14. HTML5 심화 주제

14.1 `canvas`와 2D 그래픽

— HTML5에서 픽셀 단위로 2D 그래픽을 그리는 가장 낮은 수준의 API

`<canvas>` 요소는 HTML 문서 내에서 **픽셀 기반의 그래픽을 직접 그릴 수 있는 영역**을 제공한다.
이는 정적인 HTML 콘텐츠와 달리 **동적 이미지, 게임, 데이터 시각화, 애니메이션 등**을 위한 핵심 도구다.

✅ 1. 기본 구조

```
<canvas id="myCanvas" width="400" height="300"></canvas>
```

📌 `<canvas>`는 **자체적으로 아무것도 그리지 않음**
→ JavaScript로 `context`를 통해 그림을 그려야 함.

✅ 2. 컨텍스트 객체 얻기 (`2d`)

```
const canvas = document.getElementById("myCanvas");
const ctx = canvas.getContext("2d"); // 2D 컨텍스트
```

- `ctx`는 2D 그래픽 API의 모든 메서드와 속성을 가진 객체

✅ 3. 기본 도형 그리기

🔷 사각형

```
ctx.fillStyle = "blue";
ctx.fillRect(20, 20, 150, 100); // x, y, width, height
```

메서드	설명
`fillRect(x, y, w, h)`	채워진 사각형
`strokeRect(x, y, w, h)`	테두리만
`clearRect(x, y, w, h)`	투명하게 지우기

🔷 선 그리기

```
ctx.beginPath();
ctx.moveTo(10, 10);
ctx.lineTo(200, 150);
ctx.stroke();
```

🔷 원/호 그리기

```
ctx.beginPath();
ctx.arc(100, 75, 50, 0, Math.PI * 2); // x, y, radius, startAngle, endAngle
ctx.fill();
```

✅ 4. 색상, 선 스타일, 채우기

```
ctx.fillStyle = "red";
ctx.strokeStyle = "black";
ctx.lineWidth = 3;
```

- `fillStyle` → 내부 채움 색상
- `strokeStyle` → 외곽선 색상
- `lineWidth` → 선 두께

✅ 5. 텍스트 그리기

```
ctx.font = "20px Arial";
ctx.fillText("Hello Canvas", 50, 50);
ctx.strokeText("Outlined Text", 50, 100);
```

✅ 6. 이미지 그리기

```
const img = new Image();
img.src = "image.png";
img.onload = () => {
  ctx.drawImage(img, 10, 10, 100, 100); // (img, x, y, w, h)
};
```

✅ 7. 캔버스 애니메이션 (기본 개념)

```
function draw() {
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.fillRect(x++, 50, 50, 50);
  requestAnimationFrame(draw); // 60fps 애니메이션 루프
}
draw();
```

✅ 8. 캔버스 특징

특징	설명
픽셀 기반 렌더링	확대/축소 시 해상도 손실 있음 (비벡터)
스크린 리더 지원 X	접근성 낮음 → `aria-hidden="true"` 고려
JavaScript 필요	HTML만으로는 시각적 결과 없음
상태 기반 API	`ctx`의 상태를 계속 설정/변경하면서 그림

✅ 9. 고급 활용 사례

- 게임 개발 (예: Pong, Asteroids)
- 데이터 시각화 (차트, 그래프 → Chart.js 내부도 Canvas 사용)
- 손글씨/서명 입력
- 드로잉 앱
- 2D 시뮬레이션/물리 엔진

✅ 10. 접근성 고려

> `<canvas>` 안의 컨텐츠는 시각적으로만 존재하므로, **대체 콘텐츠 제공 필수**

```
<canvas id="chartCanvas" width="300" height="200">
  당신의 브라우저는 캔버스를 지원하지 않습니다. 이 그래프는 2023년 매출 현황을 시각화한 것입니다.
</canvas>
```

또는 시각적 정보는 별도로 `<table>`이나 `<figcaption>` 등으로 제공해야 한다.

✅ 요약

> `<canvas>`는 HTML5에서 **픽셀 단위의 2D 그래픽을 동적으로 그리는 도구**로,
> **게임, 애니메이션, 시각화 등에 유용하지만 접근성 고려가 부족하므로 대체 정보 제공이 중요**하다.

14.2 `svg`와 벡터 그래픽

— 해상도에 독립적인 구조 기반 그래픽: Scalable Vector Graphics

`<svg>`(Scalable Vector Graphics)는 XML 기반의 벡터 그래픽 표준으로, HTML 내에서 선, 도형, 곡선, 텍스트 등을 해상도 손실 없이 표현할 수 있다.

캔버스(`canvas`)와 달리 **DOM 요소로 직접 접근이 가능하며**,
스타일(CSS) 적용, 애니메이션 처리, 접근성 대응이 우수한 특징을 가진다.

✅ 1. 기본 구조

```svg
<svg width="200" height="100" xmlns="http://www.w3.org/2000/svg">
  <rect x="10" y="10" width="180" height="80" fill="skyblue" stroke="black"/>
</svg>
```

요소	의미
`<svg>`	SVG 컨테이너
`<rect>`	사각형 도형
`x`, `y`	좌상단 좌표
`width`, `height`	크기
`fill`, `stroke`	색상, 테두리

✅ 2. 주요 도형 요소

태그	설명
`<rect>`	사각형
`<circle>`	원
`<ellipse>`	타원
`<line>`	직선

태그	설명
`<polyline>`	여러 점을 연결한 꺾은선
`<polygon>`	닫힌 다각형
`<path>`	자유곡선 (베지어 포함)

📌 예시:

```
<circle cx="50" cy="50" r="40" fill="orange" />
<line x1="10" y1="10" x2="200" y2="50" stroke="red" stroke-width="2"/>
<polygon points="50,10 90,90 10,90" fill="lime" />
```

✅ 3. 텍스트 추가

```
<svg width="300" height="100">
  <text x="20" y="50" font-size="20" fill="black">Hello SVG</text>
</svg>
```

- 텍스트에도 CSS와 마찬가지로 `font-size`, `fill`, `font-family` 적용 가능
- `<textPath>`로 곡선을 따라 텍스트 배치도 가능

✅ 4. CSS 스타일 적용

SVG 요소는 DOM이므로 클래스, 스타일시트 적용이 가능하다.

```
<svg>
  <circle class="dot" cx="30" cy="30" r="20" />
</svg>

<style>
  .dot {
    fill: tomato;
    stroke: black;
    stroke-width: 3;
  }
</style>
```

✅ 5. 인터랙션 및 애니메이션

```svg
<circle cx="50" cy="50" r="30" fill="blue">
  <animate attributeName="r" from="30" to="50" dur="1s" repeatCount="indefinite" />
</circle>
```

- `animate`, `animateTransform`, `set` 등을 이용한 SVG 전용 애니메이션 가능
- JavaScript나 CSS `@keyframes` 애니메이션도 사용 가능

✅ 6. 외부 파일로 분리 (`.svg`)

SVG는 `.svg` 파일로 저장하여 다음과 같이 사용 가능:

```html
<img src="logo.svg" alt="로고">
<object data="chart.svg" type="image/svg+xml"></object>
<iframe src="graphic.svg"></iframe>
```

✅ 7. 접근성 고려

요소	설명
`role="img"`	SVG에 이미지 역할 명시
`<title>`, `<desc>`	보조 설명 제공
`aria-labelledby`	제목과 설명 연결

```svg
<svg role="img" aria-labelledby="svgTitle svgDesc">
  <title id="svgTitle">사용자 성장 그래프</title>
  <desc id="svgDesc">2022년부터 2024년까지 사용자 수 변화</desc>
  <path d="..." />
</svg>
```

✅ 8. SVG의 장점 vs 단점

장점	단점
해상도 독립적 (벡터)	복잡한 애니메이션엔 성능 저하 가능
DOM 직접 조작 가능	큰 SVG 파일은 파싱 부담 있음

장점	단점
CSS/JS로 쉽게 제어	픽셀 단위 디테일 조정은 어려움
접근성 우수	복잡한 Path 문법은 러닝 커브

✅ 9. 캔버스(Canvas)와 비교

기준	SVG	Canvas
렌더링 방식	벡터	픽셀
확대 시 품질	유지됨	깨짐
요소 접근	DOM	불가능 (픽셀 레벨)
스타일 제어	CSS, JS	JS만 가능
상호작용	이벤트 가능	수동 구현
추천 용도	아이콘, UI, 다이어그램	게임, 애니메이션, 대용량 그래픽

✅ 실전 예시: 원형 차트

```
<svg width="200" height="200" viewBox="0 0 200 200">
  <circle cx="100" cy="100" r="90" fill="#eee" />
  <circle cx="100" cy="100" r="90"
          stroke="tomato" stroke-width="20" fill="none"
          stroke-dasharray="565.48" stroke-dashoffset="282.74"
          transform="rotate(-90 100 100)" />
</svg>
```

📌 `stroke-dasharray`와 `dashoffset`으로 그래프 구현

✅ 요약 정리

> `<svg>`는 **고해상도 벡터 그래픽을 HTML에서 직접 표현할 수 있는 태그**이며,
> **CSS 및 JS와의 통합이 뛰어나고, 접근성, 반응성, 인터랙션에 적합**하다.
> 차트, 아이콘, UI 요소, 애니메이션, 시각화 등에서 널리 사용된다.

14.3 `contenteditable`을 이용한 실시간 편집

— HTML 요소를 브라우저에서 직접 수정할 수 있게 만드는 속성

`contenteditable` 속성은 HTML의 **모든 요소를 실시간 편집 가능하게 만들어주는 전역 속성**이다.
이 속성이 설정된 요소는 사용자가 마우스로 선택한 후 **바로 타이핑**하여 내용을 수정할 수 있으며,
JavaScript와 결합하여 간단한 **위지윅(WYSIWYG) 에디터**도 만들 수 있다.

✅ 1. 기본 사용법

```
<div contenteditable="true">
  여기를 직접 수정할 수 있습니다.
</div>
```

- `"true"` 또는 빈 문자열(`""`) → 편집 가능
- `"false"` → 편집 불가능

```
<p contenteditable>이 문단은 편집이 가능합니다.</p>
```

✅ 2. 실시간 편집 예제

```
<h2 contenteditable>제목을 클릭해서 편집해 보세요</h2>
<p contenteditable="true">이 텍스트는 브라우저 상에서 수정할 수 있습니다.</p>
```

사용자는 이 페이지에서 마치 워드처럼 직접 내용을 바꾸고 수정할 수 있음.

✅ 3. JavaScript로 변경 감지

```
<div id="editor" contenteditable="true">내용을 입력하세요</div>

<script>
  const editor = document.getElementById('editor');
  editor.addEventListener('input', () => {
    console.log('변경된 내용:', editor.innerHTML);
  });
</script>
```

- `input` 이벤트는 사용자가 타이핑할 때마다 발생

- `innerHTML`, `innerText`, `textContent` 등을 통해 편집 결과 접근 가능

✅ 4. 스타일링 팁

```
[contenteditable="true"] {
  padding: 10px;
  border: 1px solid #ccc;
  outline: none;
  min-height: 100px;
  background-color: #fefefe;
}
```

- `outline: none` 설정으로 포커스 테두리를 제거
- `min-height`로 에디터처럼 넓이 확보

✅ 5. 브라우저 내장 명령어와 함께 사용 (deprecated 주의)

```
document.execCommand('bold'); // 선택한 텍스트 굵게
document.execCommand('insertText', false, '삽입된 텍스트');
```

> `execCommand()`는 현재는 **Deprecated** 되었지만 일부 WYSIWYG 에디터는 여전히 사용 중

✅ 6. 실제 에디터 제작 기본 구조

```
<div id="toolbar">
  <button onclick="document.execCommand('bold')">Bold</button>
  <button onclick="document.execCommand('italic')">Italic</button>
</div>

<div id="editor" contenteditable="true">
  여기에 내용을 입력하세요.
</div>
```

✅ 7. 보안 주의사항

- 사용자가 `<script>`, `<iframe>`, `<img onerror>` 등의 **XSS 코드**를 입력할 수 있음
- **백엔드 전송 전 반드시 정제(Sanitization)** 필요
  → DOMPurify 같은 라이브러리 권장

```
// 예시: DOMPurify를 이용한 정제
const clean = DOMPurify.sanitize(editor.innerHTML);
```

✅ 8. 실전 활용 예시

사용 사례	설명
간단한 노트 앱	텍스트 편집 후 저장
댓글/답글 인라인 편집	수정 버튼 클릭 시 contenteditable 적용
실시간 협업 에디터	Firebase, WebSocket과 결합
블로그/문서 작성 도구	마크다운 변환기와 결합
디자인/문서 템플릿 도구	동적으로 요소 텍스트 편집

✅ 접근성 팁 (스크린리더, 키보드 사용자)

```
<div contenteditable="true" role="textbox" aria-multiline="true" aria-label="내용 입력 영역">
  여기에 입력하세요.
</div>
```

- `role="textbox"`: 입력 필드임을 알림
- `aria-multiline="true"`: 여러 줄 가능 여부
- `aria-label`: 이름 명시

✅ 요약 정리

> `contenteditable`은 HTML 요소를 브라우저에서 실시간으로 편집할 수 있게 해주며,
> 간단한 에디터부터 복잡한 문서 작성 도구까지 다양하게 활용 가능하다.
> 단, 입력 내용은 반드시 **보안 필터링**을 거쳐야 한다.

14.4 Web Storage (localStorage, sessionStorage)

— 브라우저에 데이터를 클라이언트 측에 영구 또는 일시 저장하는 표준 API

Web Storage API는 HTML5부터 제공되는 **로컬 데이터 저장 방식**으로,
쿠키와 달리 서버에 전송되지 않으며, **속도와 용량 면에서 훨씬 효율적**이다.

✅ 1. 종류 비교: `localStorage` vs `sessionStorage`

항목	`localStorage`	`sessionStorage`
저장 위치	브라우저 내부 (디스크)	탭의 메모리
유지 기간	**무기한**	**탭 종료 시 삭제**

항목	`localStorage`	`sessionStorage`
탭 공유	모든 탭 공유	탭마다 별개
용량 제한	약 5~10MB	약 5~10MB
사용 예	자동 로그인, 임시 초안 저장	탭 간 임시 상태 유지

✅ 2. 데이터 저장 및 읽기 (공통 API)

🔹 저장 (문자열만 가능)

```
localStorage.setItem("username", "jeongseok");
sessionStorage.setItem("temp", "12345");
```

🔹 읽기

```
const name = localStorage.getItem("username");
const temp = sessionStorage.getItem("temp");
```

🔹 삭제

```
localStorage.removeItem("username");
sessionStorage.clear(); // 모든 항목 삭제
```

✅ 3. JSON 데이터 저장 방법

Web Storage는 문자열만 저장 가능하므로 객체는 JSON으로 처리해야 한다.

```
const user = {
  id: 1,
  name: "정석",
  age: 32
};

localStorage.setItem("user", JSON.stringify(user));

const parsedUser = JSON.parse(localStorage.getItem("user"));
console.log(parsedUser.name); // "정석"
```

✅ 4. 실전 예제: 실시간 메모장 + 자동 저장

```
<textarea id="memo" rows="10" cols="50" placeholder="메모 입력..."></textarea>
```

```
const memo = document.getElementById("memo");

// 저장된 메모 불러오기
memo.value = localStorage.getItem("autosave") || "";

// 입력 시마다 저장
memo.addEventListener("input", () => {
  localStorage.setItem("autosave", memo.value);
});
```

- 새로고침해도 **입력한 메모가 유지됨**
- 세션 저장으로 바꾸려면 `sessionStorage` 사용

✅ 5. 브라우저 스토리지 확인 방법

- Chrome: `F12 → Application → Local Storage / Session Storage`
- Firefox: `F12 → Storage`

✅ 6. 보안 이슈 주의사항

항목	내용
민감 정보 저장 금지	비밀번호, 토큰, 주민번호 등은 저장 X
XSS 공격에 취약	스크립트 삽입 공격으로 탈취될 수 있음
보안 저장이 필요하다면?	서버에 저장하거나, 암호화 + 안전한 저장 방식 사용

예:

```
const encrypted = btoa("secret"); // 단순 Base64 (강력한 보안은 아님)
localStorage.setItem("token", encrypted);
```

✅ 7. 이벤트 감지: `storage` 이벤트

다른 탭에서 `localStorage`가 바뀌면 감지할 수 있음

```
window.addEventListener("storage", function (e) {
  console.log("변경된 키:", e.key);
  console.log("새 값:", e.newValue);
});
```

📌 `sessionStorage`는 탭 간 공유되지 않으므로 이 이벤트로 감지할 수 없음

✅ 8. 실전 활용 사례

사례	설명
다크 모드 설정 저장	`localStorage.setItem("theme", "dark")`
최근 본 상품 목록	배열 → JSON으로 저장
임시 작성 내용 자동 저장	입력 중 자동으로 저장하고 복원
게임 점수, 레벨	간단한 게임의 상태 기억
탭 간 통신 감지	`storage` 이벤트를 이용해 탭 동기화

✅ 요약

> `localStorage`와 `sessionStorage`는 브라우저에 **클라이언트 측 데이터를 저장**하는 방법으로,
> 빠르며 서버에 부담을 주지 않지만, **보안과 용량 제한**을 고려해 사용해야 한다.
> JSON으로 구조화된 데이터를 저장하는 것이 일반적이다.

14.5 `<template>` 태그와 동적 콘텐츠 삽입

— HTML 구조를 숨겨두고, JavaScript로 동적으로 삽입하는 방법

`<template>` 태그는 **브라우저에 렌더링되지 않는 HTML 조각을 미리 정의**해 두고,
JavaScript를 통해 **필요한 시점에 동적으로 DOM에 삽입할 수 있게 해주는 기능**이다.

이는 복잡한 UI를 JavaScript로 동적으로 생성할 때 코드 중복을 줄이고, HTML 구조를 분리해 가독성을 높이는 데 매우 유용하다.

✅ 1. 기본 구조

```
<template id="my-template">
  <div class="box">
    <h3 class="title">템플릿 제목</h3>
    <p class="content">이 내용은 복사되어 사용됩니다.</p>
  </div>
</template>
```

📌 이 코드는 **브라우저 상에는 아무것도 표시되지 않음**.

✅ 2. JavaScript로 템플릿 복사 및 삽입

```
<button id="add">템플릿 추가</button>
<div id="container"></div>
```

```
document.getElementById("add").addEventListener("click", () => {
  const template = document.getElementById("my-template");
  const clone = template.content.cloneNode(true); // deep copy
  document.getElementById("container").appendChild(clone);
});
```

📌 핵심 메서드:

메서드	설명
`.content`	`<template>` 내부의 문서 조각 (DocumentFragment)
`.cloneNode(true)`	전체 구조를 깊은 복사
`.appendChild()`	DOM에 삽입

✅ 3. 템플릿 안에 동적 데이터 넣기

```
const clone = template.content.cloneNode(true);
clone.querySelector(".title").textContent = "새 제목";
clone.querySelector(".content").textContent = "이건 동적으로 설정된 내용입니다.";
container.appendChild(clone);
```

✅ 4. 반복 데이터와 함께 쓰기 (예: 게시글 목록, 카드 UI)

```
const data = [
  { title: "공지사항 1", body: "내용 A" },
  { title: "공지사항 2", body: "내용 B" }
];

data.forEach(item => {
  const clone = template.content.cloneNode(true);
  clone.querySelector(".title").textContent = item.title;
  clone.querySelector(".content").textContent = item.body;
  container.appendChild(clone);
});
```

✅ 5. 조건부 렌더링에도 활용 가능

```
if (user.isAdmin) {
  const adminTemplate = document.getElementById("admin-template");
  container.appendChild(adminTemplate.content.cloneNode(true));
}
```

✅ 6. `<template>`을 사용하는 이유

전통 방식 (innerHTML 등)	`<template>` 방식
문자열 기반 처리 → 복잡함	DOM 구조 그대로 관리 가능
보안 취약 (XSS 위험 ↑)	DOM으로 안전하게 삽입
코드 재사용 어려움	재사용 및 유지보수 쉬움
구조 확인 어려움	HTML에 명시적으로 구조 존재

✅ 7. 실전 예시: 댓글 컴포넌트

```
<template id="comment-template">
  <div class="comment">
    <strong class="author"></strong>
    <span class="message"></span>
  </div>
</template>
```

```
function addComment(author, msg) {
  const t = document.getElementById("comment-template");
  const node = t.content.cloneNode(true);
  node.querySelector(".author").textContent = author;
  node.querySelector(".message").textContent = msg;
  document.getElementById("comments").appendChild(node);
}
```

✅ 8. 한계와 주의점

- `<template>` 태그는 렌더링되지 않으므로 **JS가 반드시 있어야 작동**함
- `<template>` 내부는 **문자열 조작이 아닌 DOM 조작**이 기본 전제
- `<script>`, `<style>`은 포함 가능하지만, 동작하지 않음 → 외부 삽입 추천

✅ 9. `template`과 Shadow DOM, Web Component의 연계

- `<template>`은 **Web Components**에서 Shadow DOM 템플릿 정의에 사용됨
- 예: `customElements.define()` 와 결합해 **재사용 가능한 컴포넌트 UI** 제작 가능

```
class MyCard extends HTMLElement {
  constructor() {
    super();
    const shadow = this.attachShadow({ mode: "open" });
    const template = document.getElementById("card-template");
    shadow.appendChild(template.content.cloneNode(true));
  }
}

customElements.define("my-card", MyCard);
```

✅ 요약

> `<template>`은 **렌더링되지 않는 HTML 조각**을 정의해 두고,
> JavaScript로 **필요한 시점에 복제하여 DOM에 삽입**할 수 있는 강력한 도구이다.
> 컴포넌트 구조화, 반복 렌더링, UI 조립 등에 매우 유용하며,
> 앞으로의 Web Components 기반 프론트엔드에도 핵심 역할을 한다.

14.6 `<dialog>` 태그로 모달 구현

— HTML5에서 제공하는 기본 모달 다이얼로그 구현 태그

`<dialog>` 태그는 HTML5에서 추가된 요소로, **브라우저 기본 기능으로 모달/팝업 UI를 구현**할 수 있게 해준다.
기존에는 모달을 `<div>`로 만들고 CSS/JS로 조작해야 했지만, `<dialog>`는 **브라우저에서 모달 행동을 내장 처리**한다는 점이 특징이다.

✅ 1. 기본 구조

```
<dialog id="myDialog">
  <p>이것은 다이얼로그입니다.</p>
  <button onclick="document.getElementById('myDialog').close()">닫기</button>
</dialog>

<button onclick="document.getElementById('myDialog').showModal()">모달 열기</button>
```

📌 주요 메서드

메서드	설명
`.show()`	비모달(non-modal)로 열기
`.showModal()`	**모달(modal)**로 열기
`.close()`	닫기

✅ 2. 모달 vs 비모달 차이

구분	설명
`.show()`	다이얼로그가 열리지만 **백그라운드 클릭 가능**
`.showModal()`	다이얼로그가 열리고 **다른 UI 차단**

✅ 3. 닫기 버튼 외에 `esc`, 바깥 클릭으로 닫기

- 기본적으로 `<dialog>`는 `ESC` 키나 바깥 클릭으로 닫힘
- 바깥 클릭 닫기 방지하려면 `backdrop`을 감지해야 함 (별도 처리 필요)

```
dialog.addEventListener("cancel", (e) => {
  console.log("ESC나 바깥 클릭으로 닫힘");
});
```

✅ 4. 폼 전송과의 연계

`<dialog>`는 `<form method="dialog">`와 결합 시 폼 제출 없이 간단히 닫을 수 있다.

```
<dialog id="formDialog">
  <form method="dialog">
    <p>확인하시겠습니까?</p>
    <menu>
      <button value="cancel">취소</button>
      <button value="confirm">확인</button>
    </menu>
  </form>
</dialog>
```

```
formDialog.addEventListener("close", () => {
  console.log("선택된 값:", formDialog.returnValue);
});
```

✅ 5. 스타일링 예시 (모달처럼)

```css
dialog {
  border: none;
  border-radius: 10px;
  padding: 20px;
  box-shadow: 0 0 10px rgba(0,0,0,0.3);
}

::backdrop {
  background: rgba(0, 0, 0, 0.5);
}
```

- `::backdrop` → 모달 뒤 배경을 흐리게
- 브라우저 기본 스타일을 재정의 가능

✅ 6. `open` 속성으로 상태 확인

```js
const isOpen = dialog.open; // true 또는 false
```

✅ 7. 모달 열기/닫기 애니메이션 예제

```css
dialog[open] {
  animation: fadeIn 0.3s ease-out;
}

@keyframes fadeIn {
  from { opacity: 0; transform: scale(0.95); }
  to { opacity: 1; transform: scale(1); }
}
```

✅ 8. `<dialog>` 사용 시 주의점

항목	설명
브라우저 지원	대부분 모던 브라우저 지원 (IE 제외)
접근성	`role="dialog"`, `aria-labelledby`, `aria-describedby` 설정 권장
오래된 브라우저 대응	폴리필 필요 (예: `dialog-polyfill.js`)

```html
<dialog role="dialog" aria-labelledby="title" aria-describedby="desc">
  <h2 id="title">제목</h2>
  <p id="desc">설명 텍스트</p>
</dialog>
```

✅ 9. 다양한 실전 사례

상황	설명
삭제 확인 모달	`정말 삭제하시겠습니까?`
동의 폼	개인정보 처리방침, 이용 약관
사용자 입력 모달	이름, 이메일 등 입력 받기
이미지/카드 확대	상세 보기용

✅ 요약

> `<dialog>`는 HTML만으로 **모달 창을 매우 간단하고 접근성 있게 구현할 수 있는 태그**이다.
> 브라우저가 직접 UI 차단, 백그라운드 비활성화, ESC 동작 등을 처리해주므로
> 코드를 줄이고, 복잡한 모달 구현을 단순화할 수 있다.

14.7 `<details>`, `<summary>`로 접이식 구조 만들기

▶ 는 HTML5에서 도입된 토글 가능한 콘텐츠 블록을 만들기 위한 구조다. 별도 자바스크립트 없이도 펼치기/접기 기능이 내장되어 있어 FAQ, 설정 패널, 문서 목차 등에서 유용하게 쓰인다.

✅ 1. 기본 사용법

```
<details>
  <summary>자세히 보기</summary>
  <p>이 내용은 클릭 시 펼쳐집니다.</p>
</details>
```

- `<summary>`는 항상 `<details>` 태그 내 첫 번째 자식이어야 한다.
- 기본적으로 닫힌 상태로 표시됨.

✅ 2. 기본 동작 특징

동작	설명
클릭 시	`open` 속성이 자동으로 붙고 콘텐츠가 펼쳐짐
다시 클릭 시	`open` 속성이 제거되고 닫힘
`open` 속성 수동 설정	기본 상태를 "열림"으로 만들 수 있음

```html
<details open>
  <summary>처음부터 열림</summary>
  <p>페이지 로드시 열려 있음</p>
</details>
```

✅ 3. 스타일 커스터마이징

🔹 기본 스타일 커스터마이징 예시

```css
details {
  border: 1px solid #aaa;
  border-radius: 5px;
  padding: 10px;
  width: 300px;
}

summary {
  font-weight: bold;
  cursor: pointer;
}
```

🔹 기본 ▶️ 아이콘 숨기고 커스텀

```css
summary::-webkit-details-marker {
  display: none;
}
summary::before {
  content: "▶ ";
  display: inline-block;
  transform: rotate(0deg);
  transition: transform 0.2s;
}
details[open] summary::before {
  content: "▼ ";
}
```

✅ 4. JavaScript 이벤트 감지

```html
<details id="faq">
  <summary>FAQ</summary>
  <p>자주 묻는 질문입니다.</p>
</details>
```

```javascript
document.getElementById("faq").addEventListener("toggle", () => {
  console.log("펼침/접힘 상태:", faq.open);
});
```

- `.open` 속성으로 현재 상태를 확인 가능
- `toggle` 이벤트로 펼침/접힘 변화를 감지

✅ 5. 실전 활용 예: FAQ 리스트

```
<details>
  <summary>Q. HTML이란 무엇인가요?</summary>
  <p>A. HyperText Markup Language의 약자로...</p>
</details>

<details>
  <summary>Q. CSS는 왜 필요한가요?</summary>
  <p>A. HTML의 구조 외에 스타일을 정의하기 위해 사용합니다.</p>
</details>
```

✅ 6. 접근성(Accessibility) 측면

- `<details>` + `<summary>` 조합은 **기본적으로 접근성이 좋은 요소**
- 스크린 리더에서 `summary`는 자동으로 버튼처럼 읽힘
- 추가적으로 `aria-expanded`, `aria-controls` 속성도 함께 사용 가능

✅ 7. `<details>` vs JavaScript 토글 비교

항목	`<details>` 방식	JavaScript 토글 방식
코드 복잡도	매우 단순	조건문, 클래스 조작 필요
접근성	기본 제공	추가 처리 필요
커스터마이징	제한적	자유로움
제어 가능성	낮음 (이벤트 제한적)	자유로움

✅ 요약

> `<details>`와 `<summary>`는 자바스크립트 없이 **접이식 인터페이스를 간단하게 구현**할 수 있는 HTML 요소다.
> FAQ, 도움말, 목차, 설정 패널 등 **기본 UI 요소 구현에 매우 적합**하며
> 접근성도 기본 제공되기 때문에 가볍고 실용적인 선택지다.