9. 레이아웃 기초

블록 요소 vs 인라인 요소

HTML과 CSS에서 요소는 **기본적으로 두 가지 형태 중 하나**로 분류된다:
블록(Block) 요소와 **인라인(Inline) 요소**.
이는 요소가 **페이지에서 차지하는 공간**과 **다른 요소와의 배치 방식**에 큰 영향을 준다.

---

1. 블록 요소 (Block-level Elements)

개념

- 항상 **새 줄에서 시작**됨
- **가로 전체 너비**를 차지함 (`width: 100%` 기본)
- `width`, `height`, `margin`, `padding` 등의 속성이 **모두 적용 가능**

대표 태그

```
<div>, <p>, <h1> ~ <h6>, <ul>, <ol>, <li>, <section>, <article>, <nav>, <header>,
<footer>, <form>, <table> 등
```

예시

```
<div style="background: lightgray;">Block Element</div>
<div style="background: lightblue;">Next Block Element</div>
```

→ 서로 다른 줄에 위치하며, 기본적으로 줄바꿈 발생

---

2. 인라인 요소 (Inline Elements)

개념

- **새 줄에서 시작되지 않음** (한 줄 안에 배치됨)
- **콘텐츠 크기만큼**만 너비를 차지
- `width`, `height`는 무시되며 **적용되지 않음**
- `padding`, `margin`은 **좌우만 적용**, 위아래는 무시되거나 제한적

대표 태그

```
<span>, <a>, <strong>, <em>, <b>, <i>, <img>, <label>, <abbr>, <code> 등
```

예시

```
<p>이 문장에는 <span style="color: red;">인라인 요소</span>가 포함되어 있습니다.</p>
```

→ 줄 바꿈 없이 한 문장 안에 삽입

---

3. 비교 요약

항목	블록 요소	인라인 요소
줄 바꿈 여부	항상 줄 바꿈 발생	같은 줄에 배치
너비 기본값	부모의 100%	콘텐츠 크기만큼
`width` / `height` 설정	가능	대부분 무시됨
`margin`, `padding`	전 방향 가능	수직 방향은 제한적
레이아웃 용도	구조적 레이아웃 설계	텍스트 일부 스타일링

---

4. 대표 사례 비교

```
<!-- 블록 요소 -->
<div style="border: 1px solid black;">DIV</div>

<!-- 인라인 요소 -->
<span style="border: 1px solid red;">SPAN</span>
```

결과:

- `div`: 줄 바꿈 발생, 넓은 영역
- `span`: 줄 바꿈 없음, 텍스트 중간 삽입 가능

---

5. 혼합/변환

- CSS의 `display` 속성으로 **요소의 디스플레이 유형 변경** 가능

```
/* 인라인 요소를 블록처럼 */
span {
  display: block;
}

/* 블록 요소를 인라인처럼 */
div {
  display: inline;
}
```

- `inline-block`: 인라인처럼 흐르되, `width`, `height` 적용 가능

```
button {
  display: inline-block;
}
```

결론

> **블록 요소**는 페이지의 **전체 구조나 섹션**을 잡는 데 사용되고,
> **인라인 요소**는 **텍스트 일부나 작은 범위**의 스타일링에 주로 사용된다.
> `display` 속성으로 유연하게 형태를 변환하여 다양한 레이아웃을 구성할 수 있다.

`display` 속성

`display` 속성은 HTML 요소가 **페이지에 어떻게 배치될지를 정의**하는 가장 기본이자 핵심적인 CSS 속성이다.
모든 레이아웃 설계의 출발점이며, **요소의 형식(블록, 인라인, 그리드, 플렉스 등)**을 지정한다.

---

1. 기본 문법

```
selector {
  display: <값>;
}
```

---

2. 주요 `display` 값 요약

값	설명
`block`	블록 요소로 표시 (새 줄에서 시작, 전체 너비 차지)
`inline`	인라인 요소로 표시 (콘텐츠만큼 너비 차지, 줄바꿈 없음)
`inline-block`	인라인처럼 흐르되, `width` / `height` 설정 가능
`none`	요소를 화면에서 **완전히 제거** (렌더링 X, 공간 차지 X)
`flex`	**플렉스 컨테이너**로 설정 (1차원 정렬에 강력)
`inline-flex`	`flex`와 같지만 인라인 컨텍스트
`grid`	**그리드 컨테이너**로 설정 (2차원 레이아웃 구성)
`inline-grid`	`grid`와 같지만 인라인 컨텍스트
`table`, `inline-table`	HTML `<table>`과 유사한 테이블 형식
`list-item`	목록 항목처럼 표시 (`<li>`와 같은 스타일)

---

3. 대표적인 사용 예시

▸ `block`

```
div {
  display: block;
}
```

→ 전체 너비 차지, 줄 바꿈 발생

▸ `inline`

```
span {
  display: inline;
}
```

→ 콘텐츠 크기만큼만 공간 차지, 줄 바꿈 없음

▸ `inline-block`

```
.button {
  display: inline-block;
  width: 100px;
  height: 40px;
}
```

→ `inline`처럼 배치되지만, 크기 지정 가능

▸ `none`

```
.modal {
  display: none;
}
```

→ 요소 숨김 (DOM에는 존재하지만 시각적으로 제거됨)

---

4. 레이아웃 컨테이너용

▸ `flex`

```
.container {
  display: flex;
  justify-content: space-between;
  align-items: center;
}
```

→ 자식 요소를 가로 또는 세로 방향으로 정렬하는 유연한 1차원 레이아웃

▸ `grid`

```
.container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
}
```

→ 행과 열을 활용한 2차원 레이아웃 구성 가능

---

5. `inline` vs `inline-block` vs `block`

속성	너비 지정 가능	줄 바꿈 발생	수직 여백 적용
`inline`	❌	❌	거의 불가능
`inline-block`	✅	❌	✅
`block`	✅	✅	✅

---

6. 숨김용 vs 제거용

값	설명
`display: none`	완전히 제거 (레이아웃 공간 X)
`visibility: hidden`	요소는 보이지 않지만 공간은 유지됨

---

7. 브라우저 기본값 참고

HTML 태그는 기본적으로 `display` 속성이 설정되어 있음:

태그	기본 `display`
`<div>`, `<p>`, `<section>`	`block`
`<span>`, `<a>`, `<strong>`	`inline`
`<li>`	`list-item`
`<table>`	`table`

결론

> `display`는 CSS에서 **가장 중요하고 기본적인 레이아웃 속성**으로,
> 요소를 어떻게 배치할지 결정하며, **flex/grid와 같은 고급 레이아웃의 시작점**이 된다.
> 요소의 특성을 바꾸거나, 숨기거나, 정렬 구조를 만들기 위해 반드시 숙지해야 할 속성이다.

`position` 속성

`position` 속성은 HTML 요소를 **문서 내에서 어떻게 배치할지를 결정**하는 핵심 속성이다.
기본적인 흐름(문서의 자연스러운 배치)을 따르지 않고, **절대 위치 또는 상대 위치 지정** 등을 가능하게 한다.

---

1. 주요 값 요약

값	설명
`static`	기본 값. 문서 흐름(기본 배치)에 따라 배치
`relative`	**자기 자신을 기준**으로 위치 이동. 공간은 그대로 유지
`absolute`	**가장 가까운 위치 지정 조상(ancestor)**을 기준으로 절대 위치 배치
`fixed`	**브라우저 뷰포트**를 기준으로 고정 배치. 스크롤해도 위치 변하지 않음
`sticky`	스크롤 위치에 따라 `static`과 `fixed` 사이를 전환하는 하이브리드 방식

---

2. 사용 가능한 위치 지정 속성

`position`이 `static`이 아닌 경우 다음 속성들을 사용해 위치를 조정할 수 있다:

- `top`
- `right`
- `bottom`
- `left`
- `z-index`

```
.box {
  position: relative;
  top: 20px;
  left: 10px;
}
```

---

3. 각 값의 동작 방식

▸ `static` (기본값)

```
div {
  position: static;
}
```

- 기본 문서 흐름에 따라 배치됨
- `top`, `left` 등의 위치 속성 **무시됨**

▸ `relative`

```
.box {
  position: relative;
  top: 10px;
  left: 20px;
}
```

- **자기 자신의 원래 위치 기준으로 이동**
- 원래 공간은 **그대로 유지됨**

▸ `absolute`

```
.outer {
  position: relative;
}
.inner {
  position: absolute;
  top: 0;
  left: 0;
}
```

- **가장 가까운** `position: relative` **or** `absolute` **or** `fixed` **부모 요소**를 기준으로 위치함
- 문서 흐름에서 **제외**됨 (공간 차지하지 않음)
- `z-index`로 겹침 순서 조정 가능

▸ `fixed`

```
.floating-button {
  position: fixed;
  bottom: 10px;
  right: 10px;
}
```

- **브라우저 뷰포트** 기준
- 스크롤해도 **항상 같은 위치**에 머무름 (고정 버튼 등 UI에 사용)
- 문서 흐름에서 제외됨

▸ `sticky`

```
.header {
  position: sticky;
  top: 0;
}
```

- 평소에는 `static`처럼 동작하다가,
- 스크롤이 특정 지점에 도달하면 `fixed`처럼 고정됨
- **부모 요소의 `overflow`가 `visible`이어야 작동**

---

4. 예제: 고정 네비게이션 바

```
.navbar {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  background: #333;
  color: white;
}
```

```
<div class="navbar">고정된 상단 메뉴</div>
```

---

5. 예제: 카드 안에서 절대 위치 버튼

```
<div class="card">
  <button class="close">X</button>
</div>
```

```
.card {
  position: relative;
}
.close {
  position: absolute;
  top: 10px;
  right: 10px;
}
```

---

6. 정리 표

값	기준	흐름 포함 여부	주 용도
`static`	문서 흐름	포함	기본 배치
`relative`	자기 자신	포함	살짝 이동
`absolute`	가장 가까운 위치 지정 조상	제외	특정 위치 배치
`fixed`	뷰포트	제외	고정 UI 요소
`sticky`	뷰포트 (조건부)	포함	스크롤 고정 헤더 등

---

7. 주의점

- `absolute` 요소는 부모 중 `position` **이 지정된 요소**가 없으면 `body` 기준이 됨
- `fixed` 는 **모바일 환경에서 뷰포트 문제**를 일으킬 수 있음
- `sticky` 는 **브라우저 지원과 overflow 속성**에 민감

결론

> `position` 속성은 CSS 레이아웃의 **정밀 제어를 가능하게 해주는 강력한 도구**이다.
> 특히, `absolute`, `fixed`, `sticky`는 **기본 흐름을 깨고 직접 위치를 지정**할 수 있으므로 UI 설계 시 핵심적으로 사용된다.

`z-index`

`z-index`: 요소의 앞뒤 순서 지정

`z-index`는 CSS에서 요소들의 **겹침 순서(z축 순서)**를 지정하는 속성이다.
2D 평면 상에서의 깊이를 다루며, 숫자가 클수록 **앞쪽(사용자에 가까운 쪽)**에 표시된다.

---

1. 기본 개념

- `z-index` 는 **position 속성값이** `relative`, `absolute`, `fixed`, `sticky` **중 하나**일 때만 동작한다.
- 기본적으로 HTML 요소는 **작성 순서(HTML 구조 순)**에 따라 위에서 아래로 쌓인다.
- `z-index` 를 설정하면 이 기본 쌓임 순서를 변경할 수 있다.

---

2. 문법

```
selector {
  position: relative | absolute | fixed | sticky;
  z-index: <정수>;
}
```

```
.box {
  position: absolute;
  z-index: 10;
}
```

---

3. 예제

```
<div class="box1">박스 1</div>
<div class="box2">박스 2</div>
```

```
.box1 {
  position: absolute;
  z-index: 1;
  background: red;
}

.box2 {
  position: absolute;
  z-index: 2;
  background: blue;
}
```

→ `box2`가 `box1` 위에 그려짐 (z-index가 더 높기 때문)

---

4. 기본값

- `z-index`의 기본값은 `auto`
- `auto`는 **부모의 stacking context**를 따름

---

5. 음수도 가능

```
.underlay {
  position: absolute;
  z-index: -1;
}
```

- z-index가 음수면 **뒤쪽(배경에 가까운 쪽)**으로 배치됨

---

6. stacking context (쌓임 맥락)

`z-index`는 **"쌓임 맥락(stacking context)"**이라는 개념 하에서 동작함.

새 stacking context를 생성하는 조건

조건	설명
`position`이 `relative`, `absolute`, `fixed`, `sticky` 중 하나이고 `z-index` 설정됨	✅
`display: flex`, `grid`에서 자식에 `z-index` 설정	✅
`opacity < 1`, `filter`, `transform` 등 효과 사용	✅
`will-change`, `mix-blend-mode` 등 설정	✅

→ **서로 다른 stacking context의 자식 요소끼리는 `z-index`로 직접 비교할 수 없음**

---

7. 실전 팁

- **모달 창**이나 **드롭다운**, **툴팁** 등에서는 `z-index`를 이용해 위로 띄워야 함
- `z-index` 충돌이 잦다면 **명시적인 컨텍스트 분리**가 필요함

```
.modal-container {
  position: relative;
  z-index: 1000;
}

.modal-backdrop {
  position: absolute;
  z-index: 999;
}
```

---

8. 시각적 예

```
z-index: 3   ——> 맨 앞 (사용자에게 가장 가까움)
z-index: 2
z-index: 1
z-index: 0
z-index: -1  ——> 맨 뒤 (배경에 가까움)
```

---

9. 자주 하는 실수

- `z-index`가 먹히지 않는다면?
  → `position` 속성이 `static`일 확률이 높다

```css
/* 안 먹는 예 */
.box {
  z-index: 10;  /* ❌ position이 static이라 무시됨 */
}

/* 제대로 된 예 */
.box {
  position: relative;
  z-index: 10;
}
```

결론

> `z-index`는 요소의 앞뒤 배치를 제어하는 속성으로,
> **시각적으로 겹치는 요소들 사이의 우선순위 조정**에 필수다.
> `stacking context` 개념과 함께 이해하면 복잡한 UI에서도 정확한 겹침 제어가 가능하다.

`overflow: hidden, scroll, auto`

`overflow`: 콘텐츠 넘침 처리 속성

`overflow`는 요소의 **콘텐츠가 지정된 너비·높이를 초과했을 때**,
화면에 어떻게 표시할지를 결정하는 속성이다.

이는 특히 `width`나 `height`가 고정된 박스 안에서
텍스트, 이미지, 요소들이 넘칠 때 **스크롤바, 잘림, 자동 처리**를 어떻게 할지 지정할 수 있게 해준다.

---

1. 문법

```css
selector {
  overflow: visible | hidden | scroll | auto;
}
```

또는 수직/수평을 별도로 지정 가능:

```css
overflow-x: auto;
overflow-y: hidden;
```

---

2. 주요 값 설명

값	설명
`visible` (기본값)	넘치는 콘텐츠를 **그대로 보여줌** (잘리지 않음, 스크롤 없음)
`hidden`	넘치는 콘텐츠를 **잘라냄** (스크롤 없음)

값	설명
`scroll`	콘텐츠가 넘치든 말든 **항상 스크롤바 표시**
`auto`	콘텐츠가 넘칠 때만 **자동으로 스크롤바 표시**

---

3. 예제

▸ `overflow: hidden`

```
.box {
  width: 200px;
  height: 100px;
  overflow: hidden;
}
```

→ 영역을 벗어나는 텍스트나 이미지가 **보이지 않게 잘림**

▸ `overflow: scroll`

```
.box {
  width: 300px;
  height: 150px;
  overflow: scroll;
}
```

→ 넘치든 말든 **수직·수평 스크롤바 항상 표시**

▸ `overflow: auto`

```
.box {
  width: 300px;
  height: 150px;
  overflow: auto;
}
```

→ 콘텐츠가 넘칠 경우에만 스크롤 생성 (필요할 때만)

▸ 수직 스크롤만 적용

```
.box {
  height: 100px;
  overflow-y: auto;
  overflow-x: hidden;
}
```

→ 수직은 넘칠 때만 스크롤, 수평은 무조건 잘림

---

4. 스크롤 처리 시 주의사항

- 스크롤이 생기는 요소는 **스크롤 컨테이너**가 됨
- 자식 요소의 `position: sticky`가 **동작하지 않을 수 있음**
- `overflow: hidden`은 **마진 겹침 방지**에도 사용되기도 함 (마진 캐치 기법)

---

5. 실전 팁

상황	추천 설정
잘리는 것 없이 모두 보여주고 싶을 때	`overflow: visible`
스크롤 없이 영역을 깔끔히 유지하고 싶을 때	`overflow: hidden`
콘텐츠 넘칠 가능성 있어서 자동 스크롤 원할 때	`overflow: auto`
UI에 항상 스크롤 보여야 할 때	`overflow: scroll`

---

6. 브라우저 기본값

- `overflow` 기본값은 `visible` → 콘텐츠가 넘쳐도 계속 보여짐

---

7. 예제 HTML

```
<div class="box">
  Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio. Praesent libero. Sed cursus ante dapibus diam.
</div>
```

```
.box {
  width: 200px;
  height: 50px;
  border: 1px solid black;
  overflow: auto;
}
```

결론

> `overflow`는 박스 모델에서 **넘치는 콘텐츠를 어떻게 다룰지 제어하는 도구**다.
> 시각적 정돈, 스크롤 구현, 클리핑 처리 등 다양한 상황에서 활용된다.
> `hidden`, `auto`, `scroll`의 차이를 명확히 이해하고 **UI에 맞게 조절**해야 한다.