# 14. 고급 기능 및 심화 주제

## 사용자 정의 속성 (CSS 변수) `--main-color`

CSS 사용자 정의 속성(Custom Properties), 일명 **CSS 변수**는 **재사용 가능하고 동적으로 변경 가능한 값**을 선언하고 사용하는 기능이다.
이 기능은 **유지보수성, 일관성, 테마 관리, 모듈화**에 매우 유리하다.

---

### 1. 변수 선언

CSS 변수는 `--이름` 형식으로 선언된다. 보통 **선언 범위에 따라** 변수는 **전역 또는 지역 변수**가 될 수 있다.

```
:root {
  --main-color: #3498db;
  --base-font-size: 16px;
}
```

- `:root`는 문서의 최상위 요소 (`html`)를 가리키며, 변수의 **전역 선언**에 사용된다.

---

### 2. 변수 사용

변수를 사용할 때는 `var(--이름)` 구문을 사용한다.

```
body {
  color: var(--main-color);
  font-size: var(--base-font-size);
}
```

> 선언된 값을 불러와 속성 값으로 사용

---

### 3. 기본값(fallback) 설정

변수에 값이 없을 경우를 대비해 **기본값**을 지정할 수 있다.

```
h1 {
  color: var(--undefined-color, black);
}
```

> `--undefined-color`가 없으면 `black`이 사용됨

---

## 4. 지역 변수

```
.box {
  --main-color: red;
  color: var(--main-color);  /* red */
}

.other {
  color: var(--main-color);  /* 전역 :root에서 선언된 값 사용됨 */
}
```

> 변수는 선언된 요소와 **그 하위 요소에만 적용**되므로 **지역 범위**로 사용할 수도 있다.

---

## 5. 예제 – 버튼 테마 설정

```
:root {
  --btn-bg: #2ecc71;
  --btn-color: white;
}

.button {
  background: var(--btn-bg);
  color: var(--btn-color);
  border: none;
  padding: 10px 20px;
  font-size: 1rem;
  border-radius: 4px;
}
```

---

## 6. 변수 기반 다크모드 예시

```
:root {
  --bg-color: #ffffff;
  --text-color: #000000;
}

[data-theme="dark"] {
  --bg-color: #1e1e1e;
  --text-color: #f0f0f0;
}

body {
  background-color: var(--bg-color);
  color: var(--text-color);
}
```

> `data-theme="dark"`를 토글하면 전체 테마 색상이 즉시 변경됨

---

## 7. 자바스크립트로 동적 변경

CSS 변수는 자바스크립트로 **런타임에 조작**할 수 있다.

```
document.documentElement.style.setProperty('--main-color', '#e74c3c');
```

> 전체 페이지에서 `--main-color`가 즉시 새로운 값으로 적용됨

---

## 8. 변수와 연산

CSS 변수는 직접적으로 **연산은 안 되지만**, `calc()` 안에서는 사용할 수 있다.

```
:root {
  --base-padding: 10px;
}

.box {
  padding: calc(var(--base-padding) * 2); /* 가능 */
}
```

---

## 9. 사용자 정의 속성 vs SASS 변수

| 비교 항목 | CSS 변수 ( `--` ) | SASS 변수 ( `$` ) |
|---|---|---|
| 런타임 동작 | O (실행 중 변경 가능) | X (컴파일 시 결정) |
| 자바스크립트 제어 | O | X |
| 중첩, 반복문 지원 | X | O |
| 브라우저 지원 | IE 제외 모두 지원 | 컴파일 후 일반 CSS |

---

## 결론

> CSS 사용자 정의 속성( `--main-color` 등)은
> 테마 관리, 재사용, 유지보수에 탁월한 기능으로, 현대 웹 개발에서 필수적인 스타일링 도구다.

특히 다크모드, 디자인 시스템 구축, 컴포넌트 라이브러리 등에서는 **CSS 변수 사용이 거의 표준화**되고 있다.

# `calc()` 함수

`calc()`는 CSS에서 **길이, 비율, 수치 값 등을 계산하여 동적으로 스타일 값을 결정**할 수 있는 함수다.
이를 통해 고정값과 상대값을 혼합하거나, **반응형 레이아웃**을 유연하게 만들 수 있다.

---

# 1. 기본 문법

```
속성: calc(수식);
```

```
width: calc(100% - 80px);
font-size: calc(1rem + 2px);
padding: calc(10px * 2);
```

- 사칙연산(`+`, `-`, `*`, `/`) 사용 가능
- **공백 필수**: 연산자 앞뒤에 공백을 넣어야 함 (`calc(100% - 20px)` ← O, `calc(100%-20px)` ← X)

---

# 2. 주요 사용 예

## A. 고정 너비를 제외한 나머지 영역 사용

```
.sidebar {
  width: 250px;
  float: left;
}

.content {
  width: calc(100% - 250px);
  float: left;
}
```

> 좌측에 250px 고정 사이드바가 있을 때, 나머지 영역을 자동으로 채움

---

## B. 반응형 폰트 설정

```
h1 {
  font-size: calc(1rem + 2vw);
}
```

> 화면 너비가 커질수록 글자가 커짐
> → `rem`과 `vw`를 혼합하여 유동적 타이포그래피 구현

---

## C. 가운데 정렬 시 여백 계산

```
.box {
  width: 400px;
  margin-left: calc(50% - 200px);
}
```

> `width`의 절반을 왼쪽 여백으로 지정하여 **수평 가운데 정렬** 수행

---

### D. 레이아웃 간격 조정

```
.container {
  padding: calc(1rem + 10px);
}
```

> 기본 패딩에 추가 여백을 가산하여 유연한 공간 확보

---

## 3. 단위 혼합 지원

`px`, `%`, `em`, `rem`, `vw`, `vh` 등을 자유롭게 조합 가능

```
max-width: calc(80% - 2rem);
height: calc(100vh - 60px);
```

→ 고정 헤더가 있는 레이아웃에서 `height: 100vh - header 높이` 처럼 사용 가능

---

## 4. 변수와 함께 사용

```
:root {
  --spacing: 20px;
}

.box {
  margin: calc(var(--spacing) * 2);
}
```

> 사용자 정의 속성(`--변수`)과 `calc()`를 결합하여 동적 간격 지정

---

## 5. 주의사항

| 항목 | 설명 |
| --- | --- |
| **공백 필수** | 연산자 앞뒤에 공백이 없으면 무시되거나 오류 발생 |
| **중첩 불가** | `calc()` 안에 또 다른 `calc()` 사용 불가 |
| **단위 누락 금지** | `px + %`처럼 **단위를 명확히 해야 함** |
| **브라우저 지원** | IE 9 이상 포함 거의 모든 브라우저에서 지원됨 |

---

## 6. 사용 시점 요약

| 상황 | 이유 |
| --- | --- |
| 고정 너비 요소와 유동형 레이아웃 조합 | `100% - 고정 px` |
| 반응형 폰트, 패딩 조정 | `vw`, `vh` + `rem`, `px` 혼합 |
| 가운데 정렬 및 여백 정밀 조정 | `calc()`를 통한 위치 조정 |
| CSS 변수와의 결합 | 테마 변경 시 유연하게 계산 가능 |

---

## 결론

> `calc()`는 고정값과 유동값을 조합하거나, 복잡한 레이아웃 계산을 손쉽게 구현하는 **CSS 내장 계산기**다.
> 특히 반응형 디자인, 중앙 정렬, 테마 시스템 등에서 **불가결한 도구**로 자주 사용된다.

# `clamp()` 함수

`clamp()`는 CSS에서 **값의 최소값과 최대값을 설정한 채, 동적으로 계산되는 값**을 지정하는 함수다.
브라우저가 **"중간 값"을 우선 사용하되**, 지정된 **최솟값 이하로 내려가지 않고, 최댓값 이상으로 올라가지 않도록** 제한해 준다.

---

## 1. 기본 문법

```
속성: clamp(최솟값, 중간값, 최댓값);
```

예:

```
font-size: clamp(14px, 2vw, 24px);
```

> 브라우저 너비가 작으면 `14px`,
> 커지면 `2vw`,
> 하지만 `24px`을 넘지는 않음.

---

## 2. 구성 요소 설명

| 요소 | 의미 |
| --- | --- |
| **최솟값** | 절대 최하 한도 – 더 작아질 수 없음 |
| **중간값** | 이상적인 계산값 – 보통 반응형 단위 사용 |
| **최댓값** | 절대 최고 한도 – 더 커질 수 없음 |

---

## 3. 실전 예제

### A. 반응형 폰트 크기

```
html {
  font-size: clamp(1rem, 2vw, 1.5rem);
}
```

- 뷰포트 너비가 작으면 `1rem`,
- 너비가 중간이면 `2vw`로 계산,
- 너비가 너무 커지면 `1.5rem`까지만 증가.

---

### B. 너비 제한

```
.container {
  width: clamp(320px, 80%, 1024px);
}
```

- 모바일에서는 최소 320px 확보
- 중간 해상도는 80%
- 데스크탑에선 최대 1024px로 제한

---

### C. 패딩, 마진 조절

```
.box {
  padding: clamp(8px, 2vw, 32px);
}
```

---

## 4. 기존 방식 vs `clamp()` 비교

```
/* 기존 방식 */
@media (min-width: 320px) {
  font-size: 14px;
}
@media (min-width: 768px) {
  font-size: 24px;
}

/* clamp 방식 */
font-size: clamp(14px, 2vw, 24px);
```

→ `clamp()`는 **미디어 쿼리 없이도 반응형 조건**을 처리할 수 있어 코드가 훨씬 간결함

---

## 5. 응용 – 동적 제목 크기

```
h1 {
  font-size: clamp(2rem, 5vw, 4rem);
}
```

- 작은 화면에서는 2rem
- 넓은 화면에서는 점점 커지다가
- 4rem 이상은 제한

---

## 6. 브라우저 지원

- 대부분의 **모던 브라우저(Chrome, Firefox, Edge, Safari)** 지원
- Internet Explorer ❌ 지원하지 않음

---

## 7. 실전 팁

- **폰트 크기, 너비, 패딩, 마진**에 자주 사용
- `vw`, `vh`, `%` 등을 중간값으로 활용
- 미디어 쿼리 최소화 가능
- 다른 함수(`calc()`, `min()`, `max()`)와 조합도 가능

---

## 결론

> `clamp()`는 **반응형 디자인을 위한 유연하고 안전한 도구**다.
> 한 줄로 `min`, `ideal`, `max` 값을 동시에 설정함으로써
> **디자인 파괴 없이 유동적인 사용자 경험**을 만들 수 있다.

# `aspect-ratio`

`aspect-ratio`는 CSS에서 **요소의 너비(width)와 높이(height) 간의 비율을 명시적으로 설정**할 수 있는 속성이다.
특정 비율을 유지하면서 요소의 크기를 유동적으로 지정해야 할 때 매우 유용하며, 반응형 UI 구성에 특히 효과적이다.

---

## ✅ 기본 문법

```
요소선택자 {
  aspect-ratio: 가로 / 세로;
}
```

예시:

```css
.box {
  aspect-ratio: 16 / 9;
}
```

- 위 코드는 `.box` 요소가 **16:9 비율을 유지하도록 설정**함
- `aspect-ratio: 1 / 1;` → 정사각형

---

## ✅ 동작 원리

- `aspect-ratio`는 `width` **또는** `height` **둘 중 하나만 설정해도** 나머지 하나를 자동 계산한다.
- 둘 다 설정되면 `aspect-ratio`는 무시된다.

```css
.rectangle {
  width: 300px;
  aspect-ratio: 4 / 3;
}
/* height는 300 * (3/4) = 225px로 자동 계산됨 */
```

---

## ✅ 예제: 반응형 비디오 플레이어

```css
.video-player {
  width: 100%;
  aspect-ratio: 16 / 9;
  background-color: black;
}
```

- 부모 요소의 너비가 변해도 **항상 16:9 비율을 유지**

---

## ✅ 예제: 정사각형 썸네일 카드

```css
.thumbnail {
  aspect-ratio: 1 / 1;
  width: 200px;
  background-color: #eee;
}
```

→ 정사각형 이미지 박스를 만들기 쉬움

---

## ✅ 예제: 이미지와 함께 사용

```
img.cover {
  width: 100%;
  aspect-ratio: 3 / 2;
  object-fit: cover;
}
```

- 이미지가 해당 비율로 잘림
- `object-fit`과 함께 쓰면 **비율 유지 + 채우기 전략** 구현 가능

---

## ✅ 기존 방법 vs `aspect-ratio`

기존에는 다음과 같은 해킹 방식으로 16:9 박스를 만들었음:

```
.ratio-box::before {
  content: "";
  display: block;
  padding-top: 56.25%; /* 9 / 16 * 100 */
}
```

→ `aspect-ratio`는 이를 **간결하고 명확하게 대체**

---

## ✅ 브라우저 지원

| 브라우저 | 지원 여부 |
| --- | --- |
| Chrome | ✅ (v88 이상) |
| Firefox | ✅ (v87 이상) |
| Safari | ✅ (v14.1 이상) |
| Edge | ✅ |
| IE | ❌ 지원 안 함 |

---

## ✅ 참고 사항

- 플렉스 박스(flex)나 그리드(grid)에서도 정상 동작
- `min-width`, `max-height` 등과 함께 비율을 제한하는 방식으로 쓰면 더욱 유연한 레이아웃 설계 가능

---

## ✅ 결론

> `aspect-ratio`는 **반응형 UI, 비디오, 이미지 썸네일, 카드 레이아웃 등**에서
> 고정된 비율을 유지하며 크기를 조절할 수 있는 매우 중요한 CSS 속성이다.
> 기존 해킹 기법 없이도 **간결하고 직관적인 방식**으로 비율 기반 레이아웃을 구현할 수 있다.

# `contain`, `isolation`

## `contain` – 레이아웃, 스타일, 페인트 계산 범위를 제한하는 최적화 속성

`contain` 속성은 브라우저에게 **이 요소는 외부와 독립적으로 처리될 수 있다**는 힌트를 주는 역할을 한다.
즉, 이 속성이 설정된 요소는 **레이아웃, 스타일 계산, 페인팅 등에서 부모나 형제 요소에 영향을 주지 않는다**.

---

### ✅ 기본 문법

```
.element {
  contain: layout style paint;
}
```

또는 한꺼번에 지정:

```
contain: content;  /* layout + style + paint */
```

---

### ✅ 값 목록

| 값 | 설명 |
| --- | --- |
| `none` | 기본값 (제한 없음) |
| `layout` | 요소의 레이아웃(위치, 크기)이 외부에 영향을 주지 않음 |
| `style` | 내부 스타일 계산이 외부와 독립 |
| `paint` | 시각적 효과(그림자 등)가 외부에 영향을 미치지 않음 |
| `size` | 콘텐츠 크기 계산이 내부에서만 결정됨 |
| `content` | `layout`, `style`, `paint` 모두 포함 (가장 일반적) |
| `strict` | `size` + `content` 포함 (모든 종류의 격리) |

---

### ✅ 실전 예시

```
.card {
  contain: content;
}
```

- `.card` 요소는 다른 요소와 **레이아웃 충돌 없이 독립적으로 처리됨**
- CSS 계산 최적화 및 **렌더링 성능 개선**

---

### ✅ 브라우저 최적화 관점

`contain`을 적절히 사용하면 다음과 같은 **성능 최적화**가 가능하다:

- **재계산 영역 축소 (Recalculation Scope 감소)**
- **리플로우 최소화**
- **GPU 페인트 영역 제한**

---

### ✅ 주의 사항

- `contain`을 사용하면 **자식 요소가 넘치더라도 외부에 영향을 못 미칠 수 있음**
- `size`를 사용할 경우, 콘텐츠 크기 자동 계산이 불가능할 수도 있음

---

---

## `isolation` - 요소의 스택 컨텍스트 분리

`isolation` 속성은 **요소가 새로운 스택 컨텍스트(Stacking Context)** 를 생성할지 여부를 결정한다.
주로 `z-index`, `mix-blend-mode`, `filter`, `transform` 등의 시각적 효과가 중첩될 때 예상치 못한 스타일 충돌을 방지하기 위해 사용된다.

---

### ✅ 기본 문법

```css
.element {
  isolation: isolate;
}
```

---

### ✅ 값 설명

| 값 | 설명 |
| --- | --- |
| `auto` | 기본값. 부모 컨텍스트를 상속 |
| `isolate` | 새로운 stacking context 생성 (자기만의 z-index 세계) |

---

### ✅ 실전 예시

```
.overlay {
  isolation: isolate;
  position: relative;
  z-index: 999;
}
```

- `.overlay`는 **부모의 z-index 영향을 받지 않음**
- 자신만의 z-index 공간 안에서 렌더링됨
- 주로 **레이어 충돌 방지**, **복잡한 UI**에서 사용

---

### ✅ stacking context란?

- 요소의 `z-index`, `opacity`, `transform`, `filter`, `will-change`, `mix-blend-mode` 등이 설정되면 자동 생성되는 **시각적 레이어 컨테이너**
- `isolation: isolate`는 이를 **강제로 생성**함

---

### ✅ 브라우저 호환성

| 속성 | 지원 여부 |
| --- | --- |
| `contain` | ✅ (Chrome, Firefox, Safari, Edge) – IE ❌ |
| `isolation` | ✅ (Chrome, Firefox, Safari, Edge) – IE ❌ |

---

## 정리 비교

| 속성 | 목적 | 효과 |
| --- | --- | --- |
| `contain` | 렌더링 성능 최적화 | 요소 내부를 독립 처리 |
| `isolation` | 시각적 겹침 충돌 방지 | stacking context 분리 |

---

## 결론

- `contain`은 **렌더링 성능 최적화**를 위한 핵심 속성으로, 복잡한 UI 구조에서 큰 이점을 가짐
- `isolation`은 **시각 효과가 충돌하지 않도록 레이어를 분리**하는 데 사용됨
- 둘 다 **레이아웃 안정성과 성능 개선을 위한 고급 도구**이며, 큰 프로젝트나 컴포넌트 설계 시 활용 가치가 높다

# `mix-blend-mode`, `filter`, `backdrop-filter`

다음은 시각적 연출 및 합성 효과에 사용되는 CSS 속성 3가지의 설명이다.
각 속성은 **레이어 합성**, **이미지 및 요소 보정**, **배경 흐림 등 시각 효과**를 위한 고급 도구다.

## 1. `mix-blend-mode` – 요소와 배경의 색상 합성 방식

`mix-blend-mode`는 **요소의 내용(텍스트, 이미지 등)이 배경과 어떻게 색상적으로 섞일지를 지정**하는 속성이다.
Photoshop의 레이어 블렌딩과 유사하다.

### ✅ 문법

```
.element {
  mix-blend-mode: multiply;
}
```

### ✅ 주요 값

| 값 | 설명 |
| --- | --- |
| `normal` | 기본값, 혼합 없음 |
| `multiply` | 곱셈 블렌드 – 어둡게 |
| `screen` | 스크린 블렌드 – 밝게 |
| `overlay` | 겹침 효과 – 대비 강조 |
| `darken` / `lighten` | 어두운/밝은 쪽 선택 |
| `color-dodge` / `color-burn` | 명도/채도 강조 |
| `difference`, `exclusion` | 차이 강조 |
| `hue`, `saturation`, `color`, `luminosity` | 색상 조절 기반 합성 |

### ✅ 예시

```
h1 {
  mix-blend-mode: overlay;
  color: white;
}
```

→ 흰색 글자가 배경 이미지와 **대비되며 겹쳐 보이는 효과**

---

## 2. `filter` – 요소 자체의 시각 효과 (흐림, 밝기, 대비 등)

`filter`는 요소에 **이미지 보정 필터**를 적용한다.
단일 요소에 대해 시각적 효과를 부여할 수 있으며, `img`, `div`, `svg` 등 모든 시각적 요소에 사용 가능하다.

### ✅ 문법

```
.element {
  filter: blur(4px) brightness(1.2) grayscale(60%);
}
```

### ✅ 주요 필터 함수

| 필터 함수 | 설명 |
| --- | --- |
| `blur(px)` | 흐림 효과 |
| `brightness(%)` | 밝기 조절 |
| `contrast(%)` | 대비 조절 |
| `grayscale(%)` | 흑백화 |
| `sepia(%)` | 세피아톤 |
| `invert(%)` | 색 반전 |
| `opacity(%)` | 투명도 조절 |
| `saturate(%)` | 채도 조절 |
| `hue-rotate(deg)` | 색상 회전 |

### ✅ 예시

```
img.thumbnail {
  filter: grayscale(100%) brightness(1.1);
}
```

→ 흑백 + 밝게 보정된 이미지

---

## 3. `backdrop-filter` – 배경에만 필터 효과를 적용

`backdrop-filter`는 요소 자체가 아니라 **요소의 뒷배경에만 시각적 필터를 적용**하는 속성이다.
반투명 유리 효과(glassmorphism) 구현 시 사용된다.

### ✅ 문법

```
.overlay {
  backdrop-filter: blur(10px);
  background-color: rgba(255, 255, 255, 0.2);
}
```

### ✅ 주요 사용 예

- `blur()` – 흐림
- `brightness()` – 밝기
- `contrast()`, `saturate()`, `grayscale()` 등 동일 필터 가능

### ✅ 반드시 `투명한 배경`과 함께 사용

- `background-color: rgba(..., alpha)` 같이 배경이 투명해야 배경 필터가 보임

---

### ✅ 예시: 글래스모피즘 카드

```
.card {
  background-color: rgba(255, 255, 255, 0.15);
  backdrop-filter: blur(8px);
  border-radius: 12px;
}
```

→ 배경이 흐려지고, 카드 내용은 선명하게 표시됨

---

## ✅ 브라우저 지원

| 속성 | 지원 여부 |
| --- | --- |
| `mix-blend-mode` | ✅ 거의 모든 최신 브라우저 지원 |
| `filter` | ✅ (IE 제외) |
| `backdrop-filter` | ✅ (Safari, Chrome, Edge, Firefox 103+) |
| `backdrop-filter` in Firefox | ✅ (하지만 일부는 `layout.css.backdrop-filter.enabled` 설정 필요) |
| IE | ❌ 모두 미지원 |

---

## 정리 비교

| 속성 | 대상 | 효과 |
| --- | --- | --- |
| `mix-blend-mode` | 요소 vs 배경 레이어 | 색상 합성 방식 결정 |
| `filter` | 요소 자체 | 색상, 흐림, 밝기 등 보정 |
| `backdrop-filter` | 요소 뒤의 배경 | 배경만 흐림/보정, 유리 느낌 |

# `scroll-behavior`, `scroll-snap-type`

이 두 속성은 웹 페이지에서 **스크롤의 부드러움(smoothness)** 또는 **스크롤 시 특정 위치에 자동 정렬(snap)** 기능을 제어하기 위해 사용된다.

---

## 1. `scroll-behavior` – 스크롤 애니메이션 동작 방식 제어

`scroll-behavior`는 **페이지 내 스크롤 이동이 즉시 진행될지, 부드럽게(smooth) 애니메이션처럼 이동할지를 지정**하는 속성이다.

### ✅ 문법

```
html {
  scroll-behavior: smooth;
}
```

### ✅ 주요 값

| 값 | 설명 |
| --- | --- |
| `auto` | 기본값. 즉시 이동 |
| `smooth` | 부드럽게 애니메이션 이동 |

### ✅ 예시

```
html {
  scroll-behavior: smooth;
}
```

```
<a href="#section3">섹션 3으로 이동</a>
...
<div id="section3">...</div>
```

→ 클릭 시 `#section3`으로 **부드럽게 스크롤 이동**

### ✅ 적용 대상

- 브라우저 기본 스크롤링
- `element.scrollIntoView({ behavior: 'smooth' })` 같은 JS 동작에도 연계됨

---

## 2. `scroll-snap-type` – 스크롤 정렬 기준 설정

`scroll-snap-type`은 **스크롤이 끝났을 때 특정 위치(스냅 포인트)에 자동 정렬되도록 설정**하는 속성이다.

## ✅ 문법

```
.scroll-container {
  scroll-snap-type: y mandatory;
}
```

## ✅ 구성 요소

- **축(Axis)**: `x`, `y`, `block`, `inline`, `both`
- **강제성**: `mandatory` (항상 스냅), `proximity` (근처일 때만 스냅)

```
.scroll-x {
  scroll-snap-type: x mandatory;
}
```

→ 수평 방향으로 항상 스냅 정렬

---

## ✅ `scroll-snap-align` – 각 아이템의 스냅 기준점 설정

```
.scroll-item {
  scroll-snap-align: start;
}
```

| 값 | 의미 |
| --- | --- |
| `start` | 시작 지점 맞춤 |
| `center` | 가운데 맞춤 |
| `end` | 끝 지점 맞춤 |

---

## ✅ 실전 예: 가로 스크롤 갤러리

```
<div class="scroll-container">
  <div class="item">1</div>
  <div class="item">2</div>
  <div class="item">3</div>
</div>
```

```css
.scroll-container {
  display: flex;
  overflow-x: auto;
  scroll-snap-type: x mandatory;
}

.item {
  flex: 0 0 100%;
  scroll-snap-align: start;
}
```

→ 아이템 하나씩 정확히 좌우로 "찰칵" 스크롤됨

---

### ✅ 브라우저 지원

| 속성 | 지원 여부 |
| --- | --- |
| `scroll-behavior` | ✅ 대부분 브라우저 |
| `scroll-snap-type` | ✅ (Chrome, Firefox, Safari, Edge) |
| IE | ❌ 미지원 |

---

## 요약 비교

| 속성 | 목적 | 동작 방식 |
| --- | --- | --- |
| `scroll-behavior` | 스크롤 이동 시 부드러운 애니메이션 제어 | `auto` or `smooth` |
| `scroll-snap-type` | 스크롤이 특정 위치에서 "멈추게" 조정 | `x/y mandatory/proximity` |
| `scroll-snap-align` | 각 항목의 정렬 기준점 설정 | `start/center/end` |

---

## 함께 자주 쓰이는 속성들

- `overflow: auto / scroll`
- `scroll-padding`, `scroll-margin`
- `scroll-snap-stop`
- `scroll-snap-type: both proximity`
- JavaScript: `scrollIntoView({ behavior: 'smooth' })`

# `writing-mode`, `direction`

다음은 **글쓰기 방향**과 **문자 배치 흐름**을 제어하는 CSS 속성 `writing-mode`와 `direction`에 대한 설명이다. 이들은 주로 다국어 웹페이지, 특히 아시아권(중국어, 일본어, 한국어) 및 아랍권 사이트에서 매우 중요하게 사용된다.

---

# 1. `writing-mode` – 텍스트의 흐름 방향 설정

`writing-mode`는 요소 내부의 **글줄(텍스트 라인) 배치 방향**을 설정하는 속성이다.
이 속성은 **블록 흐름의 방향**을 제어하며, 텍스트 줄이 **가로로 흐를지, 세로로 흐를지** 결정한다.

---

## ✅ 문법

```
.element {
  writing-mode: vertical-rl;
}
```

---

## ✅ 주요 값

| 값 | 설명 |
| --- | --- |
| `horizontal-tb` | 수평 → 위에서 아래로 (기본값) |
| `vertical-rl` | 세로 → 오른쪽에서 왼쪽으로 (일본어 세로쓰기 등) |
| `vertical-lr` | 세로 → 왼쪽에서 오른쪽으로 |
| `sideways-rl` | 텍스트 회전 포함, 세로 줄이 오른쪽에서 왼쪽 |
| `sideways-lr` | 텍스트 회전 포함, 세로 줄이 왼쪽에서 오른쪽 |

---

## ✅ 예시

```
.vertical {
  writing-mode: vertical-rl;
}
```

```
<div class="vertical">縦書きテキスト</div>
```

→ 텍스트가 위에서 아래로, 줄은 오른쪽에서 왼쪽으로 흐름

---

## ✅ 참고 사항

- `writing-mode`는 **줄 방향 (block-flow)** 을 바꿈
- 글자 자체의 **방향성은** `direction` 으로 제어함 (예: 오른쪽→왼쪽 문자 흐름)

---

## 2. `direction` – 텍스트 및 콘텐츠의 기본 흐름 방향 설정

`direction`은 **문자의 정렬 방향 (ltr vs rtl)** 을 지정하는 속성이다.
이는 **인라인 텍스트의 진행 방향, 텍스트 정렬 기본값, 리스트 기호 방향 등**에 영향을 미친다.

---

### ✅ 문법

```
.element {
  direction: rtl;
}
```

---

### ✅ 주요 값

| 값 | 설명 |
| --- | --- |
| `ltr` | 왼쪽 → 오른쪽 (기본, 대부분 언어) |
| `rtl` | 오른쪽 → 왼쪽 (아랍어, 히브리어 등) |
| `inherit` | 부모로부터 상속 |

---

### ✅ 예시

```
<div style="direction: rtl;">
  هذا نص باللغة العربية.
</div>
```

→ 오른쪽에서 왼쪽으로 텍스트 흐름

---

### ✅ `direction`이 영향을 주는 요소

- 텍스트의 정렬 방향 (`text-align` 기본값)
- 텍스트 커서 이동 방향
- 리스트 마커 위치 (`<ul>`, `<ol>`)
- 테이블 셀의 기본 정렬
- 폼 요소의 텍스트 입력 방향

---

### ✅ `unicode-bidi`와 함께 사용

```
.rtl-embed {
  direction: rtl;
  unicode-bidi: embed;
}
```

- `unicode-bidi`는 텍스트 흐름 제어를 **강제로 적용하거나 무시하는 방식**
- 복잡한 다국어 콘텐츠에서 `direction`과 함께 쓰임

---

## 정리 비교

| 속성 | 대상 | 설명 |
| --- | --- | --- |
| `writing-mode` | 블록 흐름 | 줄 방향: 가로 vs 세로 |
| `direction` | 인라인 흐름 | 문자 진행 방향: 좌→우 or 우→좌 |

---

## 예제 비교

```
/* 일본어 세로쓰기 */
.jp-text {
  writing-mode: vertical-rl;
  direction: ltr;
}

/* 아랍어 */
.ar-text {
  direction: rtl;
}
```

---

## 브라우저 지원

| 속성 | 지원 여부 |
| --- | --- |
| `writing-mode` | ✅ Chrome, Firefox, Safari, Edge |
| `direction` | ✅ 모든 브라우저 |
| IE | ✅ 지원 (일부 `writing-mode` 제한 있음) |

# 다크 모드 대응: `prefers-color-scheme`

`prefers-color-scheme`은 사용자의 시스템 설정에 따라 웹사이트의 **밝은 테마(라이트 모드)** 또는 **어두운 테마(다크 모드)**를 자동으로 적용할 수 있도록 도와주는 CSS **미디어 쿼리(Media Query)** 이다.

이는 사용자 경험을 향상시키기 위한 **접근성** 및 **개인화 기능**으로, 현대 웹에서 거의 필수적인 요소가 되었다.

---

## 1. 기본 문법

```
@media (prefers-color-scheme: dark) {
  /* 다크 모드 스타일 */
}

@media (prefers-color-scheme: light) {
  /* 라이트 모드 스타일 */
}
```

---

## 2. 값

| 값 | 의미 |
| --- | --- |
| `light` | 사용자 시스템이 라이트 모드를 선호 |
| `dark` | 사용자 시스템이 다크 모드를 선호 |
| `no-preference` | (명시 X) 기본 모드 사용, 대부분 `light` 처리됨 |

---

## 3. 예시: 기본 테마 + 다크 모드 적용

```
/* 기본 테마 (라이트 모드) */
body {
  background-color: #ffffff;
  color: #000000;
}

/* 다크 모드 사용자용 */
@media (prefers-color-scheme: dark) {
  body {
    background-color: #121212;
    color: #e0e0e0;
  }

  a {
    color: #90caf9;
  }
}
```

→ 사용자의 운영체제(Windows, macOS, Android, iOS 등)에서 다크 모드를 설정하면 자동으로 이 스타일이 적용됨.

---

## 4. 실전 적용 전략

- 디자인 시스템에서 변수 기반으로 색상 관리 권장:

```
:root {
  --bg-color: #ffffff;
  --text-color: #000000;
}

@media (prefers-color-scheme: dark) {
  :root {
    --bg-color: #121212;
    --text-color: #e0e0e0;
  }
}

body {
  background-color: var(--bg-color);
  color: var(--text-color);
}
```

- 유지보수성 증가
- JavaScript와도 연동 가능

---

## 5. JavaScript에서 감지

```
if (window.matchMedia && window.matchMedia('(prefers-color-scheme: dark)').matches) {
  console.log('다크 모드 사용 중');
}
```

→ 사용자의 설정에 따라 동적으로 테마 조정 가능

---

## 6. 브라우저 지원

| 브라우저 | 지원 여부 |
| --- | --- |
| Chrome | ✅ |
| Firefox | ✅ |
| Safari | ✅ |
| Edge | ✅ |
| IE | ❌ 미지원 |

---

## 7. 사용 예시 화면

- macOS: 시스템 환경설정 > 일반 > [다크 모드 / 라이트 모드] 설정
- Windows 10/11: 설정 > 개인 설정 > 색 > [기본 앱 모드]
- Android, iOS: 시스템 디스플레이 설정

---

## 결론

- `prefers-color-scheme`은 **시스템 설정을 자동 감지**하여 테마를 적용할 수 있는 **표준적인 접근 방식**
- JavaScript와 결합하면 **수동 토글 + 자동 감지** 기능 모두 구현 가능
- 변수 기반 설계와 함께 사용하면 **대규모 웹사이트에도 유연하게 확장 가능**

# `@supports` 조건문

`@supports`는 특정 **CSS 속성이나 값이 브라우저에서 지원되는지 여부에 따라**
조건부로 스타일을 적용할 수 있게 해주는 **CSS 조건문**이다.

이는 JavaScript의 `if` 문처럼 **브라우저 기능을 감지**하여
"지원하는 경우만 해당 스타일을 적용"할 수 있도록 해준다.

---

## 1. 기본 문법

```
@supports (속성: 값) {
  /* 지원하는 경우에만 적용할 스타일 */
}
```

예:

```
@supports (display: grid) {
  .container {
    display: grid;
  }
}
```

→ `display: grid`를 **지원하는 브라우저에서만** 해당 스타일을 적용

---

## 2. 부정 조건

```
@supports not (display: grid) {
  .container {
    display: flex;
  }
}
```

→ `display: grid`를 **지원하지 않는 브라우저에서만** Flexbox로 대체

## 3. 복합 조건

AND, OR, NOT 같은 **논리 연산자**도 지원한다:

| 연산자 | 의미 |
| --- | --- |
| `and` | 모든 조건이 참이어야 함 |
| `or` | 조건 중 하나만 참이면 됨 |
| `not` | 조건이 거짓일 때 적용 |

```
@supports (display: grid) and (gap: 1rem) {
  .layout {
    display: grid;
    gap: 1rem;
  }
}
```

## 4. 중첩 사용 가능

```
@supports (display: grid) {
  @media (min-width: 600px) {
    .grid-layout {
      display: grid;
    }
  }
}
```

→ 기능 지원 + 미디어 조건 모두 충족할 때만 적용됨

## 5. 실전 예시

```
/* 기본 폴백 (지원 안 되는 경우) */
.button {
  background-color: #ccc;
}

/* 브라우저가 backdrop-filter를 지원할 경우 */
@supports (backdrop-filter: blur(10px)) {
  .button {
    backdrop-filter: blur(10px);
    background-color: rgba(255, 255, 255, 0.3);
  }
}
```

## 6. 지원 속성 확인 팁

`@supports`는 **속성과 값의 조합**이 **정확히 유효해야 true**로 평가됨:

```
@supports (display: unknown-value) {
  /* 이 블록은 절대 실행되지 않음 */
}
```

---

## 7. 브라우저 지원 현황

| 브라우저 | `@supports` 지원 여부 |
| --- | --- |
| Chrome | ✅ |
| Firefox | ✅ |
| Safari | ✅ |
| Edge | ✅ |
| IE 11 이하 | ❌ (지원하지 않음) |

---

## 8. JavaScript 대안

JavaScript에서도 비슷한 기능을 제공:

```
if (CSS.supports('display', 'grid')) {
  // display: grid 지원
}
```

---

## 결론

- `@supports`는 **점진적 향상(Progressive Enhancement)** 전략의 핵심 도구
- 최신 브라우저 기능을 사용하면서도, 하위 호환성을 유지할 수 있게 함
- `feature detection`을 CSS 차원에서 수행할 수 있음