M03-02 · GIBALJA VISUAL MANUAL

CSS 레이아웃과 반응형 판단하기

한 문장 목표: 화면이 왜 커지고 밀리고 줄 바뀌며 넘치는지 상자·기준·흐름·축·트랙·넘침·변화의 일곱 질문으로 판독하고, 여러 화면 폭의 증거가 있는 반응형 점검표를 만듭니다.

난이도	개념	실습	셀프 테스트	최종 산출물
Level 1	60분	55분	15분	CSS 레이아웃 지도, 반응형 점검표, 뷰포트별 검수 증거

"모바일에서 깨집니다"만 전달하면 개발자는 같은 화면을 다시 찾아야 합니다. "390 CSS px에서 결과 카드의 최소 폭 420px이 부모 358px보다 커서 문서 가로 스크롤이 62px 생깁니다"라고 쓰면 문제 폭·선택 요소·계산값·영향·완료 기준이 한 번에 연결됩니다.

그림 1. 레이아웃은 상자의 실제 크기에서 시작해 반응형 변화의 조건과 증거까지 일곱 질문으로 읽습니다.

1

상자

실제 크기는?

2

기준

누구의 폭을 따르나?

5

트랙

Grid 몇 줄?

6

넘침

첫 원인은 어디?

판정 = 계산된 CSS + 실제

3

흐름

앞뒤와 밀어내나?

4

축

Flex 어느 방향?

7

변화

어느 조건에 바뀌나?

치수 + 여러 폭의 화면 증거

1. 이 PDF를 공부하는 방법

1회차 · 일곱 질문만 익히기 · 15분

그림 1을 보며 `상자 → 기준 → 흐름 → 축 → 트랙 → 넘침 → 변화`를 소리 내어 읽습니다. 속성 이름을 외우기보다 각 질문이 무엇을 확인하는지 한 문장으로 말합니다.

2회차 · 레이아웃 규칙 읽기 · 30분

박스 모델·일반 흐름·Flexbox·Grid 그림을 보며 부모와 자식의 관계를 찾습니다. `width` 하나보다 `display`, 기준 상자, 최소 크기, 콘텐츠 길이를 함께 봅니다.

3회차 · 레이아웃 실험실로 깨짐 재현하기 · 30분

폭 슬라이더를 320~1280 CSS px 사이로 움직입니다. 고정 폭·줄 바꿈 금지·Flex 최소 크기·절대 위치 시나리오에서 첫 넘침 요소와 계산값을 기록합니다.

4회차 · Chrome에서 실제 화면 검수하기 · 40분

Device mode와 Elements를 함께 사용합니다. 중단점 바로 전·경계·바로 뒤를 확인하고, 화면 캡처만이 아니라 선택 요소·계산된 CSS·실제 치수도 남깁니다.

5회차 · 셀프 테스트 · 15분

정답을 가리고 10문제를 풉니다. 틀린 문제는 그림 11의 일곱 질문 중 빠뜨린 질문을 표시합니다.

학습 완료 기준
“태블릿에서 카드가 이상하다”를 “뷰포트 768px에서 3열 Grid의 첫 열이 긴 식별자의 min-content 폭 아래로 줄지 않아 body의 scrollWidth가 clientWidth보다 48px 큽니다. 해당 트랙과 문자열 줄바꿈을 수정하고 767·768·769px 및 200% 확대에서 정보 손실과 이중 스크롤이 없는 것을 완료 기준으로 합니다”처럼 설명할 수 있습니다.

2. CSS 레이아웃은 상자 트리를 배치하는 규칙입니다

M03-01에서 HTML과 문서 객체 모델(Document Object Model, DOM)을 내용의 의미와 관계로 읽었습니다. CSS(Cascading Style Sheets)는 그 DOM에서 만들어진 상자들이 어떤 크기와 위치로 그려지는지 정합니다.

```
DOM 요소 트리
   ↓ display와 상자 생성
CSS 상자 트리
   ↓ 흐름·Flexbox·Grid·위치 지정
사용된 크기와 위치
   ↓ 페인트·합성
화면 픽셀
```

`display`는 시각적 배치 방식을 바꾸지만 HTML 의미 자체를 바꾸지는 않습니다. 예를 들어 `nav { display: grid }`라고 해도 `nav`의 탐색 의미는 그대로입니다.

2.1. 레이아웃 판독의 세 가지 증거

증거	질문	Chrome에서 보는 곳
선언된 규칙	작성자는 무엇을 지정했는가	Elements > Styles
계산된 값	충돌과 상속 뒤 어떤 값이 적용됐는가	Elements > Computed
실제 기하	화면에서 몇 px이고 어디에 있는가	Box model· `getBoundingClientRect()`

Styles에서 취소선이 그어진 `width: 420px`은 선언됐지만 적용되지 않은 값입니다. Computed의 `width`는 계산 결과이고, 실제 테두리 상자의 폭은 `getBoundingClientRect().width`로 확인할 수 있습니다. 셋을 섞지 않습니다.

```
const el = $0;
const css = getComputedStyle(el);
const rect = el.getBoundingClientRect();
({
  display: css.display,
  width: css.width,
  minWidth: css.minWidth,
  overflowX: css.overflowX,
  borderBoxWidth: rect.width
});
```

2.2. 일곱 질문

순서	질문	대표 증거
1	선택 요소의 실제 상자는 얼마나 큰가	content·padding·border·margin· `rect`
2	그 크기는 어느 상자를 기준으로 계산됐는가	containing block·%· `max-width`
3	일반 흐름 안에서 앞뒤 요소를 밀어내는가	DOM 순서· `position` ·겹침
4	Flexbox라면 주축과 교차축은 어느 방향인가	`flex-direction` · `wrap` ·정렬
5	Grid라면 행·열 트랙은 어떻게 계산되는가	`grid-template-*` · `fr` · `minmax()`
6	경계를 처음 넘은 요소와 규칙은 무엇인가	`scrollWidth` ·요소 사각형·긴 콘텐츠
7	어느 공간 조건에서 배치가 바뀌는가	`@media` · `@container` ·중단점 경계

BIG IDEA 01

레이아웃 문제는 “보이는 위치”만 찍지 않고, 그 위치를 만든 부모 규칙과 콘텐츠 크기까지 거슬러 올라가야 설명됩니다.

30초 확인

Elements의 Styles에는 `width: 420px` 이 있지만 취소선입니다. 문제 보고서에 “폭 420px”이라고 써도 될까요?

정답 보기

안 됩니다. 취소선은 다른 규칙에 밀려 적용되지 않았다는 뜻입니다. Computed의 적용값과 실제 `getBoundingClientRect().width`를 확인해 선언값·계산값·실제 치수를 구분합니다.

3. 박스 모델로 실제 크기를 계산합니다

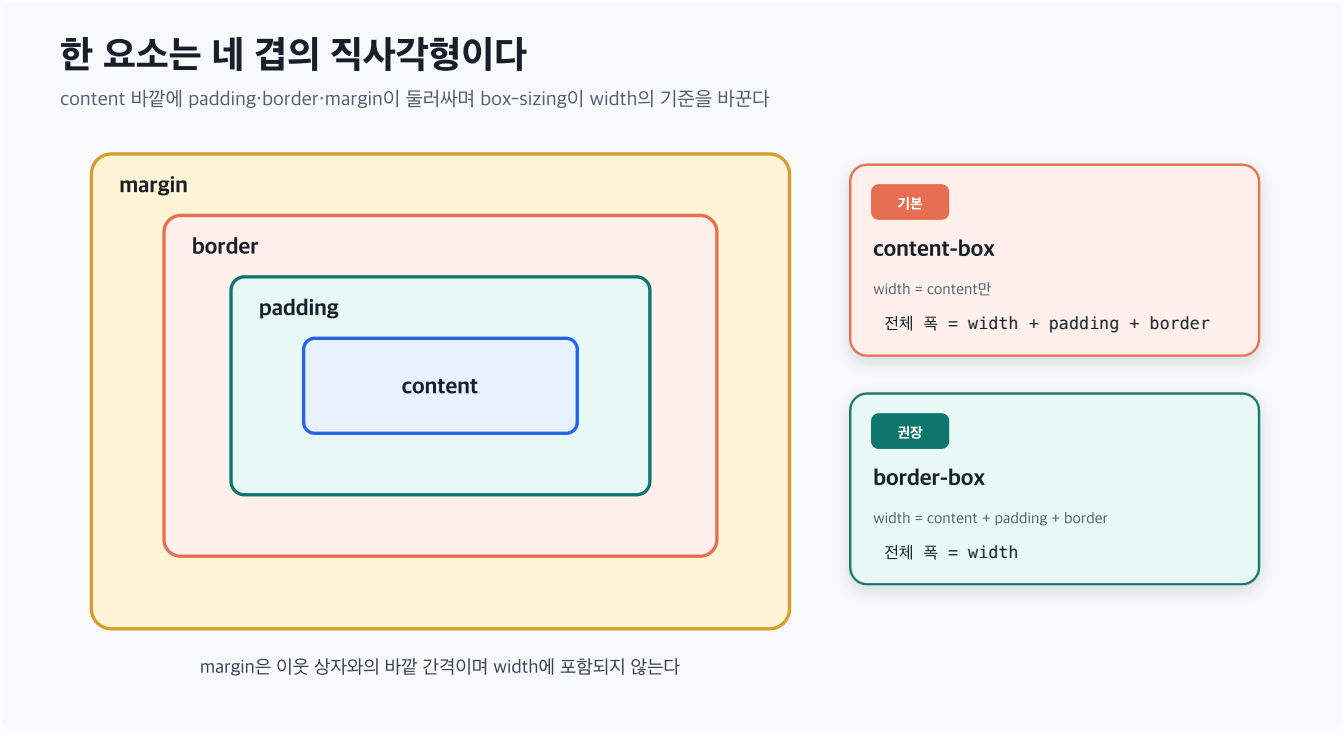
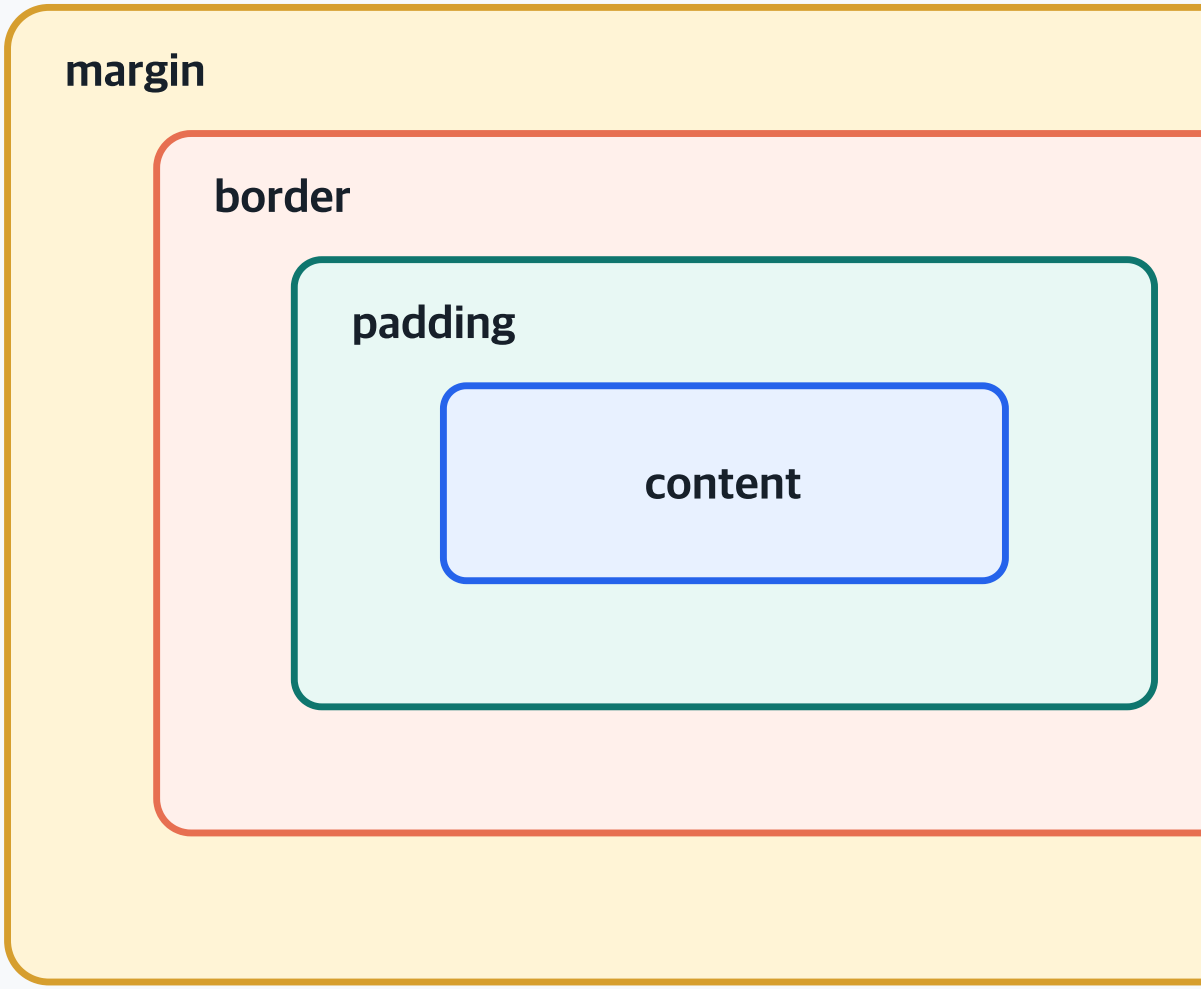
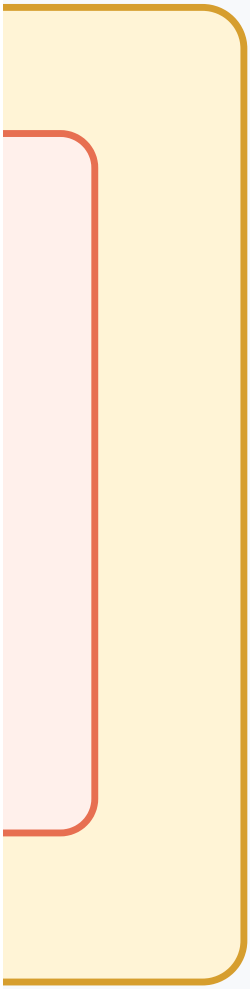


그림 2. 하나의 CSS 상자는 content·padding·border·margin 네 영역으로 읽고, `box-sizing`에 따라 `width`가 포함하는 범위를 구분합니다.


margin
border
padding
content



기본

content-box

width = content만

```
전체 폭 = width + padding + border
```

권장

border-box

width = content + padding + border

```
전체 폭 = width
```

3.1. 네 영역

- **content:** 글·이미지·자식 상자가 놓이는 내용 영역
- **padding:** content와 border 사이의 안쪽 여백
- **border:** 상자의 테두리
- **margin:** 이웃 상자와 떨어지는 바깥 여백

`margin`은 배경색이 칠해지는 상자 안이 아니며, `width`에 포함되지 않습니다. 일반 블록 흐름에서는 위·아래 margin이 합산되지 않고 접힐 수 있으므로 눈에 보이는 간격만 더해 단정하지 않습니다.

3.2. `content-box`와 `border-box`

```
.card-a {
  box-sizing: content-box;
  width: 300px;
  padding: 20px;
  border: 2px solid;
}

.card-b {
  box-sizing: border-box;
  width: 300px;
  padding: 20px;
  border: 2px solid;
}
```

상자	지정한 `width`	테두리 바깥 폭
`.card-a`	content 300px	300 + 40 + 4 = 344px
`.card-b`	content+padding+border 300px	300px

많은 프로젝트가 전체 요소에 `box-sizing: border-box`를 적용하지만 반드시 Computed에서 확인합니다.

```
*, *::before, *::after {
  box-sizing: border-box;
}
```

3.3. 폭보다 최소·최대 조건을 같이 봅니다

`width: 100%`만으로 유동적이라고 단정할 수 없습니다. 다음 속성이 결과를 제한할 수 있습니다.

규칙	작용
`min-width`	이 값보다 작아지지 않도록 제한
`max-width`	이 값보다 커지지 않도록 제한
`padding` · `border`	`content-box`에서 지정 폭 바깥에 추가
`box-sizing`	`width`가 포함하는 영역 변경
긴 콘텐츠	min-content 크기를 키워 축소 방해

3.4. 논리 축으로도 읽습니다

가로쓰기 한국어 화면에서는 보통 inline 축이 좌우, block 축이 위아래입니다. `inline-size`는 쓰기 방향에 따른 가로 폭, `block-size`는 세로 흐름 크기입니다.

```
.article {
  max-inline-size: 72rem;
  margin-inline: auto;
  padding-inline: 1rem;
}
```

이 표기는 좌우만 전제하는 `max-width`, `margin-left`, `margin-right`보다 쓰기 방향 변화에 유연합니다. 초급 단계에서는 두 표현의 대응을 알면 충분합니다.

4. 크기의 기준이 되는 containing block을 찾습니다

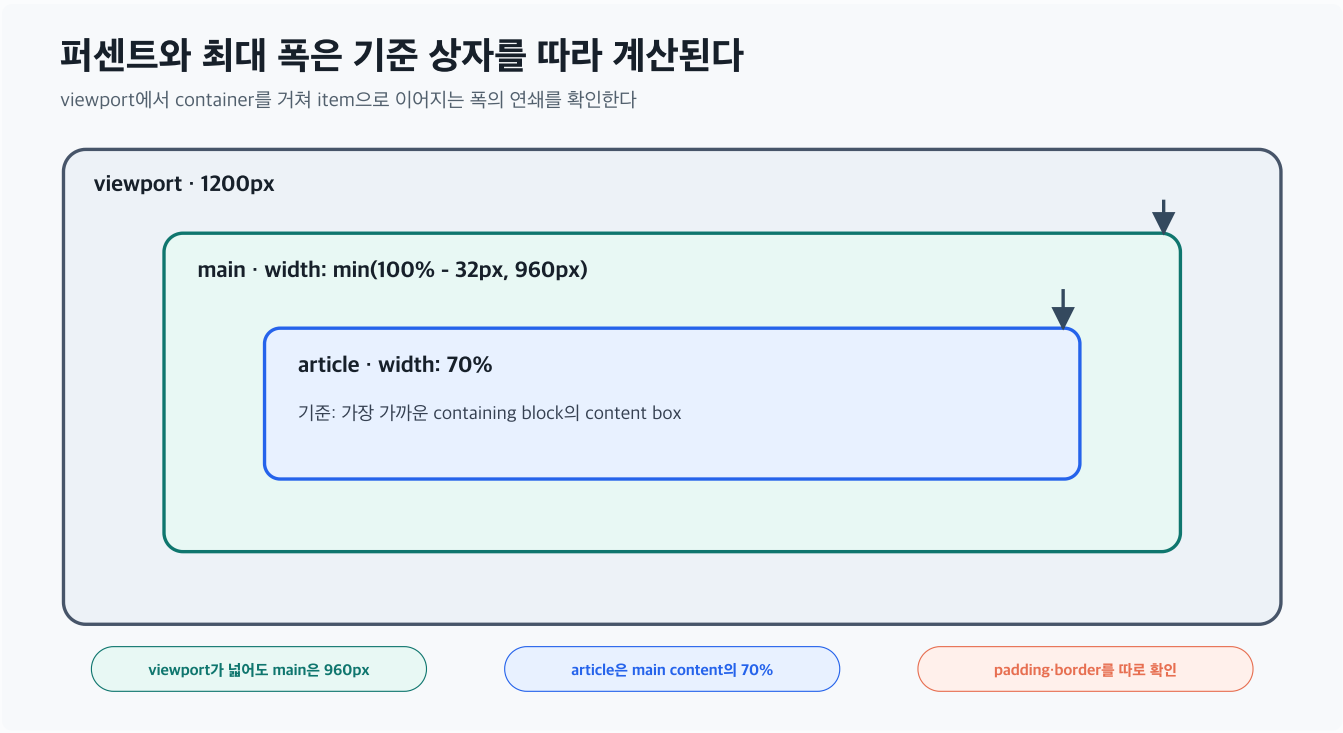
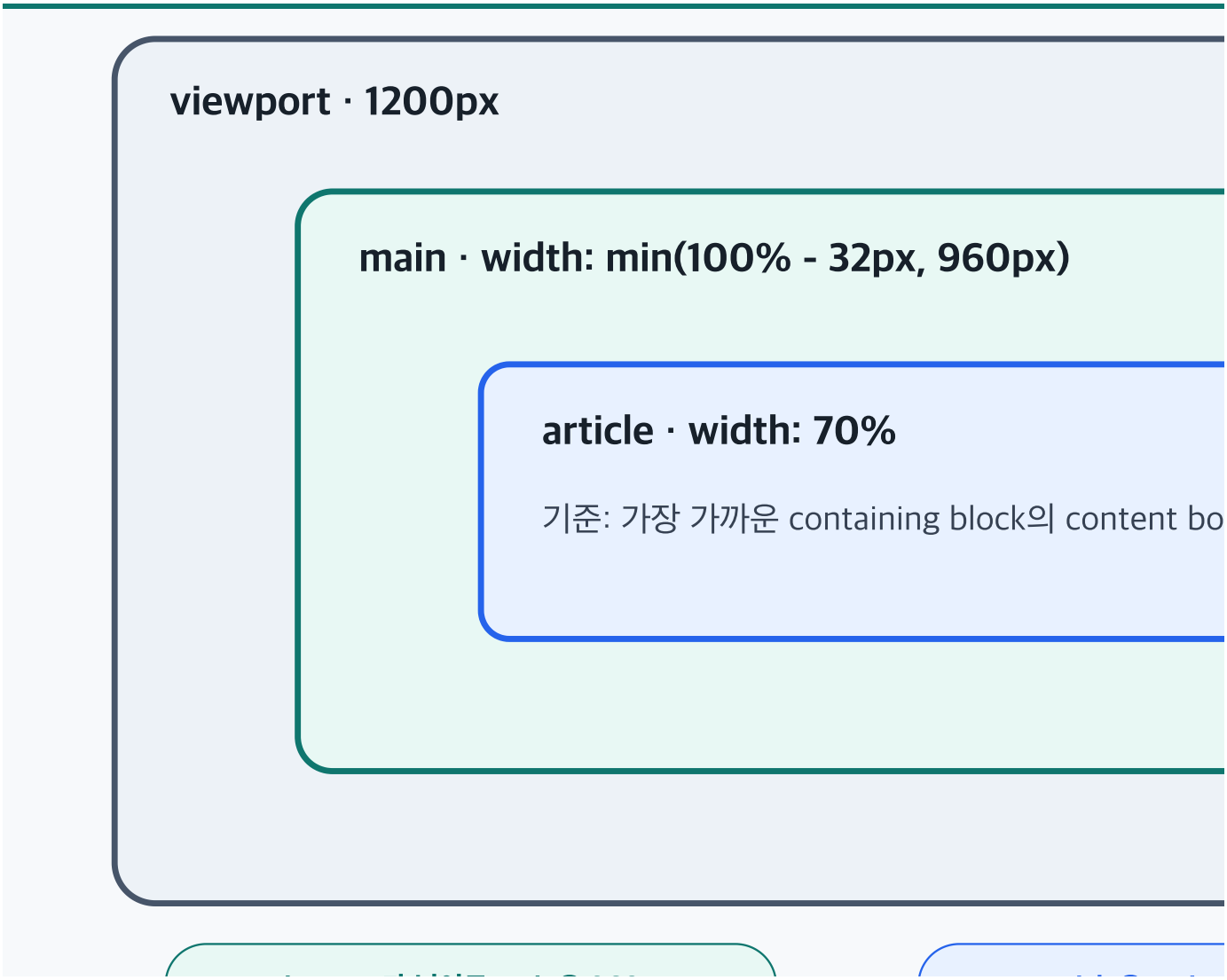
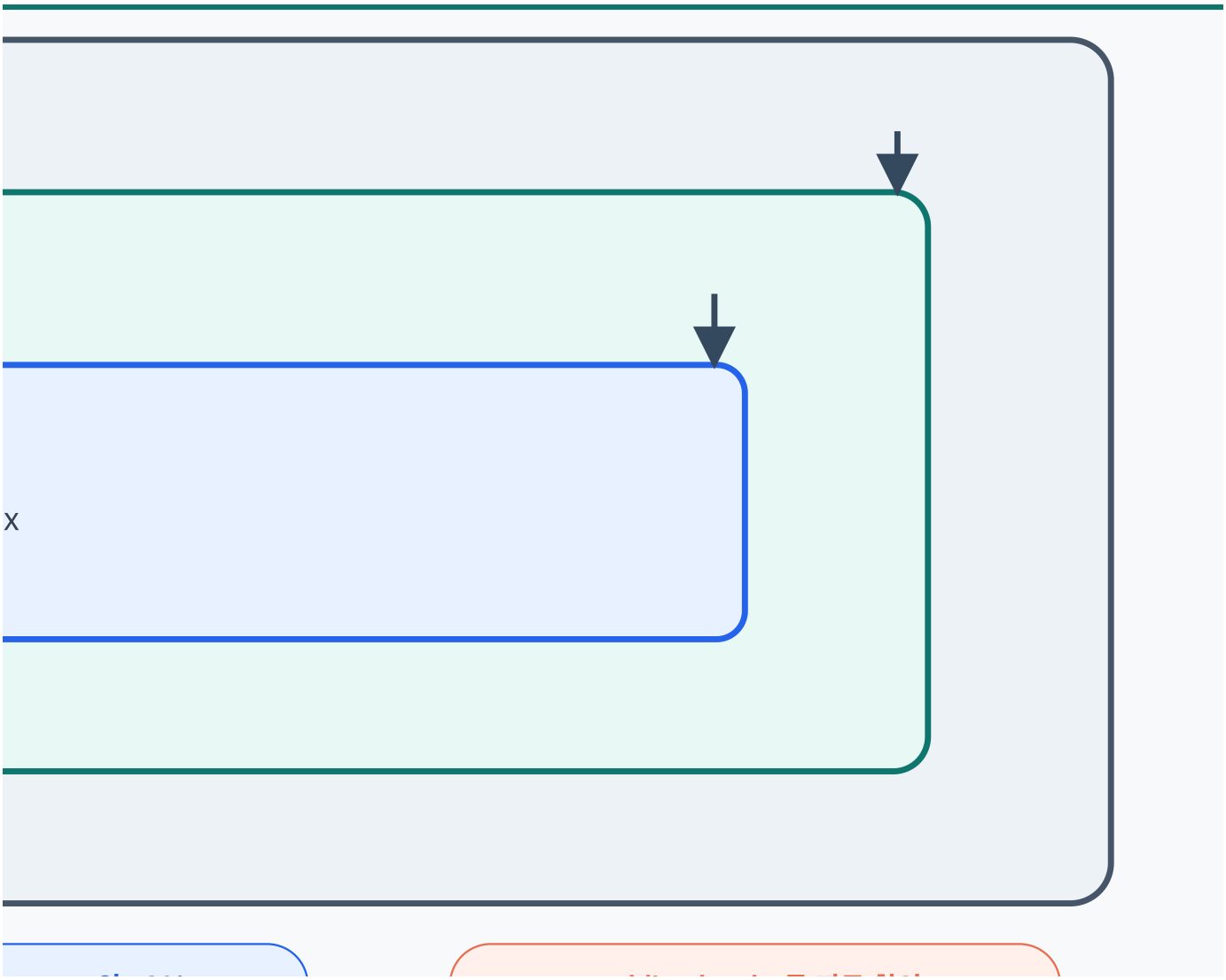


그림 3. 퍼센트 폭은 화면 전체가 아니라 해당 요소의 기준 상자를 따라 계산되므로 viewport→부모→자식의 연쇄를 읽습니다.


viewport · 1200px
main · width: min(100% - 32px, 960px)
article · width: 70%
기준: 가장 가까운 containing block의 content bo


X

4.1. 퍼센트는 항상 “무엇의 퍼센트인가”를 묻습니다

`article { width: 70% }`에서 70%가 viewport의 70%라고 단정하면 안 됩니다. 보통 그 요소를 포함하는 containing block의 content box가 기준입니다.

```
main {
  width: min(calc(100% - 2rem), 60rem);
  margin-inline: auto;
}

article {
  width: 70%;
}
```

viewport가 1200px이어도 `main`은 최대 960px입니다. `article`은 `main`의 사용 가능한 content 폭을 기준으로 계산됩니다.

4.2. 고정 폭·유동 폭·제한 폭을 구분합니다

종류	예	좁은 화면에서
고정 폭	`width: 960px`	부모보다 커질 수 있음
유동 폭	`width: 100%`	부모 폭을 따라감
제한 폭	`width: min(100%, 60rem)`	좁을 때 줄고 넓을 때 상한 유지
최소 제한	`min-width: 420px`	420px 아래로 줄지 않음
콘텐츠 기반	`width: max-content`	줄 바꿈 없이 필요한 폭을 요구할 수 있음

4.3. `position`은 기준 상자를 바꿀 수 있습니다

`position: absolute`인 요소는 보통 가장 가까운 위치 지정 조상, 즉 `position`이 `static`이 아닌 조상의 padding box를 기준으로 배치됩니다. 그런 조상이 없으면 초기 containing block까지 올라갈 수 있습니다.

```
.card { position: relative; }
.badge { position: absolute; inset-block-start: 8px; inset-inline-end: 8px; }
```

배지가 카드가 아니라 화면 구석에 붙는다면 `.card`가 실제 위치 지정 조상인지 확인합니다.

30초 확인

`width: 100%`인 입력 상자가 부모보다 32px 넓습니다. 가장 먼저 무엇을 함께 확인해야 할까요?

정답 보기

`box-sizing`, 좌우 padding·border, 부모의 content 폭을 확인합니다. `content-box`이면 100% content 폭 바깥에 padding과 border가 더해질 수 있습니다.

5. 일반 흐름과 위치 지정을 구분합니다

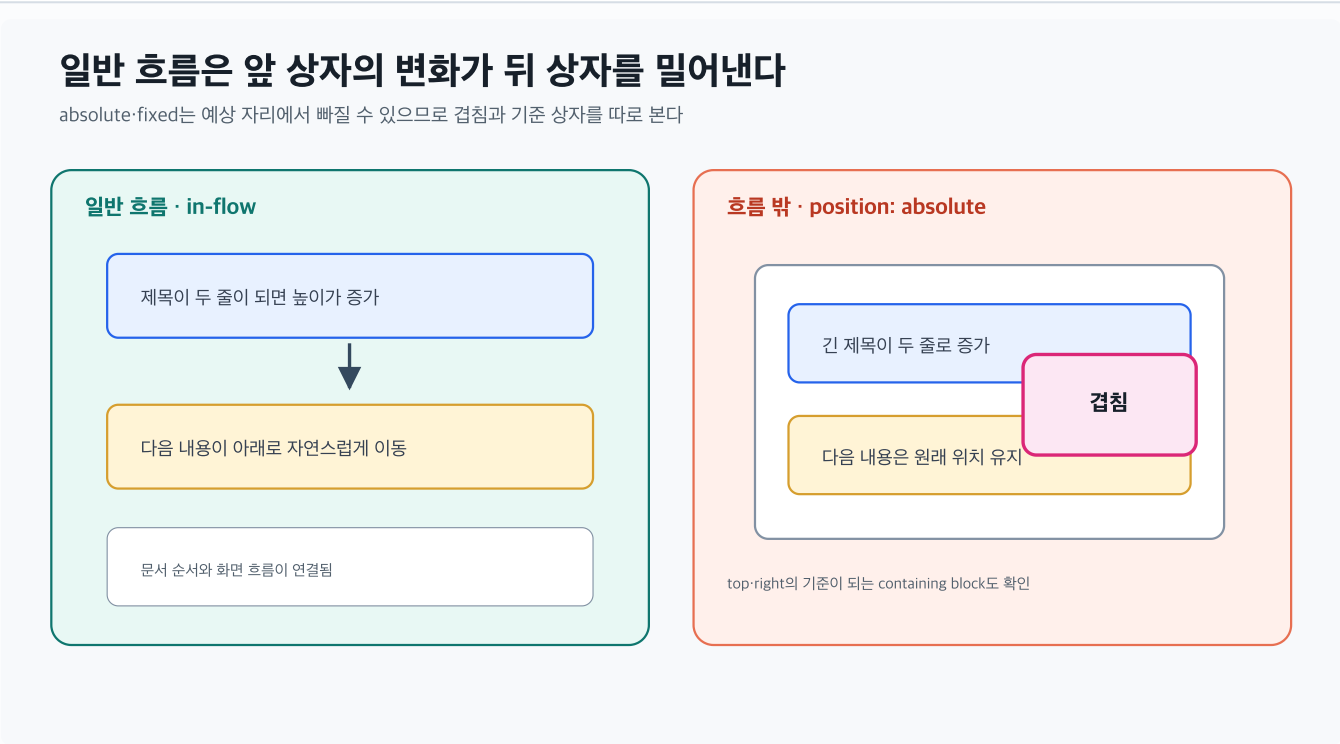
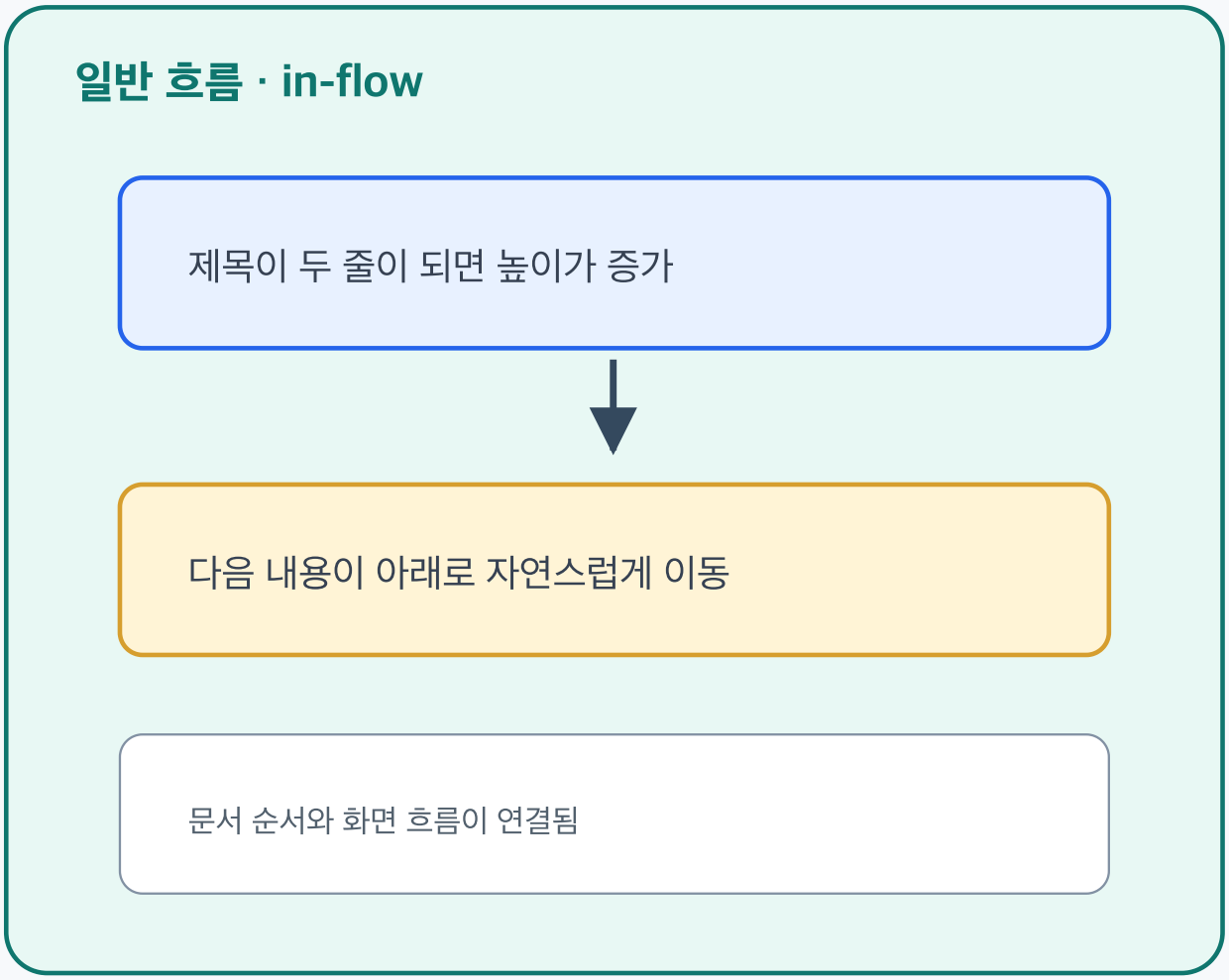
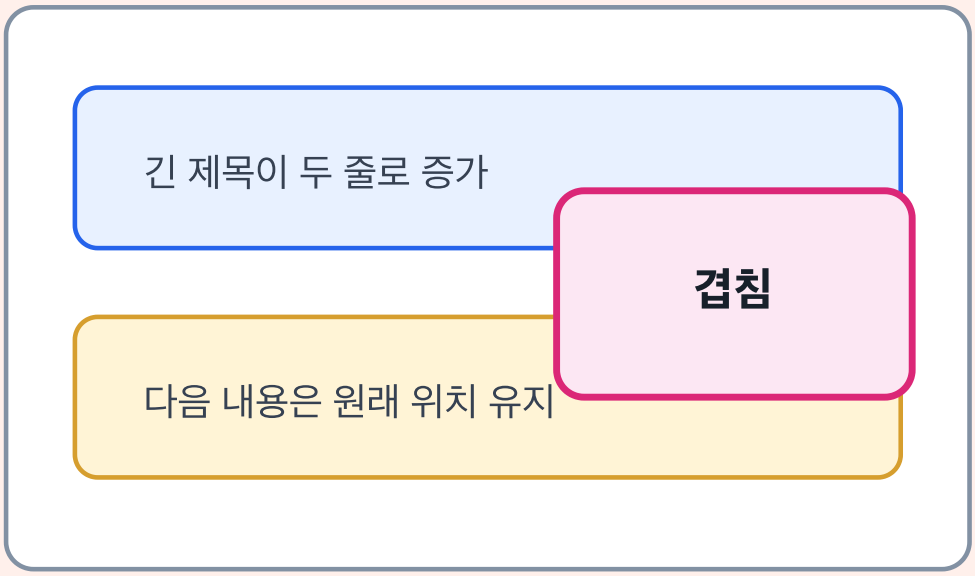


그림 4. 일반 흐름의 상자는 높이 변화가 뒤 상자를 밀어내지만, 흐름 밖 요소는 자리를 차지하지 않아 겹칠 수 있습니다.


일반 흐름 · in-flow
제목이 두 줄이 되면 높이가 증가
다음 내용이 아래로 자연스럽게 이동
문서 순서와 화면 흐름이 연결됨

흐름 밖 · position: absolute



top·right의 기준이 되는 containing block도 확인

5.1. 일반 흐름이 안전한 기본입니다

일반 흐름(normal flow)에서 블록 상자는 보통 block 축으로 차례로 놓이고, 문장과 inline 상자는 줄 안에서 흐릅니다. 제목이 두 줄이 되면 높이가 늘고 다음 내용은 아래로 이동합니다.

콘텐츠 길이와 글자 확대에 자연스럽게 반응하므로, 배치를 만들 때 먼저 일반 흐름·Flexbox·Grid 안에서 해결할 수 있는지 봅니다.

5.2. 네 가지 `position`을 읽습니다

값	일반 흐름 자리	기준·특징
`static`	유지	기본 배치
`relative`	유지	원래 자리를 남긴 채 자신을 기준으로 이동 가능
`absolute`	제거	위치 지정 조상을 기준으로 배치
`fixed`	제거	보통 viewport에 고정
`sticky`	유지 후 고정	스크롤 경계와 가장 가까운 스크롤 컨테이너 영향

5.3. 겹침은 `z-index`만의 문제가 아닙니다

버튼이 제목을 덮는다면 다음 순서로 봅니다.

1. 버튼이나 제목이 흐름 밖인지 확인
2. 어느 containing block을 기준으로 좌표가 계산됐는지 확인
3. 제목이 길어졌을 때 부모 높이가 늘어나는지 확인
4. 겹친 요소의 stacking context와 `z-index` 확인
5. 키보드 초점과 클릭 영역이 가려지지 않는지 확인

`z-index`를 높이면 앞에 보일 수 있지만, 잘못된 좌표와 부족한 공간은 해결되지 않습니다.

5.4. 화면 순서와 DOM 순서를 비교합니다

Flexbox의 `order`, Grid 배치, absolute 좌표는 시각 순서를 바꿀 수 있습니다. 그러나 키보드 초점과 읽기 순서는 DOM 순서의 영향을 받습니다.

확인할 세 순서

1. DOM 소스 순서
2. 화면의 시각 순서
3. Tab 키 초점 순서

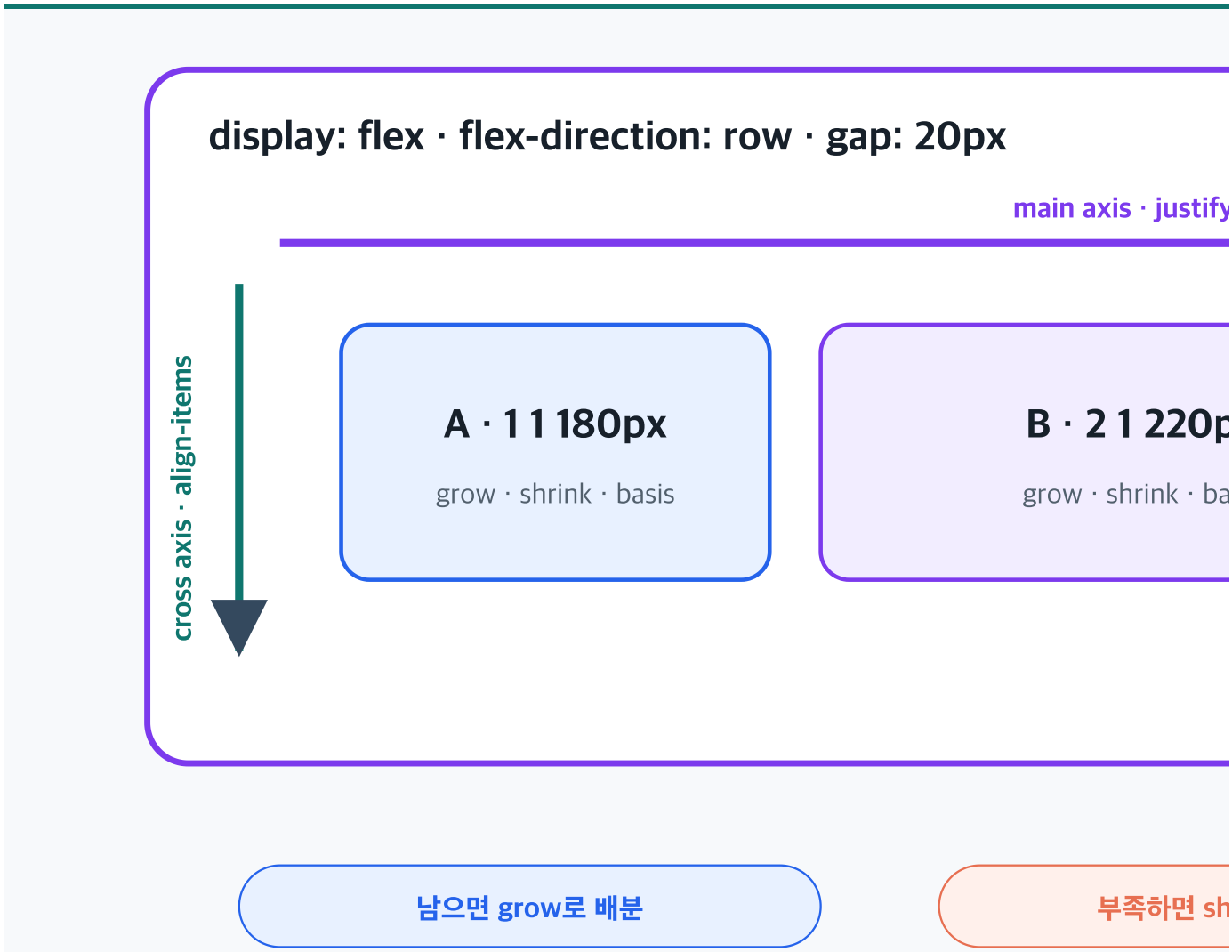
화면만 보고 “왼쪽에서 오른쪽”으로 요구사항을 쓰지 말고 M03-01의 DOM 구조표와 함께 확인합니다.

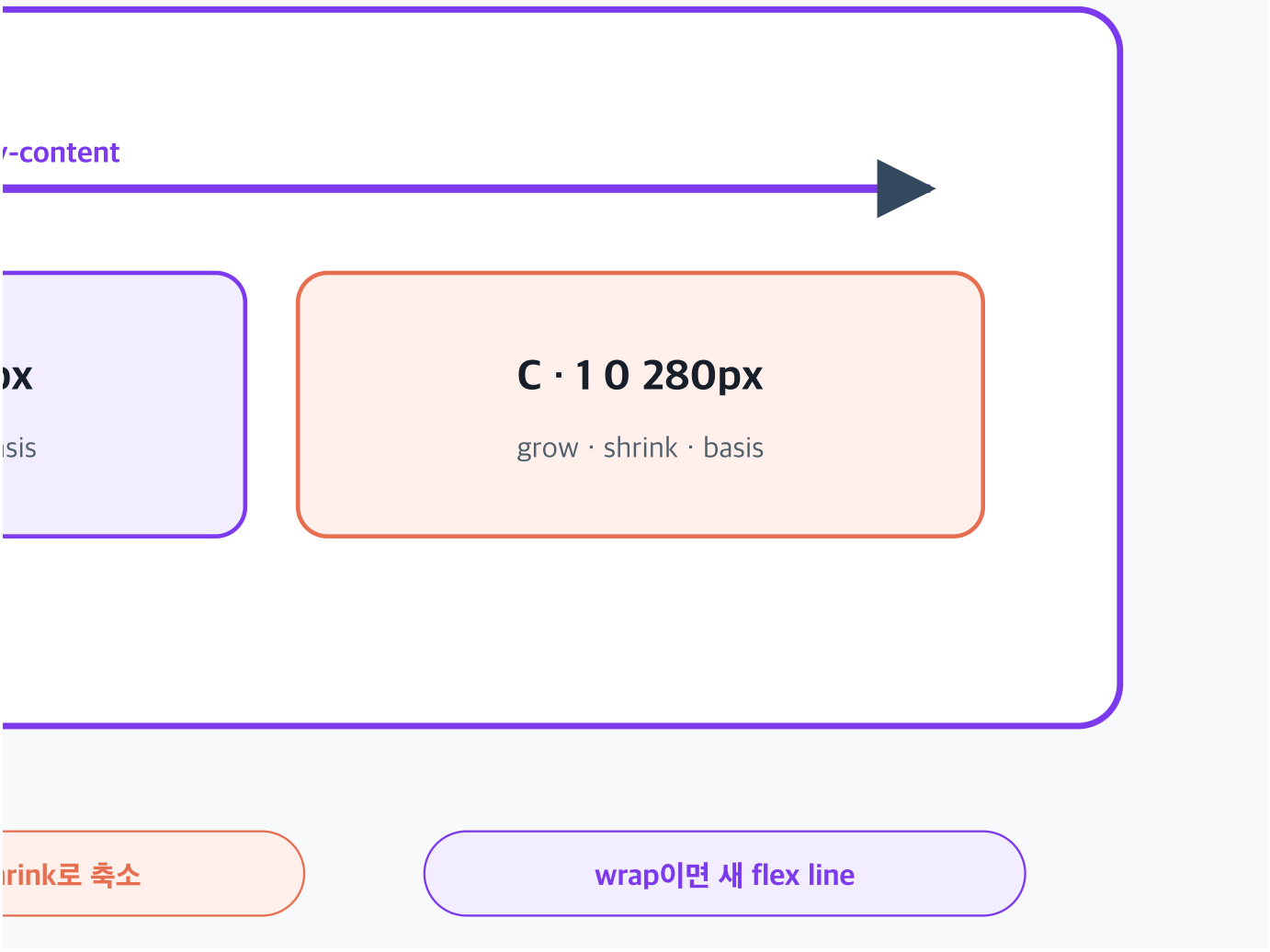
BIG IDEA 02

콘텐츠가 늘어날 때 뒤 요소가 자연스럽게 밀리는 구조가 먼저입니다. 흐름 밖 배치는 배지·팝오버처럼 겹침 자체가 목적일 때 근거를 갖고 사용합니다.

6. Flexbox는 한 축의 공간 배분으로 읽습니다

그림 5. Flexbox는 주축의 기준 크기를 놓고 남는 공간과 부족한 공간을 항목 사이에 배분합니다.


display: flex · flex-direction: row · gap: 20px
main axis · justify
cross axis · align-items
A · 1 1 180px
grow · shrink · basis
B · 2 1 220p
grow · shrink · ba
남으면 grow로 배분
부족하면 sh


-content
px
sis
C · 1 0 280px
grow · shrink · basis
rink로 축소
wrap이면 새 flex line

6.1. 먼저 컨테이너와 항목을 찾습니다

```
.toolbar {
  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
  gap: 12px;
  align-items: center;
}
```

`.toolbar`가 flex container이고 바로 아래 자식들이 flex item입니다. 손자 요소는 별도 Flexbox가 아니면 이 컨테이너의 직접 항목이 아닙니다.

6.2. 주축과 교차축은 `flex-direction`에 따라 바뀝니다

`flex-direction`	주축	`justify-content`	`align-items`
`row`	inline 방향	가로 공간 배분	세로 정렬
`column`	block 방향	세로 공간 배분	가로 정렬

"가로는 justify, 세로는 align"으로 외우면 `column`에서 틀립니다. **주축은 justify, 교차축은 align**으로 읽습니다.

6.3. `flex`는 기준·증가·축소의 묶음입니다

```
.item { flex: 1 1 16rem; }
/* grow 1 · shrink 1 · basis 16rem */
```

- `flex-basis`: 공간을 나누기 전 항목의 기준 크기
- `flex-grow`: 남는 공간을 받을 비율
- `flex-shrink`: 부족한 공간을 줄이는 비율

실제 계산은 최소·최대 크기와 콘텐츠 제약을 함께 반영합니다. `flex: 1`만 보고 항상 같은 폭이라고 단정하지 않습니다.

6.4. `wrap`은 항목을 새 flex line으로 보냅니다

`flex-wrap: nowrap`이 기본입니다. 항목의 필요한 폭 합계가 컨테이너보다 크면 축소하거나 넘칠 수 있습니다.
`wrap`이면 다음 줄을 만들지만, 항목 내부의 긴 문자열과 최소 폭은 여전히 문제를 만들 수 있습니다.

6.5. Flex 항목의 자동 최소 크기를 확인합니다

긴 제목이 있는 항목이 줄지 않아 전체 화면이 넘칠 때 `min-width: auto`의 콘텐츠 기반 최소 크기가 원인일 수 있습니다.

```
.result-content {
  min-width: 0;
}
```

`min-width: 0`은 “Flexbox 문제의 만능 정답”이 아닙니다. 해당 항목이 실제로 줄어들어야 하고, 내부 콘텐츠도 안전하게 줄 바뀌거나 잘림 없이 표시될 수 있을 때 사용합니다.

30초 확인

`flex-direction: column`인 컨테이너에서 `justify-content: center`는 어느 방향을 가운데로 맞춥니까?

정답 보기
주축인 block 방향, 일반 가로쓰기에서는 세로 방향을 가운데로 맞춥니다. `justify-content`는 고정된 가로 속성이 아니라 주축 정렬입니다.

7. Grid는 행과 열의 트랙으로 읽습니다

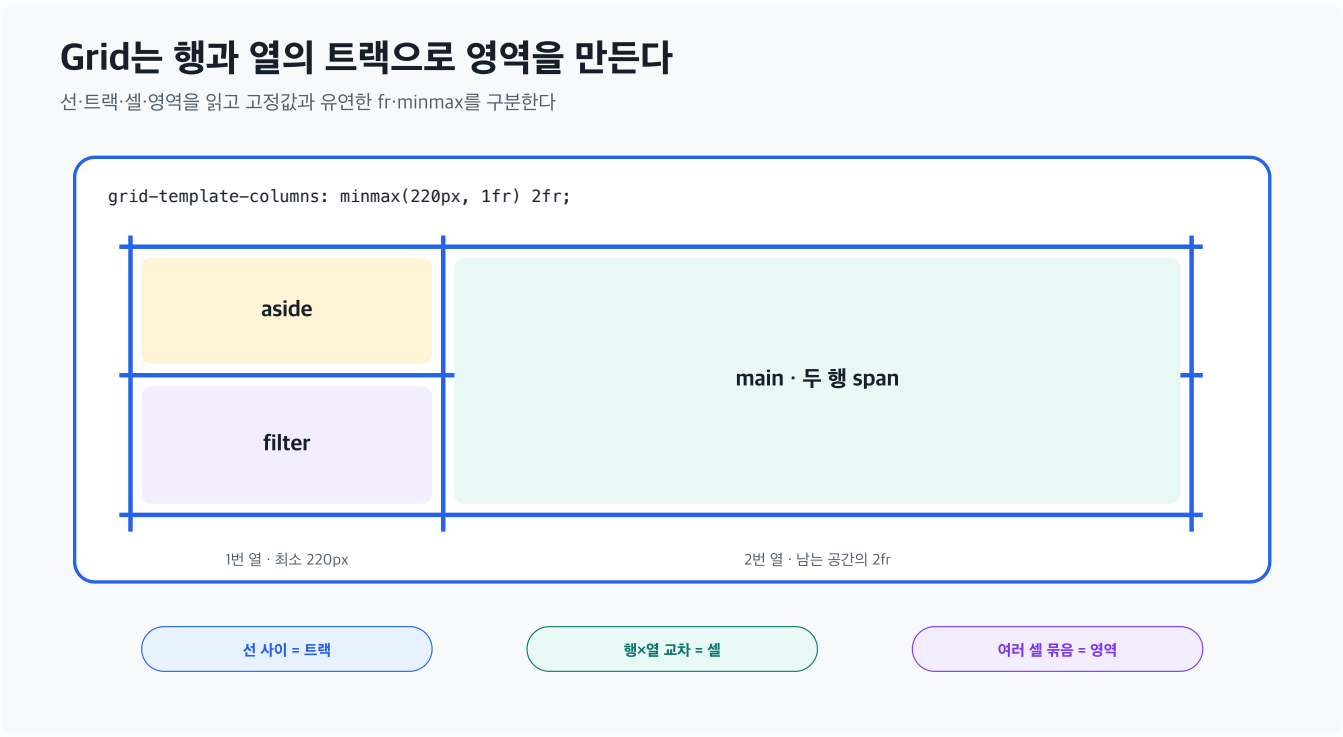
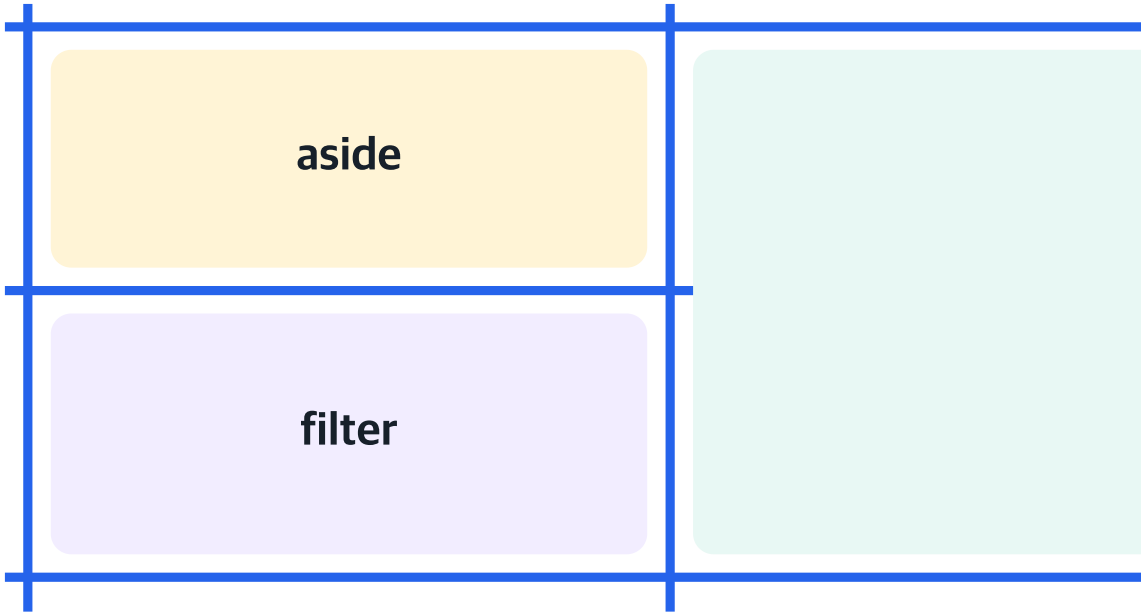


그림 6. Grid는 선 사이의 트랙, 행과 열의 교차 셀, 여러 셀을 차지하는 영역으로 배치를 만듭니다.

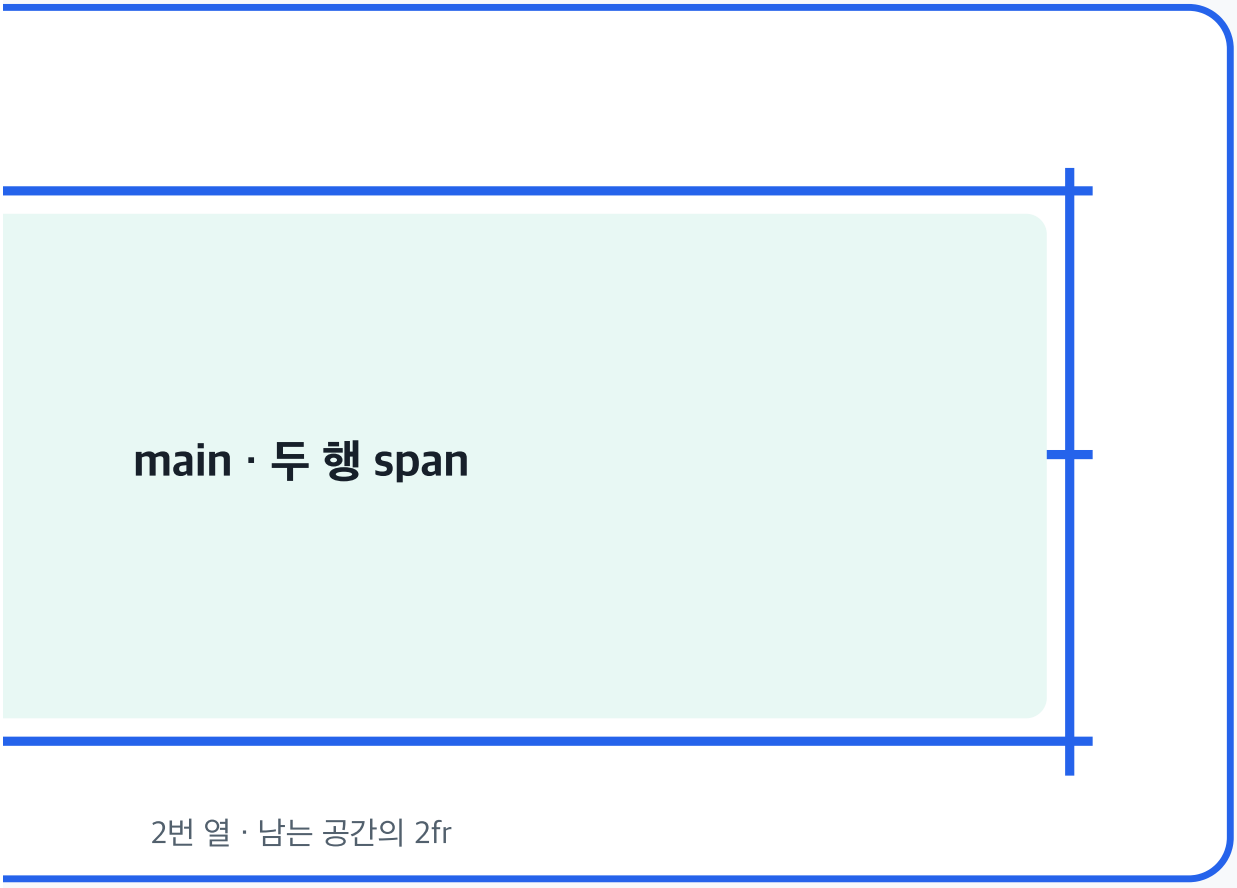
```
grid-template-columns: minmax(220px, 1fr) 2fr;
```



1번 열 · 최소 220px

선 사이 = 트랙

행×열 교



1차 = 셀

여러 셀 묶음 = 영역

7.1. Grid의 네 단위를 구분합니다

단위	뜻
grid line	행·열을 나누는 선
grid track	인접한 두 선 사이의 행 또는 열
grid cell	한 행과 한 열이 만나는 최소 단위
grid area	하나 이상의 셀로 이뤄진 직사각형 영역

```
.layout {
  display: grid;
  grid-template-columns: minmax(220px, 1fr) 2fr;
  gap: 24px;
}
```

첫 열은 최소 220px을 지키며 남는 공간의 1fr, 둘째 열은 남는 공간의 2fr을 받습니다. `fr`은 전체 폭의 단순 비율이 아니라 고정 크기·간격·콘텐츠 제약을 처리한 뒤의 유연 공간을 나눕니다.

7.2. `minmax()`와 반복으로 카드 목록을 만듭니다

```
.cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(min(100%, 16rem), 1fr));
  gap: 16px;
}
```

컨테이너가 넓으면 열이 늘고, 좁으면 열이 줄어듭니다. `min(100%, 16rem)`은 컨테이너가 16rem보다 좁을 때 최소 트랙이 부모를 강제로 넘지 않게 돕습니다.

7.3. `1fr`도 콘텐츠 최소 크기 때문에 넘칠 수 있습니다

```
.layout {
  grid-template-columns: 240px 1fr;
}
```

둘째 열 안에 공백 없는 긴 식별자가 있으면 `1fr` 트랙의 자동 최소 크기가 콘텐츠를 따라 커질 수 있습니다. 해당 열이 실제로 남는 공간 안으로 줄어야 한다면 다음을 검토합니다.

```
.layout {
  grid-template-columns: 240px minmax(0, 1fr);
}
```

하지만 `minmax(0, 1fr)`만 넣고 콘텐츠를 숨기지 않습니다. 긴 문자열의 줄 바꿈·표의 의미·이미지 축소 가능 여부를 같이 판단합니다.

7.4. 명시적 배치와 자동 배치를 구분합니다

Grid item에 `grid-column`이나 이름 있는 area가 지정되면 명시적으로 배치됩니다. 나머지는 자동 배치 알고리즘이 빈 영역을 찾습니다. 새 카드가 추가될 때 순서가 예상과 달라지면 DOM 순서·명시적 위치·`grid-auto-flow`를 함께 봅니다.

8. Flexbox와 Grid를 목적에 맞게 선택합니다

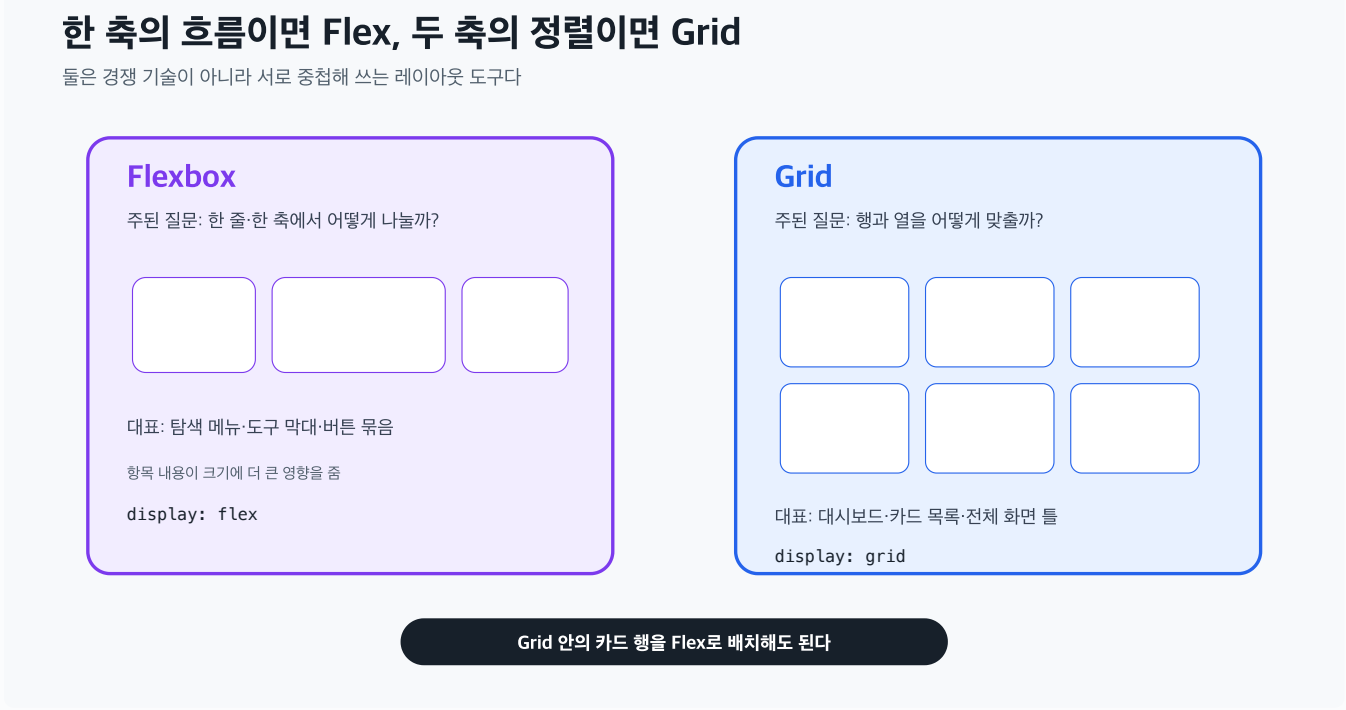


그림 7. 한 축의 내용 흐름과 공간 배분이 중심이면 Flexbox, 행과 열의 정렬이 중심이면 Grid가 출발점입니다.

Flexbox

주된 질문: 한 줄·한 축에서 어떻게 나눌까?

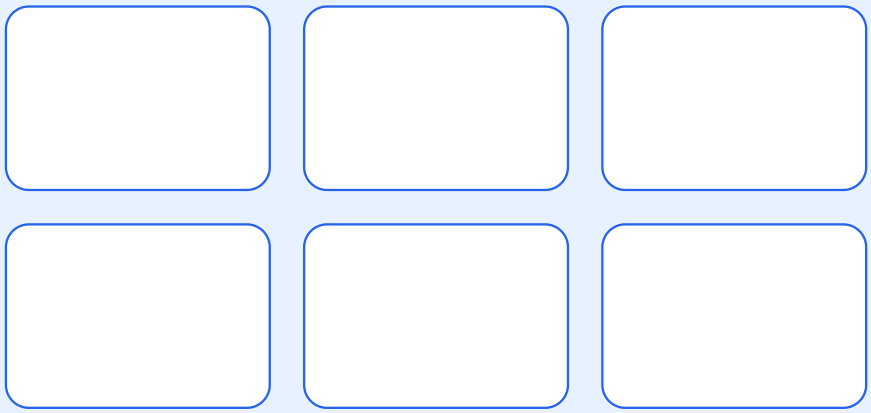
대표: 탐색 메뉴·도구 막대·버튼 묶음

항목 내용이 크기에 더 큰 영향을 줌

```
display: flex
```

Grid

주된 질문: 행과 열을 어떻게 맞출까?



대표: 대시보드·카드 목록·전체 화면 틀

```
display: grid
```

판단 질문	Flexbox 쪽	Grid 쪽
주된 관계	한 행 또는 한 열	행과 열을 함께 맞춤
크기를 이끄는 것	항목 내용과 주축 공간	정의한 트랙과 영역
대표 사례	메뉴·도구 막대·버튼 묶음	대시보드·카드 목록·화면 틀
새 항목	줄을 따라 추가	셀과 자동 배치에 따라 추가

둘은 함께 쓸 수 있습니다. 전체 페이지는 Grid로 두 열을 만들고, 각 카드의 제목과 행동 버튼은 Flexbox로 한 줄에 배치할 수 있습니다.

8.1. 시각 순서 변경을 완료 조건에 넣습니다

`order`나 Grid 좌표로 카드가 시각적으로 재배치되면 다음을 검수합니다.

- DOM 읽기 순서가 의미에 맞는가
- `Tab` 초점 순서가 화면 순서와 심하게 어긋나지 않는가
- 화면 확대와 모바일 재배치에서도 제목과 행동의 관계가 유지되는가
- CSS를 끄거나 로딩 실패해도 내용 순서가 이해되는가

8.2. 레이아웃 도구를 속성 수로 고르지 않습니다

“두 개니까 Flex, 세 개니까 Grid”가 아닙니다. 항목 수가 아니라 관계를 봅니다. 세 버튼도 한 축의 행동 묶음이면 Flexbox가 자연스럽고, 두 영역도 행·열 정렬과 이름 있는 area가 핵심이면 Grid가 적절할 수 있습니다.

9. 콘텐츠의 고유 크기로 스트레스 테스트합니다

그림 8. 평균적인 예시 글보다 긴 문자열·원본 이미지·번역·글자 확대가 레이아웃의 실제 최소 크기를 드러냅니다.

OK

짧은 한국어

정상 줄바꿈

두 줄 안에서
안전하게 흐름

LONG

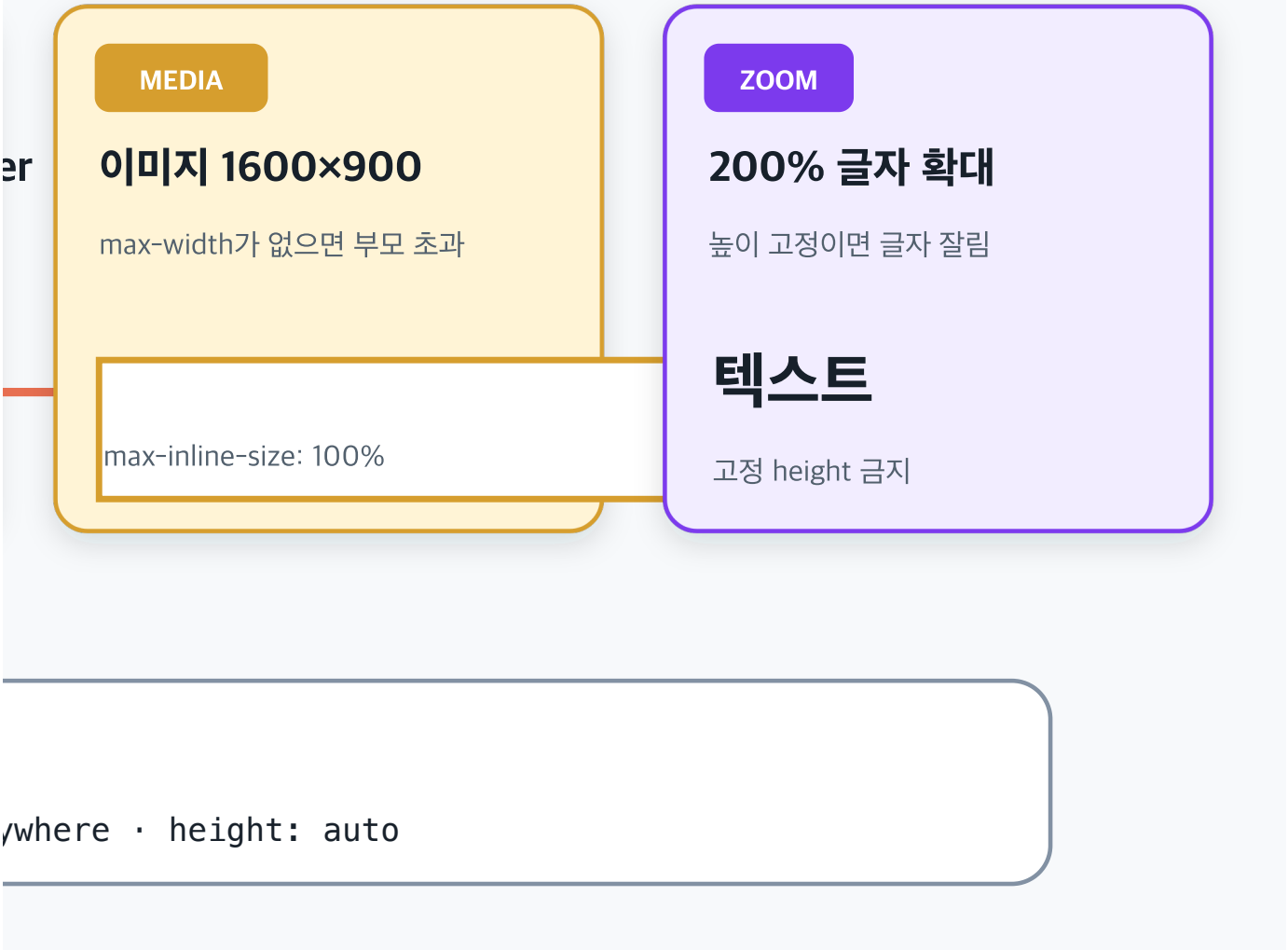
VeryLongUnbrokenIdentifie

공백 없는 긴 문자열

overflow-wrap 확인

견고한 기본값

`min-width: 0 · max-width: 100% · overflow-wrap: any`


er
MEDIA
이미지 1600×900
max-width가 없으면 부모 초과
max-inline-size: 100%
ZOOM
200% 글자 확대
높이 고정이면 글자 잘림
텍스트
고정 height 금지
ywhere · height: auto

9.1. min-content와 max-content를 직관적으로 읽습니다

- **min-content:** 피할 수 있는 넘침 없이 상자가 가질 수 있는 대체로 가장 작은 inline 크기
- **max-content:** 줄 바꿈 기회를 사용하지 않을 때 내용이 원하는 대체로 이상적인 inline 크기

공백 없는 긴 URL·식별자·파일명은 줄 바꿈 기회가 거의 없어 min-content 폭도 커질 수 있습니다.

9.2. 네 가지 시험 콘텐츠

시험	넣을 값	확인할 실패
긴 글	평소 제목의 2~3배	겹침·잘림·버튼 밀림
공백 없는 값	URL·식별자·긴 파일명	가로 넘침
큰 미디어	원본 폭이 큰 이미지·영상	부모 경계 초과
확대·번역	200% 글자·긴 번역	고정 높이 잘림·행동 누락

9.3. 원인별로 수정합니다

```
img, video {
  max-inline-size: 100%;
  block-size: auto;
}

.identifier {
  overflow-wrap: anywhere;
}

.flex-child {
  min-inline-size: 0;
}
```

이미지를 무조건 축소하면 상세 확인이 필요한 도면·지도·표를 읽기 어려울 수 있습니다. 그런 콘텐츠는 의미상 두 방향 스크롤이 필요한 예외인지, 확대·전체 화면·대체 보기 기능이 필요한지 별도로 결정합니다.

9.4. 고정 높이를 의심합니다

`height: 48px`인 버튼이나 카드가 한글 두 줄·200% 글자에서 잘린다면 `min-height`와 내용 기반 높이를 검토합니다. 높이를 늘리는 것만으로 끝내지 말고 다음 행과 겹치지 않는지, 초점 표시가 잘리지 않는지 확인합니다.

BIG IDEA 03

반응형 검수의 가장 값싼 오류 탐지기는 긴 제목·긴 식별자·큰 이미지·글자 확대입니다. 좋은 레이아웃은 내용이 달라져도 관계를 보존합니다.

10. 가로 넘침의 첫 원인을 추적합니다

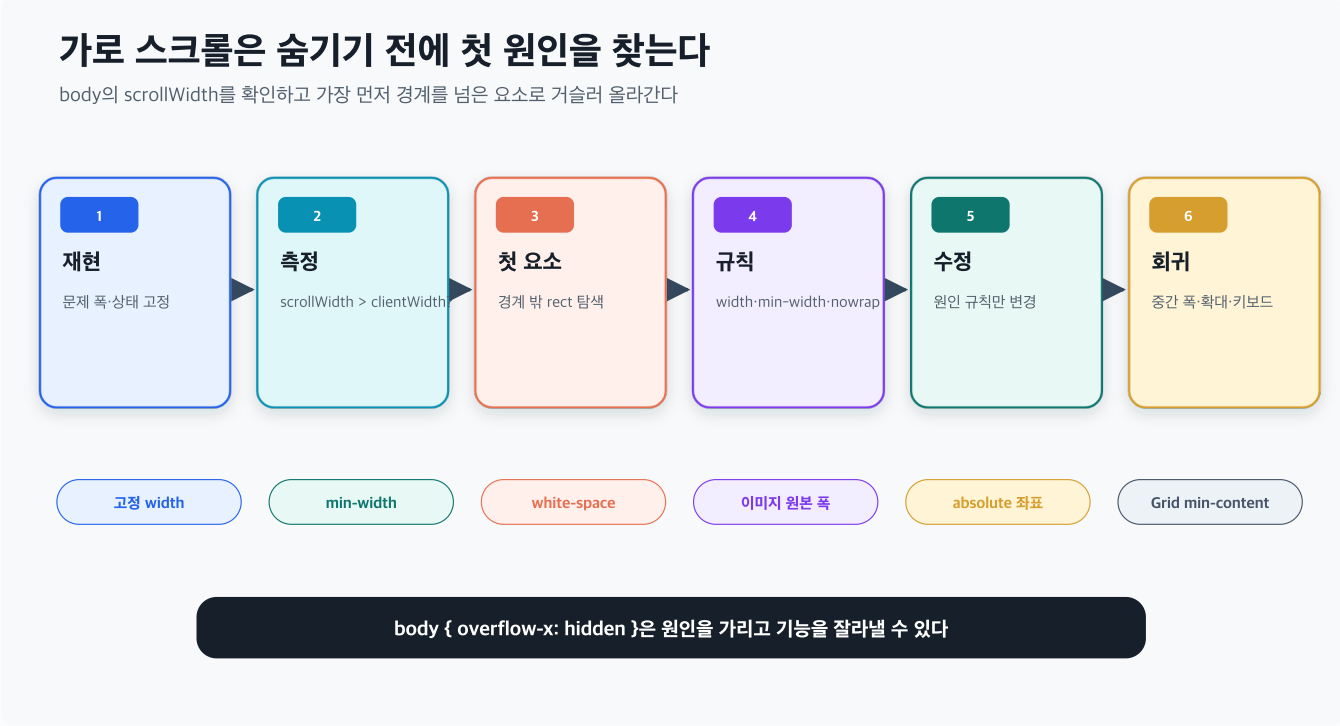
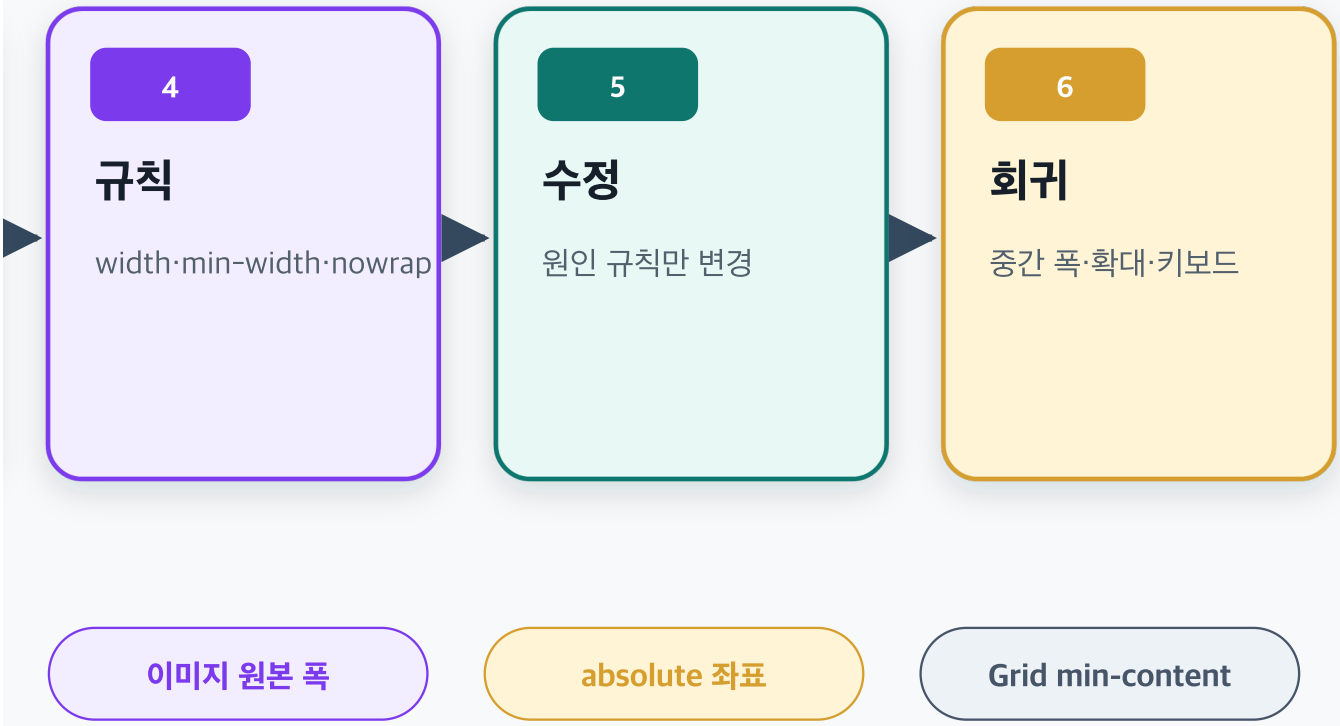


그림 9. 가로 스크롤은 문제 폭을 고정하고 문서 경계를 처음 넘은 요소와 계산 규칙을 찾아 수정합니다.


1
재현
문제 폭·상태 고정
2
측정
scrollWidth > clientWidth
3
첫 요소
경계 밖 rect 탐색
고정 width
min-width
white-space
body { overflow-x: hidden }은 원



원인을 가리고 기능을 잘라낼 수 있다

10.1. `overflow` 값의 차이

값	경계 밖 내용	스크롤 컨테이너
`visible`	바깥에 그릴 수 있음	아님
`hidden`	잘림	프로그래밍 방식 스크롤 가능
`clip`	잘림	스크롤 불가
`scroll`	잘림·항상 스크롤 방식 제공 가능	맞음
`auto`	넘칠 때 스크롤 방식 제공	넘칠 때 맞음

`overflow-x: hidden`으로 body의 가로 스크롤을 없애면 바깥에 있던 버튼·초점 표시·표 열을 잘라낼 수 있습니다. 원인 수정 뒤에도 의도한 클리핑인지 따로 확인합니다.

10.2. 문서 넘침을 수치로 확인합니다

```
const root = document.documentElement;
({
  clientWidth: root.clientWidth,
  scrollWidth: root.scrollWidth,
  overflow: root.scrollWidth - root.clientWidth
});
```

`scrollWidth`가 `clientWidth`보다 크면 가로로 더 넓은 내용이 있습니다. 브라우저의 반올림 때문에 1px 안팎 차이가 생길 수 있으므로 실제 스크롤과 요소 경계를 같이 봅니다.

10.3. 경계를 넘은 후보를 찾습니다

```
const limit = document.documentElement.clientWidth;
[...document.querySelectorAll('body *')]
  .map(el => ({ el, rect: el.getBoundingClientRect() }))
  .filter(({ rect }) => rect.right > limit + 1 || rect.left < -1)
  .map(({ el, rect }) => ({
    element: el,
    left: Math.round(rect.left),
    right: Math.round(rect.right),
    width: Math.round(rect.width)
  }));
```

이 목록은 후보를 좁히는 도구입니다. 화면 밖으로 이동하는 캐러셀·닫힌 드로어처럼 의도한 요소도 잡힐 수 있습니다. 현재 상태와 사용자 행동을 함께 판단합니다.

10.4. 첫 원인 규칙을 찾습니다

후보에서 다음을 차례로 확인합니다.

1. 고정 `width` · `min-width`
2. `white-space: nowrap` ·긴 문자열
3. 이미지·영상의 원본 크기
4. Flex 항목의 자동 최소 크기
5. Grid 트랙의 min-content 제한
6. absolute·transform 좌표
7. 음수 margin과 viewport 단위

부모와 자식 여러 개가 함께 경계를 넘더라도 가장 안쪽의 긴 문자열이 첫 원인일 수 있고, 반대로 자식은 정상인데 부모의 `width: 100vw`와 padding 합계가 원인일 수 있습니다.

10.5. 수정 뒤 회귀 폭을 확인합니다

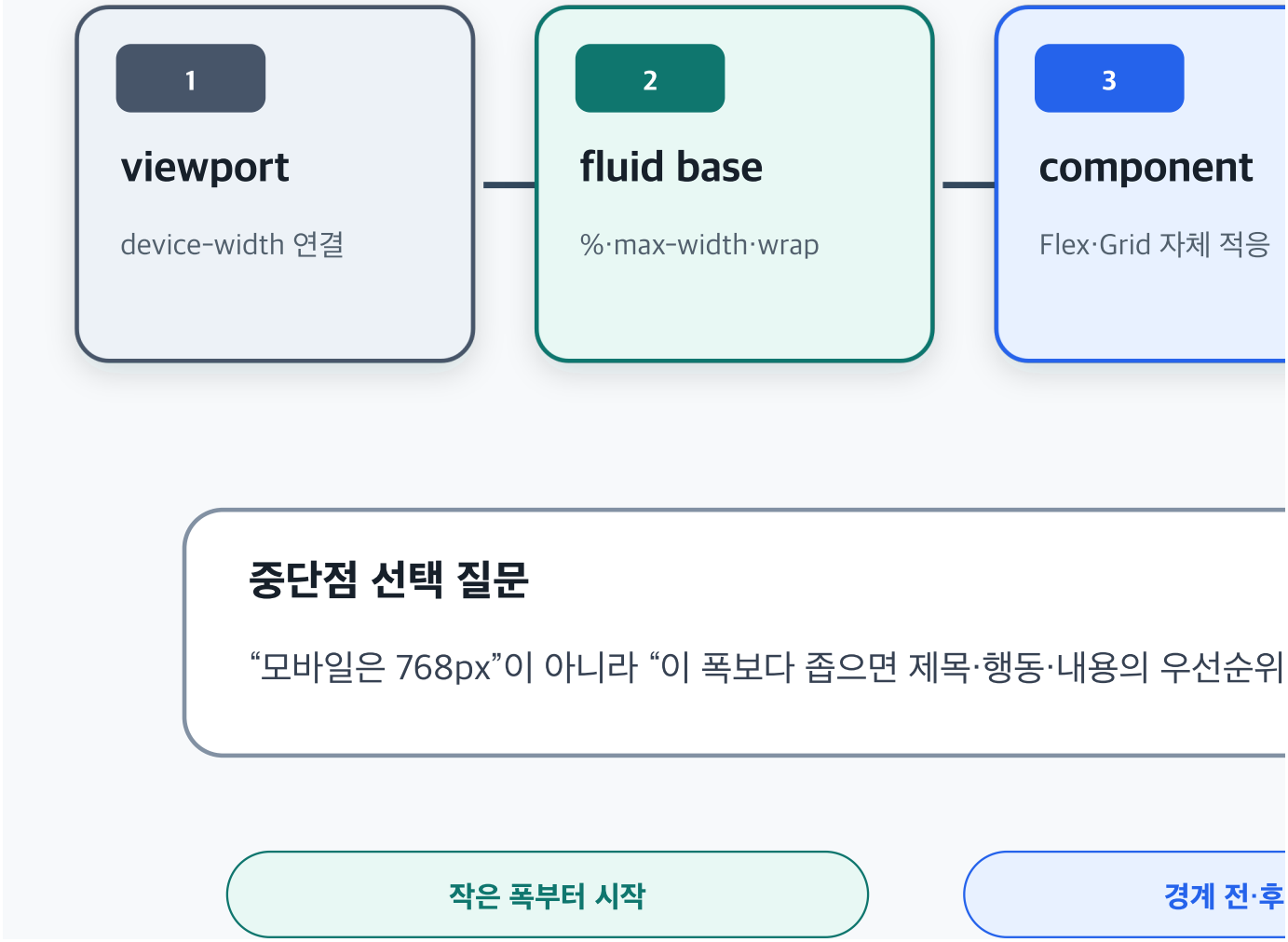
문제 폭 하나만 고치지 않습니다.

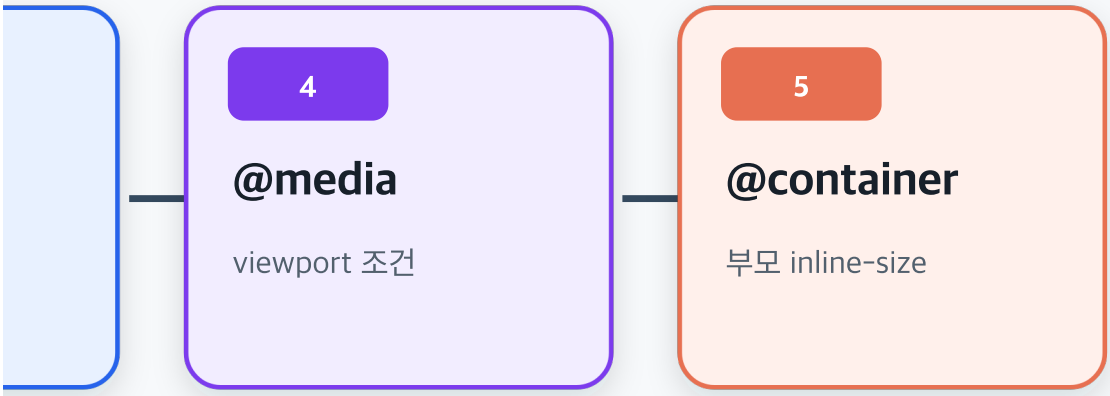
```
문제 768px → 확인 767px · 768px · 769px
추가 확인 → 320 · 390 · 1024 · 1280px
상태 확인 → 기본 · 긴 글 · 빈 결과 · 오류 · 모달 열림
입력 확인 → 마우스 · 키보드 · 200% 글자 확대
```

실행 주의
Console에 입력한 코드는 현재 페이지 문맥에서 실행됩니다. 비밀번호·토큰·개인정보가 있는 업무 화면의 DOM이나 Console 결과를 외부에 공유하지 않습니다. 검수 기록에는 필요한 선택자·치수·익명화된 화면만 남깁니다.

11. 반응형은 사용 가능한 공간의 변화로 판단합니다

그림 10. 반응형은 viewport 연결과 유동 기본값을 먼저 두고, 실제 콘텐츠가 무너지는 조건에 media·container query를 추가합니다.


1
viewport
device-width 연결
2
fluid base
%·max-width·wrap
3
component
Flex·Grid 자체 적응
중단점 선택 질문
“모바일은 768px”이 아니라 “이 폭보다 좁으면 제목·행동·내용의 우선순위
작은 폭부터 시작
경계 전·후



가 무너지는가?”

·중간 확인

기능과 정보 손실 없음

11.1. viewport를 실제 장치 폭과 연결합니다

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

이 선언은 모바일 브라우저가 문서의 viewport를 장치 독립 픽셀 폭에 맞추도록 돕습니다. `maximum-scale=1`이나 `user-scalable=no`로 사용자의 확대를 막지 않습니다.

11.2. 작은 공간의 유동 기본값을 먼저 만듭니다

```
.page {
  width: min(calc(100% - 2rem), 72rem);
  margin-inline: auto;
}

.actions {
  display: flex;
  flex-wrap: wrap;
  gap: 0.75rem;
}
```

기본 규칙이 작은 폭에서도 작동하면 넓은 폭에서 필요한 변화만 추가할 수 있습니다.

11.3. media query는 viewport·사용자 환경을 묻습니다

```
@media (width >= 48rem) {
  .layout {
    grid-template-columns: 16rem minmax(0, 1fr);
  }
}
```

`width` media feature는 viewport 폭을 묻습니다. 새 코드에서는 물리적 장치 크기인 `device-width`로 “모바일 기기”를 추측하지 않습니다.

11.4. container query는 구성요소가 받은 공간을 묻습니다

```
.results-panel {
  container-type: inline-size;
}

@container (inline-size > 42rem) {
  .result-card {
    grid-template-columns: 10rem 1fr auto;
  }
}
```

같은 카드가 넓은 `main`과 좁은 `aside`에 놓일 때 각 부모 공간에 맞춰 달라질 수 있습니다. media query가 문서 환경을 묻는다면 container query는 해당 구성요소의 조상 컨테이너를 묻습니다.

11.5. 중단점은 콘텐츠가 무너지는 지점에서 정합니다

"태블릿은 768px"이라는 기기 이름만으로 정하지 않습니다. 폭을 천천히 줄이며 다음이 처음 실패하는 지점을 찾습니다.

- 제목과 행동 버튼이 겹침
- 두 열의 핵심 내용이 읽기 어려울 만큼 좁아짐
- 메뉴가 한 줄을 강제해 문서를 넘김
- 표의 필수 열이 의미를 잃음
- 초점 표시나 오류 메시지가 잘림

그 지점 앞에서 배치가 바뀌도록 중단점을 두고, 바로 전·경계·바로 뒤를 검수합니다.

11.6. 리플로와 예외를 구분합니다

웹 콘텐츠 접근성 지침(Web Content Accessibility Guidelines, WCAG) 2.2의 Reflow 기준은 가로쓰기의 일반 콘텐츠가 320 CSS px에 해당하는 폭에서 정보·기능 손실이나 두 방향 스크롤 없이 제시되도록 요구합니다. 지도·데이터 표처럼 사용과 의미에 두 방향 배치가 필요한 부분은 예외가 될 수 있습니다.

예외는 페이지 전체에 적용하는 면제표가 아닙니다. 표 바깥의 제목·필터·설명·행동은 다시 흐르게 하고, 표 영역만 명확한 스크롤 컨테이너와 접근 가능한 대체 표현을 검토합니다.

30초 확인

카드 구성요소가 화면 전체 폭은 1200px이지만 280px짜리 사이드 패널 안에 있습니다. 카드 자체의 좁은 배치를 바꾸려면 media query와 container query 중 어느 조건이 더 직접적입니까?

정답 보기

카드가 실제로 받은 부모 공간에 반응하려면 container query가 더 직접적입니다. media query는 viewport가 넓으므로 카드의 좁은 상태를 표현하지 못할 수 있습니다.

12. Chrome DevTools로 반응형 증거를 수집합니다

12.1. Device mode는 실제 기기 전체를 복제하지 않습니다

Device mode는 모바일 viewport·장치 비율·방향·일부 입력과 네트워크 조건을 시뮬레이션하는 도구입니다. 실제 모바일 브라우저의 주소 막대·키보드·운영체제 글꼴·성능·보조 기술까지 완전히 같지는 않습니다.

따라서 순서는 다음과 같습니다.

1. Device mode로 빠르게 여러 폭을 훑음
2. 중단점과 넘침을 Elements·Computed로 분석
3. 핵심 흐름은 실제 대상 기기와 브라우저에서도 확인

12.2. 폭을 연속적으로 움직입니다

미리 등록된 기기 이름만 누르지 않습니다. Responsive 모드에서 폭을 천천히 줄이고 늘리며 다음 순간을 찾습니다.

- 줄 수가 바뀌는 순간
- 열 수가 바뀌는 순간
- 스크롤바가 처음 생기는 순간
- 버튼이 다음 줄로 내려가는 순간
- 고정 헤더와 내용이 겹치는 순간

12.3. Elements의 배지를 사용합니다

`display: grid` 요소에는 `grid` 배지가 표시될 수 있습니다. 배지를 눌러 선·트랙·영역 overlay를 켭니다. Flexbox도 flex 배지와 편집 도구를 통해 축과 정렬을 확인할 수 있습니다.

Styles에서는 적용 규칙과 출처를, Computed에서는 최종 `display`·크기·최소 크기·overflow를 봅니다. Box model에서는 content·padding·border·margin 치수를 확인합니다.

12.4. 한 건의 증거 묶음

항목	예시
환경	Chrome 150·macOS·100% 확대
상태	검색 완료·결과 12건·필터 열림

항목	예시
폭	viewport 390×844 CSS px
선택 요소	`.result-card__content`
계산값	`display:block; min-width:420px; overflow:visible`
실제 치수	부모 358px·요소 420px·문서 넘침 62px
영향	페이지 전체 가로 스크롤·오른쪽 행동 버튼 일부 가림
완료 기준	320px 이상에서 필수 정보·행동 손실과 페이지 이중 스크롤 없음

12.5. 캡처 이름을 재현 정보로 만듭니다

```
M03-02_search-results_390px_long-title_overflow-before.png
M03-02_search-results_390px_long-title_after.png
```

파일명에는 화면·폭·상태·문제·수정 전후를 넣습니다. 민감 정보는 캡처 전에 화면에서 제거하고, 흐리게 처리한 원본을 따로 남기지 않습니다.

13. 실습 · 레이아웃 실험실에서 첫 원인을 찾습니다

준비물

- L03-02 CSS 레이아웃과 반응형 검수
- L03-02 레이아웃 실험실
- Chrome 150 이상
- T03-02 반응형 레이아웃 점검표

실습 흐름 · 55분

1. 실험실을 열고 정상 유동 레이아웃을 320·390·768·1280px에서 관찰합니다.
2. 고정 폭·줄 바꿈 금지·Flex 최소 크기·absolute 겹침 시나리오를 차례로 선택합니다.
3. 폭 슬라이더를 움직여 처음 실패하는 폭을 찾습니다.
4. 진단 패널의 clientWidth·scrollWidth·넘침 요소 수를 기록합니다.

5. 요소를 선택해 display·width·min-width·overflow·position·실제 사각형을 확인합니다.
6. Box·Flex·Grid·넘침 경계 overlay를 켜 부모와 자식의 관계를 봅니다.
7. DevTools에서 같은 요소의 Styles·Computed·Box model을 확인합니다.
8. 반응형 점검표에 수정 요청과 완료 기준을 작성합니다.

완료 산출물

- CSS 레이아웃 지도 1개
- 폭 5개 이상의 반응형 점검 행
- 오류 시나리오 4개의 첫 실패 폭과 첫 원인
- 계산값과 실제 치수가 있는 개선 요청 3건 이상
- 수정 전·후 완료 기준 1세트

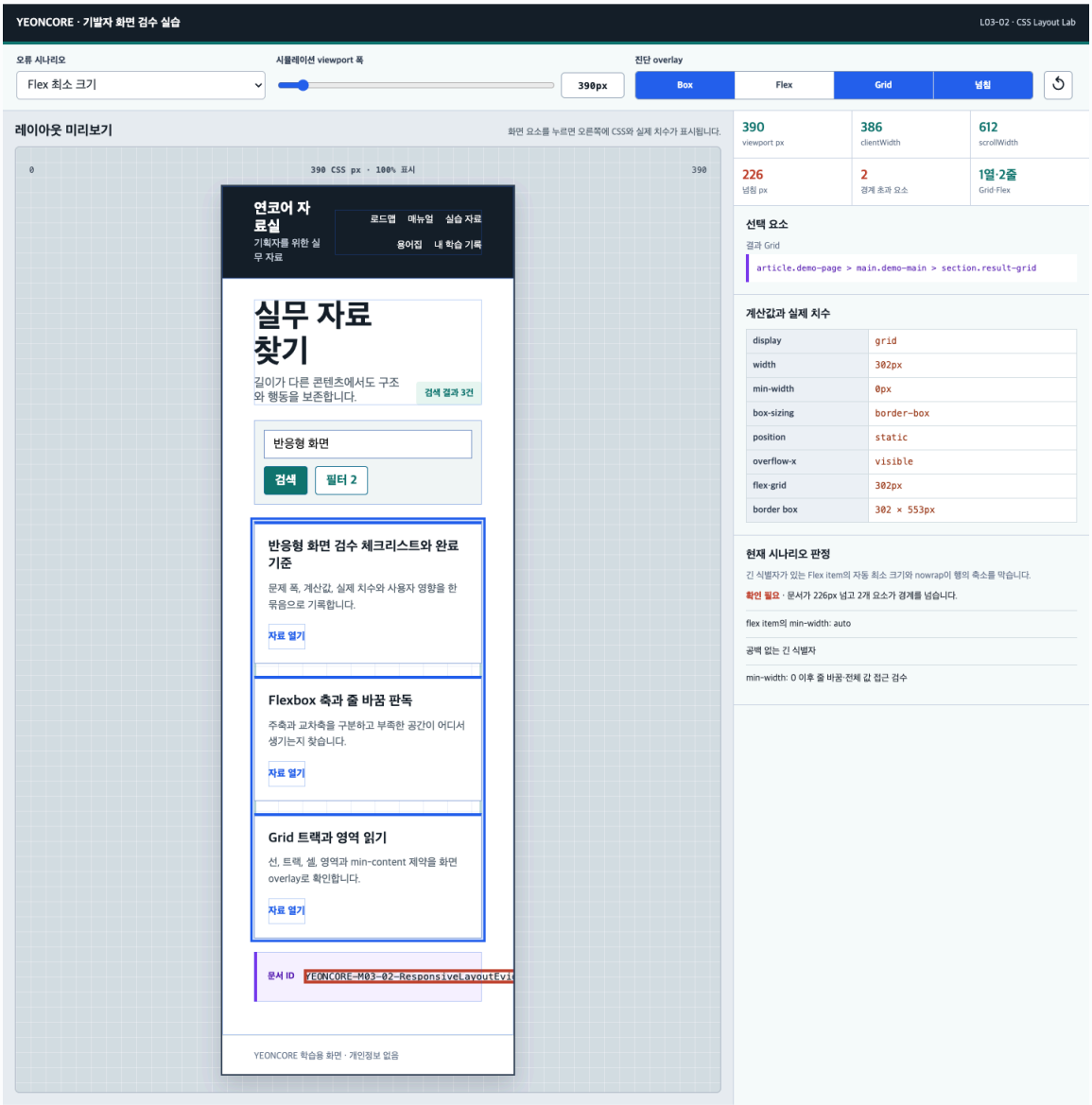


그림 11. 레이아웃 실험실은 같은 콘텐츠를 여러 폭과 오류 규칙으로 바꾸고 실제 넘침 수치를 함께 보여 줍니다.

14. 인쇄용 반응형 레이아웃 학습지

A. 화면과 환경

항목	기록
화면·구성요소	
사용자·중심 작업	
브라우저·운영체제	
확대·글자 설정	
데이터 상태	기본·긴 글·빈 결과·오류·완료

B. 레이아웃 지도

선택 요소	부모	display	크기 기준	최소·최대	흐름·position

C. Flexbox·Grid

컨테이너	종류	주축·열	wrap·트랙	gap	자식 제약
	Flex·Grid				

D. 뷰포트별 검수

폭	열·줄 변화	가로 넘침	정보·행동 손실	첫 원인	판정
320px					
390px					

폭	열·줄 변화	가로 넘침	정보·행동 손실	첫 원인	판정
768px					
1024px					
1280px					

E. 중단점 경계

```
관찰한 변화: _____________________________________________
중단점: __________ px 또는 rem
경계 바로 전: __________   경계: __________   바로 뒤: __________
변화 전 정보 순서: _____________________________________
변화 후 정보 순서: _____________________________________
```

F. 개선 요청

문제 폭·상태	선택 요소	계산값·치수	사용자 영향	수정 제안	완료 기준

15. 셀프 테스트

문제 1 · 박스 모델

`box-sizing: content-box; width: 300px; padding: 20px; border: 2px solid` 인 요소의 테두리 바깥 폭은 몇 px입니까? margin은 제외합니다.

문제 2 · 기준 상자

`width: 70%` 인 `article` 이 최대 960px인 `main` 안에 있습니다. 70%가 항상 viewport 기준이라고 말해도 됩니까?

문제 3 · 일반 흐름

긴 제목이 두 줄이 됐는데 다음 버튼이 밀리지 않고 제목을 덮습니다. 먼저 확인할 레이아웃 단서는 무엇입니까?

문제 4 · Flexbox

`flex-direction: column`에서 `justify-content`와 `align-items`는 각각 어느 축을 정렬합니까?

문제 5 · Flex 최소 크기

Flex item의 긴 파일명이 줄지 않아 전체 페이지가 넘칩니다. `min-width: 0`만 넣고 완료해도 됩니까?

문제 6 · Grid

`grid-template-columns: 240px 1fr`에서 둘째 열의 긴 식별자 때문에 화면이 넘칠 때 검토할 트랙 표현과 콘텐츠 규칙을 쓰세요.

문제 7 · 넘침

`document.documentElement.scrollWidth`가 `clientWidth`보다 64px 큽니다. body에 `overflow-x: hidden`을 넣기 전에 해야 할 일은 무엇입니까?

문제 8 · 반응형 조건

viewport는 1200px이지만 카드가 280px 사이드바 안에 있을 때 카드 배치 전환을 직접 표현하기 좋은 조건은 무엇입니까?

문제 9 · 중단점

768px 하나에서만 정상임을 확인하면 반응형 중단점 검수가 끝납니까? 최소한 어떤 주변 폭을 더 봐야 합니까?

문제 10 · 개선 요청

다음 관찰을 개발자에게 전달할 한 문장으로 바꾸세요.

```
상태: 검색 완료·긴 영문 식별자 포함
viewport: 390px
선택 요소: .result-content
부모 실제 폭: 358px
요소 계산값: min-width: 420px; overflow: visible
문서: clientWidth 390px; scrollWidth 452px
영향: 오른쪽 상세 버튼이 화면 밖으로 밀림
```

16. 정답과 해설

문제 1

344px입니다. content 300px + 좌우 padding 40px + 좌우 border 4px입니다. `content-box`에서 지정 `width`는 content 영역입니다.

문제 2

안 됩니다. 퍼센트 폭은 해당 요소의 containing block을 기준으로 계산됩니다. 보통 `main`의 content 폭과 padding·border·`box-sizing`을 함께 확인합니다.

문제 3

제목·버튼 또는 그 부모가 `position: absolute`처럼 일반 흐름에서 빠졌는지 먼저 확인합니다. 위치 지정 조상·고정 높이·Grid/Flex 배치·stacking context도 이어서 봅니다.

문제 4

`column`의 주축은 일반 가로쓰기에서 세로이므로 `justify-content`가 세로 공간을 정렬합니다. 교차축은 가로이므로 `align-items`가 가로 방향을 정렬합니다.

문제 5

안 됩니다. 해당 item이 실제로 줄어야 하는지 확인하고 긴 파일명이 줄 바뀌거나 전체 보기 기능을 제공하는지 검수해야 합니다. `min-width: 0` 뒤 정보 잘림과 키보드 접근도 확인합니다.

문제 6

둘째 트랙이 남는 공간 안으로 줄어야 한다면 `minmax(0, 1fr)`을 검토합니다. 동시에 긴 식별자에 의미에 맞는 `overflow-wrap`·줄 바꿈·복사 가능한 전체 값 표시 방식을 검토합니다.

문제 7

문제 폭과 상태를 고정하고 경계를 넘은 첫 요소를 찾습니다. 그 요소와 부모의 고정 폭·최소 폭·nowrap·미디어 원본 크기·Flex/Grid 최소 크기·absolute 좌표를 확인한 뒤 원인 규칙을 수정합니다.

문제 8

카드가 실제로 받은 조상 컨테이너의 inline 크기를 묻는 container query가 직접적입니다. media query는 viewport가 넓어 카드의 좁은 배치를 감지하지 못할 수 있습니다.

문제 9

끝나지 않습니다. 최소한 767·768·769px을 확인해 경계의 겹침과 빈 구간을 봅니다. 대표 폭 320·390·1024·1280px과 긴 글·확대 상태도 함께 검수합니다.

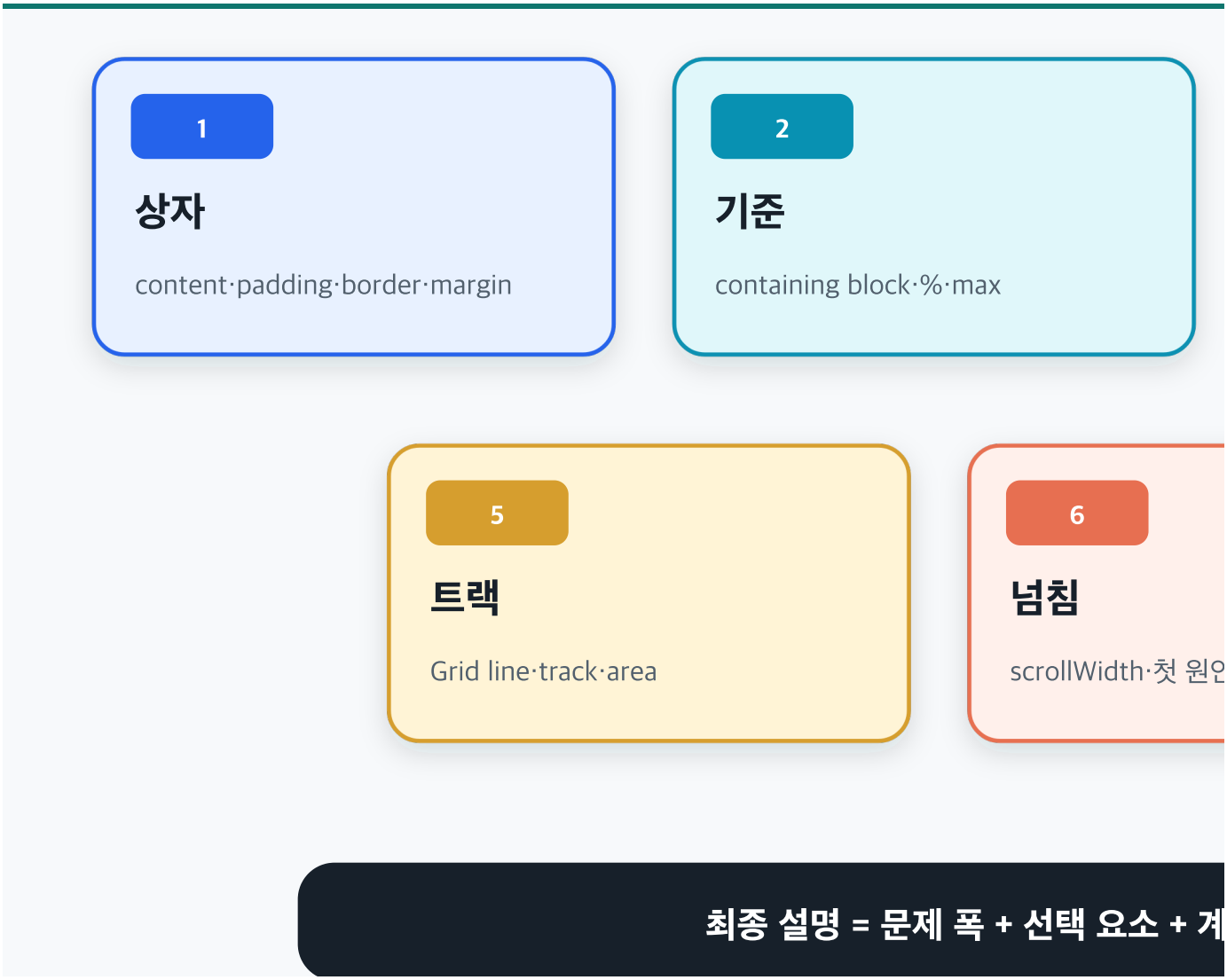
문제 10

예시 답안:

```
검색 완료 상태의 390px viewport에서 .result-content의 min-width 420px이
부모 테두리 상자 358px보다 커 문서 scrollWidth가 clientWidth보다 62px 큽니다.
그 결과 오른쪽 상세 버튼이 화면 밖으로 밀립니다. 콘텐츠 영역이 줄어들고 긴
식별자가 정보 손실 없이 줄 바뀌도록 수정한 뒤 320·389·390·391·768px과
200% 글자 확대에서 페이지 가로 스크롤 및 행동 가림이 없음을 완료 기준으로 합니다.
```

17. 한 장 요약

그림 12. 상자·기준·흐름·축·트랙·넘침·변화의 일곱 질문으로 CSS 레이아웃과 반응형 검수를 복습합니다.


1
상자
content·padding·border·margin
2
기준
containing block·%·max
5
트랙
Grid line·track·area
6
넘침
scrollWidth·첫 원인
최종 설명 = 문제 폭 + 선택 요소 + 계

3

흐름

in-flow·position·DOM 순서

4

축

Flex main·cross·wrap

7

변화

media·container·reflow

산값 + 실제 치수 + 영향 + 완료 기준

단계	질문	산출물
1	실제 상자는 얼마나 큰가	content·padding·border·margin 치수
2	어느 상자가 크기의 기준인가	containing block·%·최소·최대 지도
3	일반 흐름 안에서 앞뒤를 밀어내는가	DOM·시각·초점 순서와 position
4	Flexbox의 주축·교차축은 어디인가	basis·grow·shrink·wrap 기록
5	Grid의 행·열 트랙은 어떻게 계산되는가	선·트랙·영역·minmax 기록
6	경계를 처음 넘은 요소와 규칙은 무엇인가	scrollWidth·사각형·첫 원인
7	어느 공간 조건에서 배치가 바뀌는가	media·container query·경계 검수

처음에는 CSS 속성을 모두 외우지 않아도 됩니다. **문제 폭 → 선택 요소 → 계산값 → 실제 치수 → 사용자 영향 → 완료 기준**을 남기면 필요한 규칙과 질문이 따라옵니다.

18. 다음 학습과 출처

18.1. 다음 매뉴얼

M03-03「JavaScript 상태·이벤트·비동기 읽기」에서는 HTML 구조와 CSS 배치가 사용자 행동에 따라 어떻게 바뀌는지 확인합니다. 이벤트·상태·로딩·성공·빈 결과·오류를 화면 동작 설명서로 연결합니다.

18.2. 함께 사용할 자료

- L03-02 CSS 레이아웃과 반응형 검수
- L03-02 레이아웃 실험실
- T03-02 반응형 레이아웃 점검표
- CSS 레이아웃·반응형 용어집

18.3. 출처와 확인일

아래 공식 자료는 모두 2026. 7. 15.에 확인했습니다.

- W3C CSS Box Model Module Level 4: content·padding·border·margin 영역

- W3C CSS Box Sizing Module Level 3: `width`·최소·최대·min-content·max-content
- W3C CSS Display Module Level 3: 상자 생성·일반 흐름·Flex·Grid formatting context
- W3C CSS Flexible Box Layout Module Level 1: 주축·교차축·wrap·grow·shrink·basis
- W3C CSS Grid Layout Module Level 2: 선·트랙·셀·영역·트랙 크기·자동 배치
- W3C CSS Overflow Module Level 3: `visible` · `hidden` · `clip` · `scroll` · `auto`
- W3C Media Queries Level 5: viewport `width`·범위 조건· `device-width` 사용 제한
- W3C CSS Containment Module Level 3: size container와 `@container`
- W3C WAI WCAG 2.2 · Reflow: 320 CSS px 리플로와 두 방향 배치 예외
- Chrome for Developers · Device mode: 모바일 viewport 시뮬레이션
- Chrome for Developers · Inspect CSS grid layouts: Grid 배지·overlay·트랙·영역 확인
- Chrome for Developers · CSS features reference: Styles·Computed·Flexbox·Grid 편집과 overlay
- web.dev · Responsive web design basics: viewport 설정·유동 레이아웃·반응형 기본 원칙