11. CSS Grid Layout

Grid 컨테이너 및 아이템 정의

CSS Grid Layout은 **2차원(행과 열)** 기반의 강력한 레이아웃 시스템이다.
Flexbox가 한 방향(1차원) 정렬에 특화되었다면, Grid는 **행과 열 모두를 동시에 제어**할 수 있는 레이아웃 방식이다.
이를 위해 먼저 **Grid 컨테이너와 Grid 아이템**의 정의를 명확히 이해하는 것이 중요하다.

---

1. Grid 컨테이너 (Grid Container)

Grid 레이아웃을 사용하는 부모 요소로,
이 컨테이너 안에서 Grid가 정의되고 아이템들이 배치된다.

생성 방법

```
.container {
  display: grid;         /* 블록 레벨 그리드 */
  display: inline-grid;  /* 인라인 레벨 그리드 */
}
```

Grid 컨테이너가 되면 가능한 기능:

- 행과 열의 **수와 크기 정의**
- 그리드 라인, 셀, 영역 설정
- 격자(grid) 기반의 **정밀한 배치**

예시:

```
.container {
  display: grid;
  grid-template-columns: 200px 1fr 100px;
  grid-template-rows: auto auto;
}
```

→ 3열 2행의 Grid 레이아웃 생성

---

2. Grid 아이템 (Grid Item)

Grid 컨테이너의 직계 자식 요소들이 Grid 아이템이다.
각 아이템은 **셀(cell)에 배치되며**, 필요에 따라 특정 셀을 **병합(가로/세로로)** 하거나
지정된 위치에 정확히 배치할 수도 있다.

자동 배치 (암시적 배치)

```
<div class="container">
  <div>Item 1</div> <!-- grid item -->
  <div>Item 2</div> <!-- grid item -->
</div>
```

명시적 배치

```
.item1 {
  grid-column: 1 / 3; /* 1열부터 3열 전까지 → 2칸 병합 */
  grid-row: 1;        /* 1행에 위치 */
}
```

---

3. 구조적 용어

용어	설명
Grid Container	`display: grid`가 적용된 부모 요소
Grid Item	Grid Container의 **직계 자식 요소**
Grid Line	열과 행의 **경계선**, 시작과 끝의 기준이 됨
Grid Track	하나의 **행(row)** 또는 **열(column)**
Grid Cell	하나의 **격자칸**, 행과 열이 만나는 지점
Grid Area	여러 개의 셀을 병합하여 만든 **영역**

---

4. 시각 예시

2행 3열 그리드

```
+-----------+-----------+-----------+
|   item1   |   item2   |   item3   | ← 1행
+-----------+-----------+-----------+
|   item4   |   item5   |   item6   | ← 2행
+-----------+-----------+-----------+
 ↑    ↑           ↑
열1  열2         열3
```

`item1`을 2칸 병합:

```
.item1 {
  grid-column: 1 / 3;  /* 1열부터 3열 이전까지 → 열 2칸 병합 */
}
```

---

5. 요약 표

요소	정의	설정 방법
Grid 컨테이너	Grid 레이아웃을 설정한 부모 요소	`display: grid`
Grid 아이템	컨테이너의 직계 자식 요소	자동으로 설정됨
배치 위치 지정	`grid-column`, `grid-row`, `grid-area`	각 아이템에서 개별 설정

결론

> Grid 시스템에서는 컨테이너가 **격자의 크기와 구조를 정의**하고,
> 아이템은 그 구조 안에서 **자동 또는 수동으로 배치**된다.
> 이 구조적 구분이 명확하기 때문에 Grid는 **복잡하고 정밀한 2차원 레이아웃**에 매우 적합하다.

행/열 설정: `grid-template-rows`, `grid-template-columns`

CSS Grid 레이아웃에서 **격자(Grid)**를 정의하는 핵심은
바로 **행(row)**과 **열(column)**의 크기를 정하는 것이다.
이를 위해 사용하는 속성이 `grid-template-rows`와 `grid-template-columns`이다.

---

1. `grid-template-columns`: 열(Column) 정의

- Grid 컨테이너 내부에 **몇 개의 열을 만들지**, 그리고 **각 열의 너비를 얼마로 할지** 지정하는 속성이다.

기본 문법

```
.container {
  grid-template-columns: <크기> <크기> ...;
}
```

예시

```
.container {
  display: grid;
  grid-template-columns: 100px 200px 1fr;
}
```

- 첫 번째 열: 고정 100px
- 두 번째 열: 고정 200px
- 세 번째 열: 남는 공간을 모두 차지 (`1fr`)

---

2. `grid-template-rows`: 행(Row) 정의

- 각 **행의 높이를 얼마나 할 것인지** 설정하는 속성이다.

기본 문법

```
.container {
  grid-template-rows: <크기> <크기> ...;
}
```

예시

```
.container {
  display: grid;
  grid-template-rows: auto 100px 1fr;
}
```

- 첫 번째 행: 내용 높이에 맞춤 (`auto`)
- 두 번째 행: 고정 100px
- 세 번째 행: 남는 공간을 모두 차지

---

3. 사용 가능한 단위

단위	의미
`px`, `em`, `%`	절대 또는 상대 단위
`fr`	Fractional unit – 남은 공간의 비율
`auto`	콘텐츠에 맞춤
`min-content`, `max-content`	내용에 따라 최소/최대 크기
`minmax(a, b)`	`a` 이상, `b` 이하 크기 범위 지정

예시

```
grid-template-columns: 1fr 2fr; /* 전체 공간 중 1:2 비율로 나눔 */
grid-template-rows: minmax(100px, auto);
```

---

4. 반복(`repeat`) 함수

동일한 크기의 행/열을 반복적으로 만들고 싶을 때 `repeat()` 함수를 사용한다.

```
grid-template-columns: repeat(3, 1fr);
```

→ `1fr 1fr 1fr`과 같음

또 다른 예:

```
grid-template-rows: repeat(2, 150px) auto;
```

→ `150px 150px auto`

---

5. 혼합 예제

```
.container {
  display: grid;
  grid-template-columns: 150px 1fr 2fr;
  grid-template-rows: 50px auto 1fr;
}
```

→ 3열, 3행의 Grid 레이아웃 구성

열	크기
1열	150px
2열	남는 공간 중 1
3열	남는 공간 중 2

행	크기
1행	50px
2행	내용에 맞게 자동
3행	나머지 공간 차지

---

6. `auto-fill` / `auto-fit` 사용 예

컨테이너 너비에 따라 반복 개수가 자동으로 결정되도록 할 수 있다.

```
grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
```

- **auto-fill**: 가능한 칸 수만큼 생성
- **minmax**: 칸의 최소 너비는 150px, 최대는 남는 공간 비례

→ 반응형 레이아웃 구현에 유용

결론

> `grid-template-rows`와 `grid-template-columns`는 Grid 레이아웃의 **골격을 구성하는 핵심 속성**이다.
> 이를 통해 개발자는 정확한 **열 수, 행 수, 비율, 크기 단위**를 지정하며
> 매우 정밀하고 유연한 2차원 UI 구조를 만들 수 있다.

`gap`, `row-gap`, `column-gap`

CSS Grid와 Flexbox에서 아이템 간의 **간격(Spacing)**을 설정할 때 사용하는 속성이
바로 `gap`, `row-gap`, `column-gap`이다.
이전에는 `grid-row-gap`, `grid-column-gap`이라는 이름을 사용했지만, 현재는 축약된 형태가 권장된다.

---

1. `gap`: 행과 열의 간격을 한꺼번에 설정

```
.container {
  display: grid;
  gap: 20px;  /* 행과 열 모두 20px 간격 */
}
```

- `gap: <행 간격> <열 간격>` 형식도 가능
  → 첫 번째 값은 **row-gap**, 두 번째 값은 **column-gap**

```
gap: 10px 30px;  /* 행 간격 10px, 열 간격 30px */
```

---

2. `row-gap`: 행 간격만 지정

- Grid 행(row) 사이의 수직 간격을 설정

```
.container {
  row-gap: 15px;
}
```

---

3. `column-gap`: 열 간격만 지정

- Grid 열(column) 사이의 수평 간격을 설정

```
.container {
  column-gap: 40px;
}
```

---

4. 예제 코드

```css
.container {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: auto auto;
  gap: 20px 40px;  /* row-gap: 20px, column-gap: 40px */
}
```

```html
<div class="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
</div>
```

→ 위 예제에서는 3x2 그리드가 생성되며, 각 행 사이에는 20px, 각 열 사이에는 40px 간격이 생긴다.

---

5. 단위

- `px`, `em`, `%`, `rem`, `vw`, `vh` 등 다양한 단위 사용 가능
- `fr` 단위는 `gap`에는 사용할 수 없음 (fr은 공간 배분용이기 때문)

---

6. `gap` vs `margin`

비교 항목	`gap`	`margin`
역할	**아이템 간의 간격**	요소와 요소 외부 간격
위치	**아이템 사이**	특정 요소에만 적용
적용 대상	Grid, Flex (일부 지원)	모든 박스 모델 요소
계산법	Grid 구조 내에서만 사용됨	요소의 레이아웃에 직접 영향

---

7. Flexbox에서도 사용 가능

```css
.flex-container {
  display: flex;
  gap: 16px;
}
```

→ 최신 브라우저에서는 Flexbox에서도 `gap` 사용 가능 (과거에는 Grid 전용이었음)

결론

> `gap`, `row-gap`, `column-gap`은 **여백을 margin으로 조절하던 번거로움을 해소**하며,
> Grid나 Flex 레이아웃에서 **간단하고 직관적인 간격 조절**을 가능하게 해준다.
> 특히 반응형 디자인, 복잡한 레이아웃을 구현할 때 **구조를 깨지 않고 여백만 조정할 수 있다는 점**이 큰 장점이다.

`grid-area`, `grid-row`, `grid-column`

CSS Grid Layout에서는 각 아이템이 **그리드 셀 내부에서 어디에 위치할지, 얼마나 넓게 병합할지**를 지정할 수 있다.
이때 사용되는 핵심 속성이 바로 `grid-area`, `grid-row`, `grid-column`이다.

---

1. 기본 개념 요약

속성	역할
`grid-row`	아이템이 차지할 **행 영역** 지정
`grid-column`	아이템이 차지할 **열 영역** 지정
`grid-area`	위 둘을 합친 축약형 또는 이름 정의에 사용

---

2. `grid-row`, `grid-column`

기본 문법

```
.item {
  grid-row: <start> / <end>;
  grid-column: <start> / <end>;
}
```

- `<start>`: 시작 라인 번호 (1부터 시작)
- `<end>`: 끝 라인 번호 (미포함) 또는 `span n` 형식 사용 가능

예제

```
.item1 {
  grid-row: 1 / 3;         /* 1행부터 2행까지 사용 (2칸 병합) */
  grid-column: 2 / 4;      /* 2열부터 3열까지 사용 (2칸 병합) */
}

.item2 {
  grid-row: span 2;        /* 아래쪽으로 2행 병합 */
  grid-column: 1 / 2;      /* 1열만 사용 */
}
```

시각 예시 (행/열 라인 기준)

```
행 라인:   1       2       3
          +-------+-------+
          | item1 | item1 |
     1    +-------+-------+
          | item1 | item1 |
     2    +-------+-------+

열 라인:   1       2       3       4
```

---

3. `grid-area`

용법 1: `grid-row-start / grid-column-start / grid-row-end / grid-column-end`를 한 줄에 표현하는 축약형

```
.item {
  grid-area: <row-start> / <column-start> / <row-end> / <column-end>;
}
```

예:

```
.item {
  grid-area: 1 / 2 / 3 / 4;
}
```

→ `grid-row: 1 / 3`, `grid-column: 2 / 4`과 같음
→ **1행 2열 위치에서 시작해서 2행과 2열 차지**

용법 2: `grid-template-areas`와 함께 이름 기반 영역 정의

```
.container {
  display: grid;
  grid-template-areas:
    "header header"
    "main sidebar"
    "footer footer";
}

.header   { grid-area: header; }
.main     { grid-area: main; }
.sidebar  { grid-area: sidebar; }
.footer   { grid-area: footer; }
```

→ **시멘틱한 영역 배치**를 가능하게 함

---

4. `span` 키워드

- 특정 라인까지가 아니라 **몇 칸을 병합할지**를 지정

```
.item {
  grid-row: span 2;        /* 2행 병합 */
  grid-column: span 3;     /* 3열 병합 */
}
```

---

5. 요약 비교 표

속성	설명	예시
`grid-row`	시작/끝 라인을 기준으로 행 배치	`1 / 3` 또는 `span 2`
`grid-column`	시작/끝 라인을 기준으로 열 배치	`2 / 4` 또는 `span 2`
`grid-area`	4개의 값을 한 줄로 표현	`1 / 2 / 3 / 4`
`grid-area` (이름형)	`grid-template-areas`와 연동	`grid-area: header;`

---

6. 실전 예제

```
.container {
  display: grid;
  grid-template-columns: 100px 1fr 1fr;
  grid-template-rows: 100px 200px;
  gap: 10px;
}

.item {
  grid-area: 1 / 1 / 3 / 3; /* 1행 1열부터 시작해서 2행 2열까지 차지 (2x2 병합) */
}
```

결론

> `grid-row`, `grid-column`, `grid-area`는 Grid Layout에서 **아이템을 정밀하게 배치**하고,
> **셀 병합이나 특정 위치 지정**을 손쉽게 할 수 있게 해주는 **핵심 속성들**이다.
> 특히 `grid-area`는 축약형 또는 이름 기반으로 사용 가능하여
> **유지보수성과 가독성** 모두 향상시켜준다.

`place-items`, `place-content`

CSS Grid와 Flexbox에서는 **아이템 또는 콘텐츠의 정렬 위치**를 설정하기 위해
수직 및 수평 정렬 속성을 각각 지정한다.
이를 간결하게 축약해서 표현하는 속성이 `place-items`와 `place-content`다.

---

1. `place-items`

- 개별 아이템의 **정렬 방식(Align + Justify)**을 한 줄에 지정
- `align-items` + `justify-items`의 축약형

문법

```
place-items: <align-items> <justify-items>;
```

- 첫 번째 값: 수직 정렬 (`align-items`)
- 두 번째 값: 수평 정렬 (`justify-items`)
- 하나만 쓰면 두 값에 모두 적용됨

예시

```
.container {
  display: grid;
  place-items: center;             /* align-items: center; justify-items: center; */
}

.container {
  display: grid;
  place-items: start end;          /* align-items: start; justify-items: end; */
}
```

사용 가능한 값

- `start`, `end`, `center`, `stretch` 등

값	의미
`start`	컨테이너의 시작 지점
`end`	컨테이너의 끝 지점
`center`	가운데 정렬
`stretch`	아이템 크기를 컨테이너에 맞게 늘림 (기본값)

---

2. `place-content`

- Grid에서 **전체 콘텐츠 블록 자체의 정렬**을 지정
- `align-content` + `justify-content`의 축약형
- 다수의 아이템들이 **행/열로 묶였을 때 그 전체 묶음을 어디에 둘지** 정함

문법

```
place-content: <align-content> <justify-content>;
```

- 첫 번째 값: 수직 방향(행 정렬)
- 두 번째 값: 수평 방향(열 정렬)

예시

```
.container {
  display: grid;
  place-content: center;           /* align-content: center; justify-content: center; */
}

.container {
  display: grid;
  place-content: space-between center;
}
```

사용 가능한 값

- `start`, `end`, `center`, `space-between`, `space-around`, `space-evenly`, `stretch`

---

3. `place-items` vs `place-content`

속성	역할	적용 대상	구성 속성
`place-items`	개별 아이템 정렬	Grid 아이템 하나하나	`align-items`, `justify-items`
`place-content`	콘텐츠 묶음 정렬	전체 그리드 콘텐츠 블록	`align-content`, `justify-content`

---

4. 예제 비교

```
.container {
  display: grid;
  height: 300px;
  width: 300px;
  grid-template-columns: 100px 100px;
  grid-template-rows: 100px 100px;
  place-items: center center;      /* 각 아이템 정렬 */
  place-content: center center;    /* 전체 콘텐츠 묶음 정렬 */
}
```

결론

> `place-items`와 `place-content`는 **정렬 속성의 축약형**으로
> 코드를 간결하고 가독성 있게 만들 수 있으며,
> Grid/Flex 레이아웃 정렬을 더 직관적으로 제어할 수 있다.

`repeat()`, `minmax()`, `auto-fill`, `auto-fit`

CSS Grid에서 복잡한 행/열 구조를 간결하고 유연하게 정의하려면,
`repeat()`, `minmax()`, `auto-fill`, `auto-fit` 등의 함수를 사용하는 것이 매우 중요하다.
이들은 특히 **반응형 그리드 레이아웃** 구현에서 핵심적인 역할을 한다.

---

1. `repeat()`: 반복된 행 또는 열 정의

문법

```
grid-template-columns: repeat(count, value);
```

- `count`: 반복할 횟수
- `value`: 반복할 값

예시

```
grid-template-columns: repeat(3, 1fr);
/* → 1fr 1fr 1fr */
```

중첩 가능

```
repeat(2, 100px 1fr)
/* → 100px 1fr 100px 1fr */
```

---

2. `minmax()`: 최소-최대 범위 설정

문법

```
grid-template-columns: minmax(min, max);
```

- `min`: 최소 크기
- `max`: 최대 크기
- 콘텐츠 크기, 뷰포트 크기 등 변화에 따라 자동으로 조절됨

예시

```
grid-template-columns: minmax(150px, 1fr);
```

→ 최소 150px, 남는 공간이 있으면 `1fr` 까지 확장

---

3. `auto-fill` vs `auto-fit`: 유동적인 반복

둘 다 `repeat()` 함수와 함께 사용되며,
컨테이너의 너비에 따라 자동으로 반복 횟수를 결정하는 역할을 한다.

공통 문법

```
grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
```

- **컨테이너 크기만큼** 열을 자동으로 추가
- 각 열의 최소 크기는 150px, 최대는 남는 공간 비례(`1fr`)

차이점: `auto-fill` vs `auto-fit`

항목	auto-fill	auto-fit
빈 칸	**자리를 유지**함	**자리를 접어서 제거**함
반응형 변화	고정된 열 수를 확보	열 개수를 줄여 재배치
효과	**그리드 공간 유지**	**아이템 압축 정렬**

시각적 예

- 4칸 자리가 있는데 3개 아이템만 있을 경우:

```
repeat(auto-fill, minmax(150px, 1fr))
→ [O][O][O][ ]
```

```
repeat(auto-fit, minmax(150px, 1fr))
→ [O][O][O]
```

요약 예제

```
.container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 20px;
}
```

→ 아이템 수에 따라 열 개수가 유동적으로 조절되며,
최소 200px을 확보하고 남는 공간이 있으면 균등 배분

---

5. 함께 조합하는 패턴

```
grid-template-columns: repeat(auto-fill, minmax(250px, 1fr));
```

- 넓은 화면에서는 여러 열로 배치됨
- 좁은 화면에서는 자동으로 열 개수가 줄어듦 → **반응형 UI 구현에 최적**

결론

함수	역할
`repeat()`	반복되는 열/행 간결하게 작성
`minmax()`	최소~최대 범위 지정으로 유연한 크기 설정
`auto-fill`	가능한 만큼 열을 자동 생성 (빈 공간 유지)
`auto-fit`	가능한 만큼 열을 자동 생성 (빈 공간 제거)

> 이 네 가지 함수는 CSS Grid의 **자동 배치** 및 **반응형 레이아웃**을 구현할 때 필수 도구다.
> `repeat(auto-fit, minmax())` 조합은 대부분의 그리드 UI에서 가장 널리 사용된다.