 실행하고, 예제4~5는 수업 시간에 따라 함께 하거나 복습 때 실행합니다.

Console에서 실행하는 순서

1. 완성 예제 페이지를 열고 개발자 도구의 Console 탭으로 이동합니다.
2. 코드 한 묶음을 붙여 넣고 Enter를 누릅니다.
3. 출력이 예상과 같은지 확인한 뒤, 숫자나 문자열을 바꾸어 다시 실행합니다.
4. `Identifier has already been declared` 가 나오면 페이지를 새로고침해 Console 상태를 초기화합니다.

문장을 읽는 규칙

문장과 주석

대입이나 함수 호출 같은 한 동작을 문장으로 읽습니다. 문장 끝의 `;`은 생략 가능한 경우도 있지만 교재에서는 경계를 분명히 하려고 일관되게 씁니다. `//` 뒤는 실행하지 않는 설명입니다.

괄호의 역할

`()`는 함수에 값을 전달하거나 조건을 묶고, `{}`는 함수·조건문·반복문이 실행할 문장 범위를 만듭니다. `[]`는 여러 값을 순서대로 담는 배열에 사용합니다.

이름과 들여쓰기

`productName`처럼 역할이 보이는 이름을 씁니다. JavaScript는 대소문자를 구분하므로 `price`와 `Price`는 다른 이름입니다. 들여쓰기는 실행 규칙은 아니지만 코드의 포함 관계를 보여 줍니다.

값과 자료형

값은 프로그램이 기억하고 계산할 대상이며, **자료형**은 그 값을 어떤 방식으로 다룰지 구분합니다. JavaScript의 원시 자료형은 일곱 개이고, 원시 값이 아닌 나머지는 객체로 다룹니다.

학습 순서를 구분하세요. 이번 주 필수는 직접 코드에 쓰고, 보면 알아보기는 오류를 읽을 정도로 이해합니다. 참고는 존재만 확인하고 외우지 않습니다.

자료형	예	의미와 쓰임
문자열 `string` 필수	`'무광 물병'`	`'click'`, `'aria-expanded'`처럼 글자를 표현하는 원시 자료형입니다.
숫자 `number` 필수	`29000`, `3.14`, `NaN`	가격·수량처럼 계산할 값을 표현합니다. `NaN`도 연산에 실패해 얻은 특수한 `number` 값입니다.
불리언 `boolean` 필수	`true`, `false`	답변을 열지, 숨길지처럼 둘 중 하나인 상태를 표현합니다.
미정 값 `undefined` 알아보기	`let selectedOption;`	변수를 만들었지만 값을 아직 넣지 않았거나, 객체에 요청한 속성이 없을 때 나타나는 원시 값입니다.
의도적인 빈 값 `null` 필수	`null`	값이 없음을 개발자가 명시합니다. `querySelector`가 조건에 맞는 HTML 요소를 찾지 못했을 때도 반환합니다.
큰 정수 `bigint` 참고	`9007199254740993n`	`number`가 안전하게 표현하기 어려운 매우 큰 정수에 사용합니다. 이번 FAQ에서는 직접 사용하지 않습니다.
고유 식별자 `symbol` 참고	`Symbol('id')`	서로 겹치지 않는 고유한 값을 만들어 객체의 속성 키 등에 사용합니다. 이번 FAQ에서는 직접 사용하지 않습니다.
객체 `object` 필수	`{ name: '물병' }` `['배송']`	이름표가 있는 값이나 여러 값을 묶습니다. 일반 객체, 배열, HTML 요소가 여기에 속합니다.

참고: 배열·함수·`null`을 `typeof`로 확인할 때

아래 예외는 이번 주 암기 항목이 아닙니다. Console에서 예상과 다른 결과가 나왔을 때 다시 찾아보세요.

- 배열은 객체이므로 `typeof ['배송']`은 `'object'`입니다. 배열인지 정확히 확인할 때는 `Array.isArray(['배송'])`를 사용합니다.
- 함수는 호출할 수 있는 객체이지만 `typeof`로 함수를 확인하면 `'function'`이라는 별도 결과가 나옵니다.
- `null`은 원시 값이지만 역사적인 이유로 `typeof null`의 결과는 `'object'`입니다. 실제 객체라는 뜻은 아닙니다.

[예제1] 값을 저장하고 출력하기

상품 이름, 가격, 판매 여부, 아직 선택되지 않은 답변을 각각 알맞은 자료형으로 저장하고 한 줄에 출력해 봅시다.

Console 필수 실습 아래 묶음을 실행하고 예상 출력과 비교합니다. 파일에는 저장하지 않습니다.

```
const productName = '무광 물병';
const price = 29000;
const isAvailable = true;
const selectedAnswer = null;

console.log(productName, price, isAvailable, selectedAnswer);
```

예상 출력: `무광 물병 29000 true null`

값 바꿔 보기
상품 이름과 가격을 원하는 값으로 바꿉니다. 이어서 `console.log(typeof productName, typeof price);`를 실행해 자료형도 확인합니다.

Q. `console.log` 줄을 지우면 왜 화면에는 아무 변화가 없을까요?
변수에 값을 저장하는 일과 그 값을 Console이나 웹 화면에 표시하는 일은 서로 다른 명령이기 때문입니다.

변수: const와 let

변수는 값에 이름을 붙여 이후 문장에서 다시 읽게 합니다. 기본적으로 `const`를 사용하고, 같은 이름에 다른 값을 다시 대입해야 할 때만 `let`을 사용합니다. `const answer`는 `answer`가 다른 요소를 가리키게 만들지 않겠다는 뜻이며, 그 요소의 `hidden` 같은 속성까지 절대 바뀌지 않는다는 뜻은 아닙니다.

[예제2] 수량을 바꾸기

처음 수량을 `1`로 저장한 뒤 하나를 더하고, 바뀐 값을 출력해 봅시다.

Console 필수 실습 실행 뒤 첫 수량과 더하는 값을 바꿔 다시 확인합니다.

```
let quantity = 1;
quantity = quantity + 1;

console.log(quantity);
```

예상 출력: `2`. `const quantity = 1`로 바꾸면 다음 줄의 재대입에서 오류가 납니다.

값 바꿔 보기
처음 수량을 `3`으로 바꾸고 두 개를 더하도록 식을 수정합니다. 예상값을 먼저 적은 뒤 실행 결과와 비교합니다.

Q. 모든 변수를 `let`으로 만들면 안 될까요?
실행은 가능하지만 바뀌지 않아야 할 값까지 실수로 재대입할 수 있습니다. 그래서 기본값은 `const`, 재대입이 필요한 이름만 `let`으로 구분합니다.

연산자와 조건식

연산자는 값을 계산하거나 비교합니다. `=`은 오른쪽 값을 왼쪽 이름에 대입하고, `===`은 두 값과 자료형이 같은지 비교해 불리언을 만듭니다. `&&`는 두 조건이 모두 참인지, `||`는 하나라도 참인지, `!`는 참·거짓을 반대로 바꾸는지 나타냅니다.

[예제3] 주문 금액 계산하기

단가와 수량을 곱해 주문 금액을 구하고, 재고가 있으면서 수량이 1개 이상인지 함께 판정해 봅시다.

Console 필수 실습 실행 전에 두 출력값을 먼저 예상합니다.

```
const unitPrice = 29000;
const orderQuantity = 2;
const hasStock = true;
const totalPrice = unitPrice * orderQuantity;
const canOrder = hasStock && orderQuantity > 0;

console.log(totalPrice, canOrder);
```

예상 출력: `58000 true`. FAQ의 `=== 'false'`와 `!willOpen`도 같은 비교·반전 규칙입니다.

값 바꿔 보기
`orderQuantity`를 `0`으로, `hasStock`을 `false`로 각각 바꾸어 어떤 조건에서 `canOrder`가 거짓이 되는지 확인합니다.

Q. `=`과 `===`은 왜 구분해야 할까요?
`=`은 값을 저장하고, `===`은 두 값을 비교해 `true` 또는 `false`를 만듭니다.

조건문과 함수

`if` 는 괄호 안 조건이 참일 때 첫 블록을 실행하고, 거짓이면 `else` 블록을 실행합니다. **함수**는 여러 문장을 하나의 이름으로 묶습니다. 아래 함수의 `inStock` 과 `quantity` 는 호출할 때 받는 매개변수이고, `return` 은 계산 결과를 함수 바깥으로 돌려줍니다.

[예제4] 주문 가능 여부를 함수로 판단하기

재고와 수량을 받아 주문 가능 여부를 알려 주는 함수를 만들고, 실제 값을 전달해 결과를 확인해 봅시다.

Console 선택 실습 시간이 남거나 복습할 때 실행합니다. 파일에는 저장하지 않습니다.

```
function getOrderMessage(inStock, quantity) {
  if (inStock && quantity > 0) {
    return '주문할 수 있습니다.';
  } else {
    return '주문할 수 없습니다.';
  }
}

console.log(getOrderMessage(true, 2));
```

예상 출력: `주문할 수 있습니다.` . 뒤에서 만날 `() => { ... }` 도 이름 없이 작성한 함수이며, 클릭이 일어날 때 호출됩니다.

값 바꿔 보기

`getOrderMessage(false, 2)` 와 `getOrderMessage(true, 0)` 을 각각 호출하고, 두 결과가 같은 이유를 조건식에서 찾아봅니다.

Q. 함수를 선언했는데 왜 결과가 바로 출력되지 않을까요?

함수 선언은 실행할 문장을 준비하는 단계입니다. 함수 이름 뒤에 괄호를 붙여 호출해야 내부 문장이 실행됩니다.

배열·객체·반복문

배열은 여러 값을 순서대로 담고 `faqTopics[0]`처럼 0부터 시작하는 위치로 읽습니다. **객체**는 관련 값을 이름표가 있는 속성으로 묶어 `product.name`처럼 읽습니다. **반복문**은 배열이나 요소 묶음의 각 항목에 같은 동작을 적용합니다.

[예제5] FAQ 주제를 반복 출력하기

FAQ 주제를 배열에 담고, 상품 객체의 이름과 함께 각 주제를 차례로 출력해 봅시다.

Console 선택 실습 반복문이 같은 작업을 여러 값에 적용하는지 확인합니다.

```
const faqTopics = ['배송', '교환'];
const product = { name: '무광 물병', price: 29000 };

for (const topic of faqTopics) {
  console.log(`${product.name}: ${topic}`);
}
```

예상 출력: `무광 물병: 배송`, 다음 줄에 `무광 물병: 교환`. `faqTopics.length`는 배열에 들어 있는 값의 수인 `2`입니다.

값 바꿔 보기
배열 마지막에 `'반품'`을 추가하고 다시 실행합니다. `faqTopics[0]`과 `faqTopics.length`도 각각 출력해 봅니다.

FAQ 코드와 연결하기

기본 문법	FAQ 코드	읽는 방법
변수	`const buttons = ...`	찾은 버튼 묶음에 `buttons` 라는 이름을 붙입니다.
불리언·비교	`const willOpen = ... === 'false'`	현재 닫혀 있는지를 비교해 다음 상태를 `willOpen` 에 저장합니다.
배열과 반복	`for ... of`	찾은 FAQ 버튼을 하나씩 `button` 으로 읽습니다.
객체의 속성	`answer.hidden`	답변 요소가 가진 숨김 상태를 읽거나 바꿉니다.
객체의 메서드	`button.getAttribute(...)`	`getAttribute` 라는 동작을 호출해 속성값을 읽습니다.
함수	`() => { ... }`	클릭 뒤 실행할 문장을 하나의 함수로 묶습니다.

▲ 실습: 작은 상품 주문 계산기

예제 코드를 그대로 복사하지 말고, 문제1에서 만든 값을 다음 문제에 계속 사용합니다. 수업에서는 **문제1~3을 먼저** 진행하고, 문제4~5는 시간이 남거나 복습할 때 선택합니다. 먼저 예상 결과를 적고 Console 결과와 비교합니다.

Console 연속 실습 문제1의 코드를 시작으로 다음 문제의 코드를 같은 Console 입력 묶음에 이어 작성합니다. 이 계산기는 FAQ 구현 전 문법 연습이며 `script.js` 에 저장하지 않습니다.

막혔을 때 확인할 것
변수 이름의 대소문자, 문자열 따옴표, 괄호의 짝, `=` 과 `===` , 중괄호 안 들여쓰기를 차례로 확인합니다.

[문제1] 상품 정보와 자료형 확인하기

`productName` , `unitPrice` , `inStock` 에 상품 이름·가격·재고 상태를 저장해 출력하고, `typeof` 로 세 값의 자료형을 확인합니다.

[문제2] 두 개 주문 금액 계산하기

수량 `2` 를 새 변수에 저장하고 단가와 곱해 `orderTotal` 을 만든 뒤 출력합니다.

[문제3] 무료 배송 판정하기

`orderTotal`이 50,000원 이상이면 `'무료 배송'`, 아니면 `'배송비 3,000원'`을 출력하는 조건문을 작성합니다.

[문제4 · 선택] 주문 금액 함수 만들기

가격과 수량을 받아 곱한 값을 반환하는 `calculateTotal` 함수를 만들고 두 가지 수량으로 호출합니다.

[문제5 · 선택] FAQ 코드 설명하기

`willOpen`, `for ... of`, `answer.hidden`, `getAttribute`가 각각 어떤 기본 문법을 사용하는지 한 문장씩 설명합니다.

// 3. 요소 찾기

`const`는 이번 코드에서 다시 다른 값으로 대입하지 않을 이름을 만듭니다. 이름을 붙여 두면 긴 선택이나 속성 표현을 이후 줄에서 짧게 읽을 수 있습니다.

읽기용 조각 코드 지금은 한 줄의 의미만 읽습니다. 실제 `script.js` 저장은 5장에서 시작합니다.

```
const buttons = document.querySelectorAll('[data-faq-button]');
```

`querySelectorAll`은 주어진 CSS 선택자와 일치하는 모든 요소를 찾습니다. 여기서는 `data-faq-button` 속성이 있는 두 질문 버튼을 찾습니다. 반환된 `NodeList`는 배열과 완전히 같지는 않지만 `length`를 확인하고 `for ... of`로 항목을 차례로 읽을 수 있는 요소 묶음입니다. 이번 기능에서는 반복 방식을 `for ... of` 하나로 통일합니다.

질문과 답변의 연결 고리

읽기용 HTML 버튼의 `aria-controls` 와 답변의 `id` 가 같은지만 찾아보세요.

```
<button type="button"
        data-faq-button
        aria-expanded="false"
        aria-controls="example-answer">
  언제 배송되나요?
</button>
<p id="example-answer" data-faq-answer hidden>
  결제 확인 후 순차적으로 발송됩니다.
</p>
```

버튼의 `aria-controls` 값과 답변의 `id` 가 같아야 합니다. `getElementById` 는 그 값을 이용해 한 개의 답변을 찾습니다.

읽기용 조각 코드 앞의 버튼에서 연결 이름을 읽고 같은 `id` 의 답변을 찾는 두 동작입니다.

```
const answer = document.getElementById(
  button.getAttribute('aria-controls')
);
```

`answer` 가 `null` 이라면
JavaScript 문법보다 먼저 `aria-controls` 와 답변의 `id` 철자를 비교합니다. 두 값이 다르면 답변을 찾지 못해 `answer.hidden` 을 읽는 줄에서 오류가 날 수 있습니다.

선택 결과를 콘솔에서 확인
개발자 도구 Console에 `document.querySelectorAll('[data-faq-button]').length` 를 입력합니다. 실제 예제에서는 `2` 가 보여야 합니다.

// 4. 이벤트 연결하기

이벤트는 클릭이나 키 입력처럼 브라우저에서 일어난 사건입니다. `addEventListener`는 "어떤 요소에서 어떤 사건이 생기면 어떤 함수를 실행할지"를 미리 등록합니다.

읽기용 기본형 `button` 이라는 변수가 준비되어 있다고 가정한 문법 모양입니다. 이 조각만 파일이나 Console에 복사하지 않습니다.

```
button.addEventListener('click', () => {
  // 버튼이 클릭된 뒤 실행할 코드
});
```

`addEventListener`를 세 부분으로 읽기

부분	이번 코드	의미
이벤트 대상	`button`	사건을 기다릴 HTML 버튼 객체입니다.
이벤트 이름	`'click'`	기다릴 사건의 종류를 문자열로 적습니다.
콜백 함수	`() => { ... }`	클릭이 발생한 뒤 브라우저가 호출할 함수입니다.

페이지를 읽을 때 브라우저는 콜백을 **등록**할 뿐 함수 본문을 바로 실행하지 않습니다. 사용자가 클릭한 **나중**에 브라우저가 등록해 둔 함수를 호출합니다. 기본 `button`은 포커스 상태에서 Enter나 Space를 눌러도 `click` 이벤트를 발생시키므로 같은 콜백을 재사용할 수 있습니다.

함수 자체를 전달하기

읽기용 비교 함수 이름만 전달하는 경우와 괄호를 붙여 즉시 호출하는 경우의 차이를 읽습니다.

```
function handleClick(event) {
  console.log(event.type, event.currentTarget);
}

button.addEventListener('click', handleClick);
```

`handleClick`은 **함수 자체**를 전달합니다. 반대로 `handleClick()`처럼 괄호를 붙이면 등록하는 줄에서 함수를 즉시 실행하고, 그 반환값을 리스너로 전달하게 됩니다. "나중에 실행할 일"을 연결할 때는 호출 결과가 아니라 함수를 전달해야 합니다.

브라우저는 콜백을 실행할 때 사건의 정보를 담은 **이벤트 객체**를 첫 번째 인수로 건넵니다. 위 코드에서 `event.type`은 `'click'`, `event.currentTarget`은 리스너를 등록한 `button`입니다.

화살표 함수 읽기

화살표 함수도 일반 함수처럼 값을 입력받고, 여러 문장을 실행하고, 결과를 돌려줄 수 있습니다. 아래 첫 예제는 **클릭한 FAQ 버튼의 글자를 읽어 "○○ 클릭"이라는 문장을 만드는 함수**입니다.

Console 선택 실습 4주차 완성본을 새로고침한 뒤 아래 전체 묶음을 Console에 실행하고 첫 질문을 누릅니다. `script.js`에는 저장하지 않습니다.

```
const describeClick = (event) => {
  const label = event.currentTarget.textContent.trim();
  return `${label} 클릭`;
};

const firstButton = document.querySelector('[data-faq-button]');

firstButton.addEventListener('click', (event) => {
  const message = describeClick(event);
  console.log(message);
});
```

첫 번째 FAQ 버튼을 누르면 이렇게 실행됩니다

1. 브라우저가 클릭 정보를 담은 `event` 객체를 이벤트 리스너의 화살표 함수에 전달합니다.
2. `describeClick(event)`가 같은 이벤트 객체를 `describeClick` 함수의 매개변수로 전달합니다.
3. `event.currentTarget`은 리스너를 등록한 `firstButton`을 가리키고, `textContent.trim()`은 버튼에 보이는 글자에서 앞뒤 공백을 제거합니다.
4. `return`이 만든 문자열을 호출한 자리로 돌려주고, 그 값을 `message`에 저장합니다.
5. `console.log(message)`가 Console에 `언제 배송되나요? 클릭`처럼 표시합니다.

중요: `return`은 결과를 함수 바깥으로 돌려줄 뿐 화면에 자동으로 표시하지 않습니다. 결과를 보려면 이 예제처럼 `console.log`에 전달하거나 HTML 요소의 글자·상태를 바꾸는 코드에 사용해야 합니다.

한 줄 화살표 함수는 무엇이 다른가요?

Console 선택 실습 아래 세 줄은 HTML 요소 없이도 실행할 수 있습니다.

```
const double = (number) => number * 2;

const result = double(3);
console.log(result); // 6
```

`double(3)`을 호출하면 `number`에 `3`이 들어가고 `3 * 2`를 계산합니다. 함수 본문이 한 식뿐이므로 중괄호와 `return`을 생략했지만, 계산 결과 `6`을 돌려주는 동작은 같습니다.

- `(event)`는 함수가 전달받을 **매개변수**입니다. 값이 없으면 `()`, 하나면 `event`처럼 괄호를 생략할 수도 있습니다.
- `=>`는 입력과 실행할 **함수 본문**을 나누는 화살표입니다.
- 중괄호 `{ ... }`로 여러 문장을 쓰면 결과를 돌려줄 때 `return`을 직접 적습니다. `number * 2`처럼 한 식만 쓰면 중괄호와 `return`을 함께 생략할 수 있습니다.
- 이번 실습에서는 화살표 함수의 고급 규칙까지 외우지 않습니다. 매개변수, 화살표, 함수 본문 세 부분을 구분하고 클릭 뒤 실행된다는 점에 집중합니다.

이번 코드의 실행 순서

1. 브라우저가 `defer`가 붙은 `script.js`를 HTML 해석 뒤 실행합니다.
2. `querySelectorAll`로 FAQ 버튼을 찾고 각 버튼에 클릭 함수를 연결합니다.
3. 사용자가 버튼을 클릭하거나 버튼에 초점을 둔 채 Enter 또는 Space를 누릅니다.
4. 그 버튼과 연결된 답변을 찾고 다음 상태를 계산합니다.
5. 접근성 상태와 화면 표시를 함께 바꿉니다.

실제 코드의 `for ... of`는 찾은 버튼마다 같은 연결 작업을 적용하는 반복문입니다. 각 `button`은 HTML 버튼을 나타내는 객체이고, `addEventListener`는 그 객체가 제공하는 메서드입니다.

// 5. FAQ 열기·닫기

여기서부터는 내 작업 폴더의 실제 파일에 저장합니다. 기본 완성본을 세 단계로 나누어 작성합니다. 각 단계에서 파일과 위치를 먼저 확인하고, 저장 전에 변화를 예상한 뒤 실제 상태와 비교합니다.

화면 상태	aria-expanded 속성값	hidden 속성값
답변 닫힘	`'false'` · HTML 속성에서 읽은 문자열	`true` · 숨김 여부를 나타내는 불리언
답변 열림	`'true'` · HTML 속성에 쓸 문자열	`false` · 숨기지 않는 불리언

핵심 관계: 답변을 열 때는 펼침 상태가 `true`, 숨김 상태가 `false`입니다. 두 값은 반대 방향으로 함께 바뀝니다.

---

1. HTML 버튼과 CSS 준비

지금 따라 하기 · index.html → style.css

1. **index.html 위치:** `<h2 id="faq-title">` 바로 아래의 기존 FAQ `article` 두 개를 찾습니다.
2. **행동:** 두 `article`을 아래 HTML 전체로 교체하고 저장합니다.
3. **style.css 위치:** 기존 `.faq-item h3, .faq-item p` 규칙 다음 줄에 버튼 규칙을 추가하고 저장합니다.
4. **확인:** 답변은 닫히고, 질문은 버튼이지만 기존 제목처럼 보이는지 확인합니다.

실행 전 예상

버튼의 `aria-controls`와 답변의 `id`를 같은 값으로 두면 프로그램이 질문에 해당하는 답변을 정확히 찾을 수 있을 것입니다.

index.html에 저장할 코드

파일에 저장 기존 FAQ `article` 두 개를 아래 두 개로 교체합니다.

```
<article class="faq-item">
  <h3>
    <button type="button" data-faq-button
            aria-expanded="false"
            aria-controls="shipping-answer">
      언제 배송되나요?
    </button>
  </h3>
  <p id="shipping-answer" data-faq-answer hidden>
    결제 확인 후 순차적으로 발송됩니다.
  </p>
</article>

<article class="faq-item">
  <h3>
    <button type="button" data-faq-button
            aria-expanded="false"
            aria-controls="exchange-answer">
      교환할 수 있나요?
    </button>
  </h3>
  <p id="exchange-answer" data-faq-answer hidden>
    수령 후 안내된 기간 안에 접수할 수 있습니다.
  </p>
</article>
```

style.css에 저장할 코드

파일에 저장 아래 두 규칙을 `style.css` 의 기존 FAQ 규칙 다음에 추가합니다.

```
.faq-item button {
  padding: 0;
  border: 0;
  color: var(--color-ink);
  background: transparent;
  font: inherit;
  font-weight: 700;
  text-align: left;
  cursor: pointer;
}

.faq-item button:focus-visible {
  outline: 3px solid var(--color-accent);
  outline-offset: 4px;
}
```

저장 후 확인

Elements에서 각 버튼의 `aria-controls` 와 바로 아래 답변의 `id` 가 일치합니다. 답변은 `hidden` 때문에 닫혀 있고, 아직 이벤트 코드는 없으므로 질문을 눌러도 열리지 않는 것이 정상입니다.

코드 한 줄 설명

`aria-controls="shipping-answer"` 는 이 버튼이 어떤 답변을 제어하는지 이름으로 연결합니다.

☐ **두 답변이 닫혀 있고 Tab 키로 질문에 초점을 옮겼을 때 테두리가 보이면 1단계 완료입니다.**

---

2. 클릭할 때 열릴 상태 계산

지금 따라 하기 · script.js

1. **위치:** 내 작업 폴더의 `script.js` 를 엽니다.
2. **행동:** 파일 안의 내용을 모두 아래 관찰용 코드로 교체합니다.
3. **저장:** 페이지를 새로고침하고 질문을 누른 뒤 Console 값을 봅니다.

실행 전 예상

이 단계에서는 `aria-expanded` 를 아직 바꾸지 않습니다. 처음 값이 `false` 이므로 클릭할 때 계산한 `willOpen` 을 Console에서 관찰하면 `true` 가 나올 것입니다.

script.js 전체를 교체할 코드

파일에 저장 · 관찰용 중간 단계 화면을 바꾸기 전, 다음 상태 계산이 맞는지 Console로 확인하는 코드입니다.

```
const buttons = document.querySelectorAll('[data-faq-button]');

for (const button of buttons) {
  button.addEventListener('click', () => {
    const answer = document.getElementById(
      button.getAttribute('aria-controls')
    );
    const willOpen = button.getAttribute('aria-expanded') === 'false';
    console.log(willOpen);
  });
}
```

저장 후 확인

이 단계는 계산만 하고 상태를 저장하지 않으므로 화면은 바뀌지 않고 `aria-expanded` 도 `false` 에 머뭅니다. 따라서 버튼을 반복해서 클릭해도 Console에는 계속 `true` 가 보입니다.

코드 한 줄 설명

`const willOpen = ... === 'false';` 는 현재 닫혀 있는지를 비교해 다음 상태를 참 또는 거짓으로 저장합니다.

☐ **질문을 누를 때 Console에 `true` 가 보이고 답변은 아직 닫힌 채라면 2단계 완료입니다.**

3. 접근성 상태와 화면을 함께 변경

지금 따라 하기 · script.js

1. **위치:** 방금 저장한 `script.js`를 그대로 엽니다.
2. **행동:** 2단계의 관찰용 코드를 모두 지우고 아래 최종 코드로 교체합니다.
3. **저장:** 페이지를 새로고침하고 두 질문을 각각 두 번씩 눌러 봅니다.

실행 전 예상

열기로 계산되면 `aria-expanded`는 `true`가 되고 답변의 `hidden`은 `false`가 되어야 합니다. 닫을 때는 둘 다 반대 상태가 되어야 합니다.

script.js 전체를 교체할 최종 코드

파일에 저장 · 최종 구현 이 코드가 이번 주 완성본에 남아야 합니다.

```
for (const button of document.querySelectorAll('[data-faq-button]')) {
  button.addEventListener('click', () => {
    const answer = document.getElementById(button.getAttribute('aria-controls'));
    const willOpen = button.getAttribute('aria-expanded') === 'false';
    button.setAttribute('aria-expanded', String(willOpen));
    answer.hidden = !willOpen;
  });
}
```

저장 후 확인

질문을 누르면 상태를 실제로 갱신해 해당 답변이 열립니다. 열린 상태에서 같은 질문을 다시 누르면 다음 열림 상태는 `false`가 되어 답변이 닫힙니다. 버튼의 `aria-expanded`와 답변의 `hidden`이 항상 반대 방향으로 함께 바뀝니다.

코드 한 줄 설명

`answer.hidden = !willOpen;` 은 열 예정이면 숨김을 끄고, 닫을 예정이면 숨김을 켭니다.

☐ **두 질문이 각각 열리고 다시 닫히며, Console에 빨간 오류가 없으면 3단계 완료입니다.**

`classList` 와 `hidden` 을 구분하세요.
`classList` 는 강조색 같은 CSS class를 추가하거나 제거할 때 쓸 수 있습니다. 이 예제는 답변의 표시 여부를 나타내는 HTML `hidden` 속성을 직접 바꾸므로 별도의 숨김 class를 만들지 않습니다.

// 6. AI와 기능 수정

현재 완성본은 각 질문을 독립적으로 열고 닫습니다. 따라서 첫 답변을 연 뒤 두 번째 답변을 누르면 **두 답변이 동시에 열릴 수 있습니다.** 이것이 실제 `week-4-javascript/script.js` 의 기본 동작입니다.

다음 실습에서만 요구사항을 "한 번에 하나"로 바꾸고, 기본 완성본과 AI 수정안을 구분해 비교합니다.

AI에 그대로 전달할 프롬프트
현재 FAQ 기능은 그대로 유지하면서 한 번에 하나의 답변만 열리도록 script.js만 수정해줘. 변경되는 줄과 상태가 어떻게 달라지는지 먼저 설명하고 HTML과 CSS는 수정하지 마.

1. 예상 수정 범위

- `script.js` 만 바꾼다.
- 새 질문을 열 때 다른 버튼과 답변을 닫는다.
- `index.html` 과 `style.css` 는 바꾸지 않는다.

2. AI 설명과 변경 줄 비교

- 다른 버튼을 찾는 범위가 FAQ 버튼으로 제한되는지 본다.
- 다른 버튼의 `aria-expanded` 와 답변의 `hidden` 을 함께 바꾸는지 본다.
- 기존 클릭 열기·닫기 줄을 불필요하게 지우지 않았는지 본다.

3. 실제 브라우저에서 시험

- 첫 질문을 열고 두 번째 질문을 연다.
- 첫 답변이 닫히고 두 번째 답변만 열리는지 본다.
- 같은 질문을 다시 눌러 모두 닫을 수 있는지 본다.

AI 수정안의 한 가지 예

아래는 기본 완성본이 아니라 **AI 수정안에서 한 번에 하나만 열리게 만든 이후 상태**입니다. 중첩 반복문이 들어가는 심화 예시이므로 직접 외워 작성할 필요는 없습니다. "새 답변을 열기 전에 다른 답변을 닫는다"는 흐름과 변경 범위만 확인합니다.

심화 읽기 · 수업 필수 구현 아님
기본 FAQ 열기·닫기가 정상 동작한 학습자만 아래 수정안을 시험합니다. 막히면 기본 완성본으로 돌아가도 이번 주의 핵심 목표를 달성한 것입니다.

선택 심화 코드 먼저 읽으며 기본 완성본과 다른 줄을 찾습니다. 시험하려면 현재 `script.js` 를 별도 복사해 둔 뒤에만 교체합니다.

```
const buttons = document.querySelectorAll('[data-faq-button]');

for (const button of buttons) {
  button.addEventListener('click', () => {
    const answer = document.getElementById(button.getAttribute('aria-controls'));
    const willOpen = button.getAttribute('aria-expanded') === 'false';

    if (willOpen) {
      for (const otherButton of buttons) {
        const otherAnswer = document.getElementById(
          otherButton.getAttribute('aria-controls')
        );
        otherButton.setAttribute('aria-expanded', 'false');
        otherAnswer.hidden = true;
      }
    }

    button.setAttribute('aria-expanded', String(willOpen));
    answer.hidden = !willOpen;
  });
}
```

AI 결과를 채택하는 기준
설명이 예상 수정 범위와 맞고, 변경된 줄이 `script.js` 에만 있으며, 브라우저에서 클릭·Enter·Space와 상태 동기화가 모두 확인될 때만 저장합니다. 전체 파일을 다시 썼다면 변경 범위를 줄여 다시 요청합니다.

// 7. 브라우저 콘솔

화면이 예상과 다르면 개발자 도구의 Console과 Network를 먼저 확인합니다. 브라우저마다 문구와 줄 번호가 조금 다르므로 메시지를 외우기보다 파일명, 줄, 토큰, 요청 상태 같은 단서를 봅니다. 아래 세 오류만 차례로 재현하고 바로 원상 복구합니다.

오류 실습 순서 · 한 번에 하나만

1. 현재 정상 동작을 먼저 확인합니다.
2. 1번 오류만 만들고 저장·새로고침한 뒤 Console 또는 Network의 단서를 찾습니다.
3. 반드시 원래 코드로 복구하고 정상 동작을 확인한 뒤 2번으로 넘어갑니다.
4. 3번까지 마친 뒤 최종 코드와 비교하고 Console의 빨간 오류가 없는지 확인합니다.

1. 잘못된 script.js 경로

`<script src="scripts.js" defer></script>` 처럼 실제 파일명과 다르게 바꿔 새로고침합니다.

콘솔에서 보이는 단서

Network에는 보통 `scripts.js` 요청이 `404` 로 나타나고, Console에는 “Failed to load resource”와 비슷한 문구가 보일 수 있습니다. 정확한 표현은 브라우저에 따라 다릅니다.

원인 위치

`index.html` 의 `script` 요소에서 `src` 파일명과 실제 폴더의 `script.js` 를 비교합니다.

수정 후 확인

`src="script.js"` 로 되돌리고 저장·새로고침합니다. Network가 성공하고 FAQ 클릭이 다시 작동하는지 확인합니다.

2. 오타 선택자

`[data-faq-button]` 을 `[data-faq-buton]` 처럼 바꾸고 새로고침합니다.

콘솔에서 보이는 단서

유효한 선택자이지만 일치하는 요소가 없어 빨간 오류가 나타나지 않을 수 있습니다. Console에서 `document.querySelectorAll('[data-faq-buton]').length` 를 실행하면 `0` 이 나오는 것이 단서입니다.

원인 위치

`script.js` 의 선택자 철자와 `index.html` 버튼의 `data-faq-button` 철자를 글자 단위로 비교합니다.

수정 후 확인

선택자를 `[data-faq-button]` 으로 되돌립니다. Console에서 길이가 `2` 인지 확인한 뒤 두 버튼을 각각 눌러 봅니다.

3. 닫히지 않은 괄호

`button.addEventListener('click', () => {` 의 마지막 닫는 `);` 에서 오른쪽 괄호 하나를 지우고 새로고침합니다.

콘솔에서 보이는 단서

`SyntaxError` 와 함께 닫는 괄호나 예상하지 못한 토큰을 가리키는 문구가 보일 수 있습니다. 표시 줄은 실제 원인 줄보다 뒤일 수 있고 브라우저마다 표현이 다릅니다.

원인 위치

Console이 가리키는 `script.js` 줄 주변에서 여는 `(` 와 닫는 `)`, 여는 `{` 와 닫는 `}` 를 짝으로 확인합니다.

수정 후 확인

괄호를 복구하고 저장·새로고침합니다. SyntaxError가 사라지고 첫 질문을 열고 닫을 수 있는지 확인합니다.

□ **세 오류를 하나씩 재현하고 모두 원상 복구했으며 FAQ가 다시 열리고 닫히면 디버깅 실습 완료입니다.**

// 8. 요구사항과 결과 비교

코드가 저장되었다는 사실보다 실제 요구가 재현되는지가 중요합니다. 기본 완성본은 질문별 독립 열기·닫기까지, AI 수정안을 적용한 버전은 한 번에 하나만 열기까지 요구사항으로 구분합니다.

상태	예상 동작	확인 결과
기본 완성본	각 질문이 독립적으로 열리고 닫히며 두 답변이 함께 열릴 수 있음	실제 `week-4-javascript/script.js` 및 기본 예제와 일치
AI 수정안	새 답변을 열면 기존 답변이 닫히고 한 번에 하나만 열림	수정 코드를 적용한 뒤 별도로 브라우저에서 검증

최종 QA 다섯 항목

1. 각 FAQ 버튼을 마우스로 **클릭**하고, 키보드 포커스 상태에서 **Enter**와 **Space**로 같은 동작을 사용할 수 있는가?
2. 각 버튼의 `aria-expanded` 값이 답변이 보이거나 숨겨진 **화면 상태**와 일치하는가?
3. 개발자 도구를 **390px** 너비로 맞췄을 때 제목, 상품 이미지, 버튼, FAQ 내용이 가로로 **잘리지** 않는가?
4. 상품 이미지의 `alt` 가 색과 형태를 포함해 **상품의 의미**를 설명하는가?
5. **JavaScript 오류**가 발생해도 상품명, 설명, 가격, 특징 같은 핵심 **상품 정보**는 HTML에서 계속 **읽**을 수 있는가?

4주 프로젝트 정리

HTML은 정보 구조, CSS는 모양과 반응형 배치, JavaScript는 사용자 행동에 따른 상태 변경을 담당합니다. AI는 이 세 역할을 대신 결정하는 도구가 아니라, 수정 범위를 설명하고 제안하는 협업 도구이며 최종 판단은 코드 비교와 브라우저 검증으로 내립니다.

// 9. JavaScript와 Liquid 비교

Liquid는 Shopify 테마에서 상점 데이터를 HTML에 넣기 위한 템플릿 언어입니다. 이번 교재에서 다룬 **브라우저 JavaScript**는 완성된 HTML을 받은 뒤 클릭 같은 사용자 행동에 반응합니다. 따라서 둘은 경쟁 관계가 아니라 **서로 다른 시점의 일을 나누어 맡는 도구**입니다.

Liquid: 보내기 전에 내용 만들기

- **실행 위치**: Shopify 서버의 테마 렌더링 단계
- **주요 입력**: 상품, 컬렉션, 장바구니, 테마 설정 같은 상점 데이터
- **주요 역할**: 상품명·가격을 출력하고 조건에 따라 만들 HTML을 결정
- **브라우저가 받는 결과**: Liquid 문법이 아니라 생성된 HTML

JavaScript: 받은 뒤 행동에 반응하기

- **실행 위치**: 이 교재에서는 사용자의 브라우저
- **주요 입력**: DOM 요소, 클릭·키보드 이벤트, 현재 화면 상태
- **주요 역할**: FAQ를 열고 닫거나 화면 상태와 접근성 속성을 변경
- **사용자가 보는 결과**: 새로고침 없이 바뀐 DOM과 화면

한 상품 페이지에서 실행되는 순서

1 Shopify 서버
Liquid가 상품 데이터와 조건을 읽어 HTML을 만듭니다.

2 브라우저 도착
브라우저가 생성된 HTML을 읽고 CSS와 JavaScript를 불러옵니다.

3 사용자 행동
사용자가 버튼을 누르면 JavaScript가 DOM과 화면 상태를 바꿉니다.

같은 상품 정보를 코드로 비교하기

하고 싶은 일	Liquid	브라우저 JavaScript
상품명 출력	`{{ product.title }}`	이미 생성된 제목을 DOM에서 찾고 읽음
가격 표시	`{{ product.price | money }}`	표시된 가격을 읽거나 사용자 선택에 따라 화면을 갱신
판매 가능 여부에 따른 HTML	`{% if product.available %}` 조건으로 만들 내용을 결정	페이지가 열린 뒤 발생한 이벤트와 현재 상태에 반응

```
<h1>{{ product.title }}</h1>
<p>{{ product.price | money }}</p>

{% if product.available %}
  <button type="button">구매하기</button>
{% else %}
  <p>현재 품절입니다.</p>
{% endif %}
```

`{{ ... }}`는 값을 HTML에 **출력**하고, `{% ... %}`는 조건문처럼 출력 흐름을 **제어**합니다. 위 코드는 Shopify 서버에서 처리되므로 브라우저는 최종 상품명·가격·버튼 또는 품절 문구가 들어간 HTML을 받습니다.

한 문장으로 구분하기
Liquid는 “처음 어떤 HTML을 보낼지” 결정하고, **JavaScript는 “받은 화면에서 사용자가 행동한 뒤 무엇을 바꿀지”** 결정합니다. 일반 웹의 HTML·CSS·JavaScript를 Shopify 테마 폴더에 연결하는 자세한 내용은 Shopify 실무 연결 가이드에서 이어집니다.

공식 문서: Liquid 문법 · Shopify 테마의 구성 언어