13. 애니메이션과 트랜지션

`transition`

CSS `transition`은 **특정 CSS 속성이 변화할 때 부드러운 애니메이션 효과**를 줄 수 있게 해주는 기능이다.
기존에는 자바스크립트를 사용해 구현하던 인터랙션을 CSS만으로도 쉽게 표현할 수 있게 해준다.

---

1. 기본 개념

```
transition: [속성] [지속 시간] [타이밍 함수] [지연 시간];
```

> 모든 값은 선택적으로 생략 가능하나, **최소한 속성과 지속 시간은 지정**해야 한다.

---

2. 예시

```
.box {
  background-color: blue;
  transition: background-color 0.5s ease;
}

.box:hover {
  background-color: red;
}
```

> 마우스를 올리면 `background-color`가 0.5초에 걸쳐 **부드럽게** 파란색에서 빨간색으로 변함

---

3. 속성 구성

속성	설명	예시
`transition-property`	전환 대상 속성	`background-color`, `width`, `all`
`transition-duration`	지속 시간 (필수)	`0.3s`, `500ms`
`transition-timing-function`	가속도 곡선	`ease`, `linear`, `ease-in`, `ease-out`, `ease-in-out`, `cubic-bezier()`
`transition-delay`	지연 시간	`0s`, `0.2s` 등

---

4. 단축 속성 사용

```
transition: all 0.3s ease-in-out 0.1s;
```

- `all`: 모든 속성에 적용
- `0.3s`: 지속 시간
- `ease-in-out`: 가속도 곡선
- `0.1s`: 지연 시간

---

5. 여러 속성 동시에 전환

```
.box {
  transition: width 0.4s ease, height 0.6s linear;
}
```

- `width`는 0.4초 동안 `ease`로,
- `height`는 0.6초 동안 `linear`로

---

6. 전환 가능한 속성 예시

속성 유형	예시
색상 관련	`color`, `background-color`, `border-color`
박스 크기	`width`, `height`, `margin`, `padding`, `max-width` 등
위치 이동	`top`, `left`, `right`, `bottom`
투명도	`opacity`
회전, 확대 등	`transform`
그림자	`box-shadow`, `text-shadow`

> 단, 일부 속성은 애니메이션이 적용되지 않거나 자연스럽지 않을 수 있음

---

7. `transition`과 `hover`, `focus`, `active` 조합 예

```
button {
  background-color: #3498db;
  color: white;
  padding: 10px 20px;
  transition: background-color 0.3s ease;
}

button:hover {
  background-color: #2980b9;
}
```

> 버튼의 배경색이 부드럽게 변경되어 **자연스러운 사용자 피드백** 제공

---

8. `transform`과의 조합

```
.card {
  transform: scale(1);
  transition: transform 0.2s ease-out;
}

.card:hover {
  transform: scale(1.05);
}
```

> 마우스를 올리면 카드를 확대하는 **효과적인 UI 피드백** 구현

---

9. `JavaScript`와의 연동

자바스크립트로 `class`를 추가/제거하면서도 `transition`이 자연스럽게 동작함:

```
<div id="box" class="box"></div>

<style>
.box {
  width: 100px;
  height: 100px;
  background: red;
  transition: width 0.5s;
}

.box.expand {
  width: 300px;
}
</style>

<script>
```

```
  document.getElementById('box').addEventListener('click', function () {
    this.classList.toggle('expand');
  });
</script>
```

---

10. 주의사항

- `display` 속성은 `transition`이 적용되지 않음 (`block → none`은 갑작스러운 변화)
- 변화 전후 값이 명확하게 정의되어야 함
- `transition`은 **시작과 끝 상태만 다룬다** (프레임 기반 애니메이션은 `@keyframes` / `animation` 사용)

결론

> `transition`은 **간단한 인터랙션 효과**를 줄 때 가장 유용한 도구로,
> CSS만으로도 직관적이고 효율적인 UI 효과를 만들 수 있게 한다.
> `transform`, `opacity`, `background-color` 등과 함께 사용하면 **가볍고 빠른 반응형 디자인** 구현에 적합하다.

`transform`

`transform` 속성은 요소에 **회전, 크기 조절, 기울이기, 이동** 등의 **2D 및 3D 변형 효과**를 적용하는 데 사용된다.
자바스크립트 없이도 인터랙티브하고 생동감 있는 UI를 만드는 핵심 도구다.

---

1. 기본 문법

```
transform: 변형함수1 값1 [변형함수2 값2 ...];
```

하나 이상의 변형 함수를 **공백으로 구분하여 나열**할 수 있다.
→ 복합 변형도 가능함

---

2. 주요 2D 변형 함수

함수	설명	예시
`translate(x, y)`	위치 이동	`translate(50px, 100px)`
`scale(x, y)`	크기 확대/축소	`scale(2, 0.5)`
`rotate(angle)`	회전	`rotate(45deg)`
`skew(x-angle, y-angle)`	기울이기	`skew(30deg, 0deg)`
`matrix(a, b, c, d, e, f)`	위의 2D 효과를 하나로 합친 행렬 표현	고급 사용 시

✅ `translate()`

```
transform: translate(50px, 0);
```

> 요소를 오른쪽으로 50px 이동

- `translateX()`, `translateY()`도 있음

✅ `scale()`

```
transform: scale(1.5);
```

> 1.5배 확대

- `scaleX(2)`, `scaleY(0.5)`처럼 방향별 조정도 가능

✅ `rotate()`

```
transform: rotate(45deg);
```

> 요소를 45도 시계 방향 회전

✅ `skew()`

```
transform: skew(20deg, 0);
```

> X축 기준으로 기울이기

---

3. 복합 변형 예시

```
transform: translate(50px, 0) scale(1.2) rotate(10deg);
```

> 이동, 확대, 회전을 **동시에** 적용

---

4. `transform-origin`: 기준점 설정

- 변형이 어디를 기준으로 적용될지 설정

```
transform-origin: center center;    /* 기본값 */
transform-origin: top left;
transform-origin: 100% 50%;
```

> 예: 회전할 때 **왼쪽 상단**을 중심으로 회전하고 싶을 경우

---

5. 3D 변형 함수 (선택적 고급)

함수	설명
`rotateX(deg)`	X축 기준 회전
`rotateY(deg)`	Y축 기준 회전
`translateZ(z)`	Z축 방향 이동
`scaleZ(n)`	Z축 크기 조절
`perspective(n)`	원근감 부여

```
transform: perspective(500px) rotateY(45deg);
```

---

6. 실전 예: 카드 확대 효과

```
.card {
  transition: transform 0.3s ease;
}

.card:hover {
  transform: scale(1.05);
}
```

> 마우스를 올리면 부드럽게 커짐

---

7. 실전 예: 회전 버튼

```
button {
  transition: transform 0.2s;
}

button:hover {
  transform: rotate(360deg);
}
```

> **회전하는 버튼**처럼 시각적 인터랙션을 간편하게 구현

---

8. `transform` vs `position`/`margin`/`top`

항목	`transform`	`position`, `margin`, etc.
애니메이션 성능	✅ GPU 가속 (더 부드러움)	❌ CPU 중심
문서 흐름 영향	❌ 흐름에 영향 없음	✅ 레이아웃 변경 발생

항목	`transform`	`position`, `margin`, **etc.**
연산 단순성	✅ 다양한 효과 결합	제한적 위치 조절만 가능

결론

> `transform`은 **레이아웃을 깨뜨리지 않고 시각적 변형만 적용**할 수 있는 강력한 도구다.
> 특히 `transition`, `hover`, `animation` 등과 함께 사용하면 **인터랙티브하고 부드러운 사용자 경험**을 손쉽게 구현할 수 있다.

`animation`

CSS `animation` 속성은 정적인 스타일 변화에 그치지 않고, **시간에 따른 복잡한 애니메이션 시퀀스를 설정**할 수 있게 해준다.
이는 `@keyframes` 규칙과 함께 사용되며, JavaScript 없이도 강력한 동적 UI 효과를 구현할 수 있다.

---

1. 기본 개념

`transition`이 **상태 변화**에 반응하는 애니메이션이라면,
`animation`은 **자체적으로 반복되거나 시간에 따라 진행되는 동작**을 정의한다.

---

2. 핵심 구성 요소

```
selector {
  animation-name: slide;
  animation-duration: 1s;
  animation-timing-function: ease;
  animation-delay: 0s;
  animation-iteration-count: infinite;
  animation-direction: alternate;
  animation-fill-mode: forwards;
  animation-play-state: running;
}
```

- `animation-name`: 적용할 키프레임 이름
- `animation-duration`: 한 번 재생되는 데 걸리는 시간
- `animation-timing-function`: 가속도 곡선 (ease, linear 등)
- `animation-delay`: 시작 전 대기 시간
- `animation-iteration-count`: 반복 횟수 (`1`, `infinite`)
- `animation-direction`: 방향 (`normal`, `reverse`, `alternate`)
- `animation-fill-mode`: 시작 전/끝난 후 상태 (`none`, `forwards`, `backwards`, `both`)
- `animation-play-state`: 실행 상태 (`running`, `paused`)

---

3. 단축 속성

```
animation: slide 1s ease-in-out 0.2s infinite alternate;
```

순서:
`[name] [duration] [timing-function] [delay] [iteration-count] [direction]`

---

4. `@keyframes` 정의

```
@keyframes slide {
  from {
    transform: translateX(0);
  }
  to {
    transform: translateX(200px);
  }
}
```

또는 중간 단계 포함:

```
@keyframes fadeMove {
  0% {
    opacity: 0;
    transform: translateY(50px);
  }
  100% {
    opacity: 1;
    transform: translateY(0);
  }
}
```

---

5. 예제: 자동 이동하는 상자

```
<div class="box"></div>

<style>
.box {
  width: 100px;
  height: 100px;
  background: red;
  animation: moveRight 2s linear infinite;
}

@keyframes moveRight {
  0% {
    transform: translateX(0);
  }
  100% {
```

```
    transform: translateX(300px);
  }
}
</style>
```

---

6. 반복, 방향, 채움 속성

속성	예시	설명
`animation-iteration-count`	`infinite`, `3`	반복 횟수
`animation-direction`	`normal`, `reverse`, `alternate`	재생 방향
`animation-fill-mode`	`forwards`	종료 상태 유지
`animation-delay`	`1s`	시작 전 대기 시간
`animation-play-state`	`paused`, `running`	재생 상태 제어

---

7. 다중 애니메이션 적용

```
.box {
  animation: fadeIn 1s ease, moveUp 2s ease-in-out;
}
```

> 여러 애니메이션을 동시에 적용할 수 있으며, 쉼표로 구분함

---

8. `hover`와 애니메이션 차이

- `transition`: 상태 변화 시 부드러운 변화 (마우스 오버, 포커스 등)
- `animation`: **자동으로 실행**, 반복 가능, 더 복잡한 타임라인 가능

---

9. 자바스크립트와 함께 사용

```
element.addEventListener('animationend', function () {
  console.log('애니메이션 완료');
});
```

---

10. 실전 활용 예

A. 로딩 스피너

```css
.loader {
  border: 4px solid #ccc;
  border-top: 4px solid blue;
  border-radius: 50%;
  width: 40px;
  height: 40px;
  animation: spin 1s linear infinite;
}

@keyframes spin {
  100% {
    transform: rotate(360deg);
  }
}
```

---

11. `animation` vs `transition` 비교

항목	`transition`	`animation`
트리거	상태 변화 (ex: `hover`)	자동 또는 JS로 실행
키프레임	시작-끝만 존재	다단계 타임라인 구성 가능
반복	❌	✅ 가능 (`infinite`)
정교함	단순	복잡한 시퀀스 가능

결론

> CSS `animation`은 JavaScript 없이도 강력한 **시퀀스 기반 인터랙션**을 만들 수 있게 해준다.
> `@keyframes`로 타임라인을 설계하고, 다양한 속성으로 제어하면 UI의 **생동감, 피드백, 주목성**을 크게 향상시킬 수 있다.

`will-change` 최적화

`will-change` 속성은 **브라우저에게 특정 속성이 앞으로 변경될 것임을 미리 알려주는 힌트**를 제공함으로써, **렌더링 성능 최적화**를 유도할 수 있다.
→ 특히 **애니메이션**, **스크롤 반응 요소**, **transform**, **opacity** 변화 등에 유용하다.

---

1. 기본 문법

```
.selector {
  will-change: 속성명;
}
```

예:

```
.card {
  will-change: transform, opacity;
}
```

> 브라우저는 이 요소가 곧 `transform`이나 `opacity` 속성의 변화를 가질 것이라 예상하여
> GPU 레이어를 미리 생성하는 등 **선제적 최적화**를 수행할 수 있음

---

2. 사용 목적

- 렌더링을 위한 브라우저의 **컴포지팅 단계**를 미리 준비
- 변화할 속성을 지정함으로써 **레이어 승격**을 유도
- 애니메이션이나 전환 시 **프레임 드랍 감소**, **부드러운 UI 제공**

---

3. 대표 사용 예시

A. 요소가 자주 이동하거나 변형될 경우

```
.animated-box {
  will-change: transform;
}
```

- `hover`, `focus`, `animation`, `scroll` 등에서 이동할 경우
  미리 `transform`을 브라우저가 준비하면 **GPU 가속**을 일으켜 렌더링 병목을 줄인다.

B. 점점 사라지거나 나타날 요소

```
.fade-in {
  will-change: opacity;
}
```

---

4. 주의사항

주의 항목	설명
과도한 사용 금지	브라우저는 `will-change` 요소에 **GPU 레이어를 미리 할당**하므로 **메모리 소비 증가**
변화 직전에만 적용	해당 속성이 실제로 **변하지 않을 경우 오히려 성능 저하**

주의 항목	설명
동적 제어 권장	JavaScript로 **변화 직전**에 `will-change` 추가, 이후 제거하는 방식이 가장 효율적

---

5. 자바스크립트 예시

```
const box = document.querySelector('.box');

box.addEventListener('mouseenter', () => {
  box.style.willChange = 'transform';
});

box.addEventListener('mouseleave', () => {
  box.style.willChange = 'auto';
});
```

> 변화가 **임박했을 때만** `will-change`를 적용하고 끝나면 제거 → **정확한 타이밍 최적화**

---

6. 추천 대상 속성

속성	설명
`transform`	이동, 확대/축소, 회전 등
`opacity`	투명도 변화
`top`, `left`, `right`, `bottom` (때때로)	`position: absolute/fixed`인 경우
`scroll-position`, `contents`	레이아웃 변화 예측 시 사용 (드물게 활용)

---

7. 잘못된 사용 예

```
/* BAD: 의미 없는 레이아웃 전체 최적화 */
body {
  will-change: all;
}
```

- ⚠️ 불필요한 GPU 리소스 사용으로 오히려 성능 저하 초래

---

8. 대체 전략

> 대부분의 상황에서 `will-change`보다 `transform`, `opacity`**만을 활용한 CSS 애니메이션**이 더 최적화되어 있음
> → 가능한 한 `top`, `left`, `width` 대신 `transform`을 사용하자

결론

> `will-change`는 브라우저에게 "이 속성이 곧 바뀔 거야!"라고 **힌트를 주는 성능 향상 도구**다.
> 그러나 잘못 사용하면 **GPU 리소스를 낭비하고, 성능을 오히려 떨어뜨릴 수 있으므로**
> 정확히 필요한 요소에, 필요한 타이밍에만 적용해야 한다.