

“**알아서 잘 짠
딱 맞는 꿈하고
센스있게
정리하는
flex&grid**

: flex가 grid 어렵드나?

김민찬 강혜진 김상준 김진 김혜원 김도희 김태희 김세훈 김보람 김유진 김예지 김희진
류재준 박누리 박소현 박유진 신현수 여소희 윤수영 이현섭 이호준 이수빈 임홍렬 임희래 임다현
장효순 조미진 조윤희 전유진 지다혜 최수빈 최원범 차경림 한재현 허지현 홍재섭

“앞에서 잘
“딱 잘”
“센”
flex&grid
: flex가 grid 어렵드나?



Contents

Channel. 01 들어가며

- 머리말
- 저자목록
- 홈페이지, Notion, PDF 파일 및 QR 코드

Channel. 02 Flex 핵심개념

- 1. Flex
 - 1.1. 들어가기에 앞서
 - 1.2. Flex란?
 - 1.3. 당신이 Flex를 알면 만들 수 있는 것들 : Flex 속성을 통해 만들 수 있는 레이아웃 소개
 - 1.4. Flex의 특징
 - 1.5. axis와 cross-axis
 - 1.5.1. axis란?
 - 1.5.2. 주축과 교차축
 - 1.5.3. dir 특성
 - 1.5.4. writing mode에 따라 바뀌는 축
 - 1.6. 기본 속성 정보
 - 1.6.1. Flex 알아보기
 - 1.6.2. Flex 사용법
 - 1.6.3. Flex 기본 속성
- 2. Flex-direction
 - 2.1. flex-direction : row
 - 2.2. flex-direction : row-reverse
 - 2.3. flex-direction : column
 - 2.4. flex-direction : column-reverse
 - 2.5. 접근성 고려사항

3. Justify-content

3.1. flex-start

3.2. flex-end

3.3. center

3.4. space-between, space-around, space-evenly

3.4.1. space-between

3.4.2. space-around

3.4.3. space-evenly

3.5. 자주 사용하지 않는 속성들

3.5.1. justify-content : normal

3.5.2. justify-content : initial

3.5.3. justify-content : inherit

3.5.4. justify-content : revert

3.5.5. justify-content : unset

3.5.6. firefox에서만 지원하는 속성

4. Align-items, Align-content

4.1. align-items

4.2. align-content

5. Flex-wrap

5.1. flex-wrap : nowrap

5.2. flex-wrap : wrap

5.3. flex-wrap : wrap-reverse

6. Flex-basis

6.1. flex-basis란?

6.2. flex-basis에 들어가는 값

6.3. flex-basis의 유연성

6.4. flex-basis와 width/height의 관계

6.5. flex-basis : auto; 와 flex-basis : 0;

6.6. flex-basis : content;

6.7. flex-basis보다는 flex 축약 속성을! (W3C)

6.8. flex-basis 호환성

7. Flex-grow

7.1. flex-grow란?

7.2. flex-grow 기능 : 확장 또는 고정

7.3. flex-grow 사용 시 주의할 점

7.4. flex 축약 속성

7.5. flex-shrink보다는 flex 축약 속성을! (W3C)

7.6. flex-grow 호환성

8. Flex-shrink

8.1. Flex-shrink 란?

8.2. Flex-shrink 기능: 축소 또는 고정

8.3. Flex-shrink 사용 시 주의할 점

8.4. Flex 축약 속성

8.5. Flex-shrink보다는 Flex 축약 속성을! (W3C)

8.6. Flex-shrink 호환성 (caniuse)

9. 그 밖의 Flex-item에 사용하는 속성들

9.1. align-self

9.2. order

9.3. z-index

10. IE 지원을 위한 Flex

10.1. -ms- prefix 사용 예시

10.2. IE에서의 Flex 이슈

11. 성배 레이아웃 그리기

11.1. Flex 속성을 이용하여 기본 성배 레이아웃 구현하기

11.2. Flex 속성을 이용하여 변형된 성배 레이아웃 구현하기

Channel. 03 Grid 핵심개념

1. Grid

1.1 grid로 할 수 있는 것들

1.2. 그리드 용어정리

1.3. 개발자 도구로 그리드 체크하는법

1.4. grid 사용법

2. Grid row , Grid column

2.1. grid-template-rows, grid-template-columns

2.2. repeat / 1fr이란

2.3. min-max 함수

2.4. grid-auto-rows , grid-auto-columns

3. Gap

3.1. gap이란?

3.2. margin과 gap의 차이점

3.3. gap의 종류

4. 각 셀의 영역 지정

- 4.1. grid-column-start & grid-column-end
- 4.2. grid-row-start & grid-row-end
- 4.3. 셀의 영역 지정
- 4.4. grid-column & grid-row
- 4.5. span 을 이용한 레이아웃 지정
- 4.6. grid-area
- 4.7. grid-template-areas
- 4.8. grid line 외의 추가적인 방법

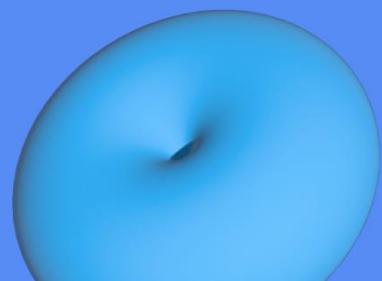
5. IE지원을 위한 Grid

01 들어가며

머리말

저자목록

홈페이지, Notion, PDF 파일 및 QR 코드





머리말

해당 책은 22년 진행한 flex & grid 오픈소스 프로젝트 그룹의 결과물입니다.

사용자에게 컨텐츠를 잘 전달하는 일은 좋은 컨텐츠를 만드는 것 만큼이나 중요한 일입니다. 이를 위해 CSS 레이아웃의 형태는 지속적으로 발전해왔고, 현재는 모던 레이아웃이라고 불리는 flex와 grid가 나옴으로써 복잡한 웹페이지의 레이아웃을 만드는 것도 매우 간단해졌습니다.

하지만 CSS를 처음 배우는 초심자에게 flex와 grid의 문턱이 높아보이는 건 사실입니다. 아직은 최신 기술에 속하다보니 막상 배우려고 하면 기초부터 잘 정리된 문서나 사이트를 찾기란 쉽지 않죠.

이 책에서는 flex와 grid를 처음 배우는 사람의 입장에서 잘 이해할 수 있도록 핵심 개념들을 정의하고 최대한 많은 예제를 통해 이해를 도왔습니다. 또 속성마다 달라지는 변화를 직접 눈으로 확인하고 실무에서 Cheat-Sheet처럼 사용할 수 있도록 **오픈소스 홈페이지도 제작**하였으니, 꼭 홈페이지와 함께 책을 보시기를 권해드립니다.

책이 나오기까지 힘써주신 위니브 임직원과 멋쟁이사자처럼 프론트엔드스쿨 운영진에게, 프로젝트를 끝까지 함께해주신 구성원에게 감사 인사를 드립니다.



저자목록

그
르
브
으
즈
大
亨

ㄱ

강혜진

김민찬

김상돈

김진

김혜원

김도희

김태희 - 혼자가 아니라 동료들과 함께 머리를 맞대고 도왔기에 책을 성공적으로 집필할 수 있었습니
다! 또한 독자들에게 지식을 공유할 수 있는 소중한 기회를 주신 호준님과 동료들에게 감사드립니다! ❤️

김세훈

김보람

김유진

김예지

김희진

■

류재준 - Flex, Grid 내용을 집필하며 저 자신과 동료들이 함께 성장할 수 있는 시간이 되었습니다. 저희들이 경험한 만큼 이 책이 독자 여러분에게도 성장 디딤돌이 될 수 있으면 참으로 좋겠습니다. 그리고 이러한 비옥한 땅을 다져줄 수 있는 기반을 마련해준 호준님에게도 다시한번 감사의 말씀을...!!

▣

박누리

박소현

박유진

신현수

○

여소희

윤수영

이현섭

이호준

이수빈

임홍렬

임희래

임다현

✖

장효순

조미진

조윤희

전유진

지다혜

大

최수빈

최원범

차경림

ホ

한재현

허지현 - 애정과 열정으로 작업한 책이 출간되어 감격스럽습니다. 지도해주신 호준님 존경하고 감사합니다.

홍제섭 - 독자 여러분, 집필 동료 여러분, 호준 대표님 모두 감사합니다. 배움의 자세를 잊지 않는 개발자가 되겠습니다 😊



홈페이지, Notion, PDF 파일 및 QR 코드

해당 책은 리디북스, 교보문고 등에서 무료로 다운로드 받으실 수 있습니다. 아래 링크에서 Notion으로 접속하셔서 보실 수도 있으니 참고 바랍니다. (단축 URL의 경우 서비스의 상태에 따라 접속이 안 될 수도 있습니다.)

PDF는 인쇄해서 교재로 사용 가능합니다. 별도의 허락을 구하시지 않으셔도 괜찮습니다.

- 홈페이지 :

flexnggrid

```
container { display: grid; grid-template-columns: repeat(10, 1fr); grid-template-rows: repeat(4, 1fr); row-gap: 16px; } .item1 { grid-area: 1/3/2/8; } .item2 { grid-area: 1/9/2/11; background-color: #FFF0E6; } .item3 { grid-area: 2/1/3/5; } .item4 { grid-area: 3/1/4/5; }
```

 <https://flexnggrid.com/>

- Notion 링크 : <https://paullabworkspace.notion.site/flex-grid-e5bdea43ba4f427990ff8666ea53ec56>
- QR 코드 :



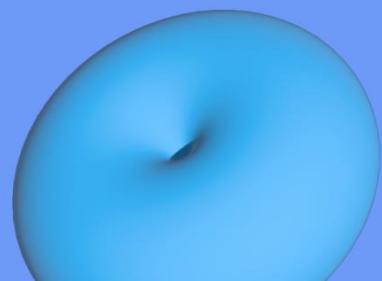
- PDF :

 알잘딱깔센 flex&grid.pdf 36827.3KB

02

Flex 핵심개념

1. Flex
2. Flex-direction
3. Justify-content
4. Align-items, Align-content
5. Flex-wrap
6. Flex-basis
7. Flex-grow
8. Flex-shrink
9. 그 밖의 Flex-item에 사용하는 속성들
10. IE 지원을 위한 Flex
11. 성배 레이아웃 그리기



1. Flex

1.1. 들어가기에 앞서

기기가 다양해짐에 따라 화면 크기도 다양하게 변화되었다. 다양한 기기에 대응하기 위해 웹 서비스를 유연하게 만드는 것은 선택이 아닌 필수가 되었다.

웹 서비스를 만들기 전 서비스의 구조, 대응 기기, 레이아웃, 디자인 등을 계획하여 구현 우선순위를 정하게 된다. 여기서 페이지 레이아웃을 지정하는 방법은 CSS 기술이 발전함에 따라 많은 변화가 있었다. 다양한 화면의 크기에 맞추어 유연하게 대응할 수 있는 레이아웃을 만들기 위해 새롭게 추가된 속성이 Flex와 Grid이다. **Flex는 1차원 적인 레이아웃을** 잡을 때 사용하며, **grid는 2차원 적인 레이아웃을** 잡을 때 용이하다. 쉽게 설명하자면 다음과 같다.

Flex는 부모 요소(컨테이너) 아래에서 자식 요소(아이템)들이 한 방향으로 배치가 된다. 따라서 **행 또 는 열** 한 가지 방향으로 레이아웃을 배치해야 하는 경우에 사용한다. 또한 다양한 속성을 통해 자식 요소 사이를 일정한 간격으로 줄 수 있고 정렬하기가 매우 편하다.

Grid는 **행과 열** 두 가지 방향으로 레이아웃을 배치해야 하는 경우 사용한다. 원하는 요소를 어디서부터 어디까지, 어느 방향으로 차지하게 할 건지 정하고 배치하면 된다.

지금부터 '알아서 잘 딱 깔끔하고 센스 있게' Flex를 배워보자.

1.2. Flex란?

Flexbox는 모던한 웹사이트를 위하여 제안된 CSS3의 새로운 레이아웃 방식이다. 공식적으로 "CSS Flexible Box Layout Module"이라고 정의된 Flexbox 레이아웃은 요소의 사이즈가 명확하지 않거나 동적으로 변화할 때, 방향은 물론 순서 등을 개선하기 위해 만들어진 CSS3의 새로운 레이아웃 방식이다.

Flexbox 레이아웃을 통해 요소의 위치를 더 간단하게 만들고 더 적은 코드로 복잡한 레이아웃을 만들 수 있기에 개발 과정을 편하게 도와주며 사용하기가 더 쉽다.

Flexbox 레이아웃은 블록이나 인라인 레벨 요소와는 달리 '방향'을 중심으로 구성한다.

1.3. 당신이 Flex를 알면 만들 수 있는 것들 : Flex 속성을 통해 만들 수 있는 레이아웃 소개

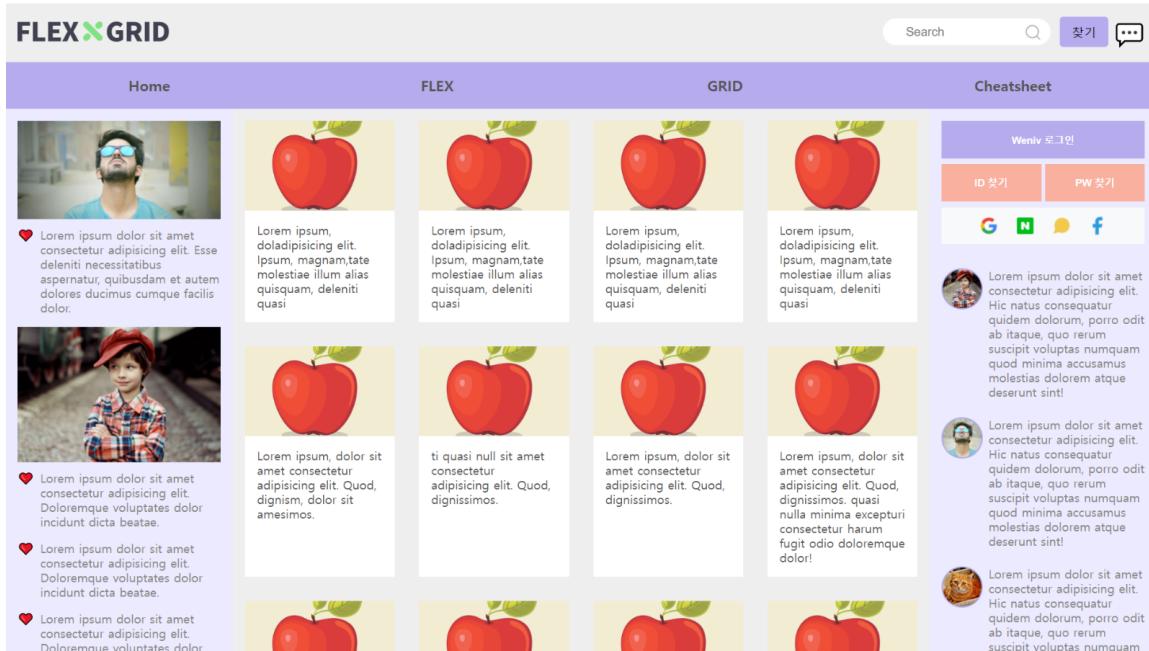
Flex는 레이아웃 배치에 용이한 속성이다. 아래의 그림과 같이 전체적인 카드를 순서대로 배치할 수 있고, 또한 각 카드 내부에서도 Flex를 이용해서 요소를 보다 쉽게 배치할 수 있다.

Card



flex로 구현된 UI - 1

Flex를 활용한 홈페이지이다. 전체적인 레이아웃을 Flex로 배치하고, 아이템들 또한 Flex를 사용한 것을 확인할 수 있다. 이처럼 Flex를 사용한다면, 기존에 float를 이용한 방법보다 더욱 쉽고 간편하게 요소들은 배치할 수 있다.



flex로 구현된 UI - 2

1.4. Flex의 특징

Flex는 ul과 li처럼 부모와 자식 태그가 필요하고, 부모 요소(`flex-container`)와 자식 요소(`flex-item`)에 적용하는 속성이 구분되어 있다는 특징을 갖는다.

`flex-container`에는 정렬 방식과 item의 배치 흐름을 정의하고, `flex-item`에는 크기, 속성, 순서를 정의 한다.

`flex-container`의 속성으로는 `flex-direction`, `flex-wrap`, `justify-content`, `align-items`, `align-content`이 있고, `flex-item`은 `flex`, `flex-grow`, `flex-shrink`, `flex-basis`, `align-self`, `order`, `z-index` 등의 속성이 있다.

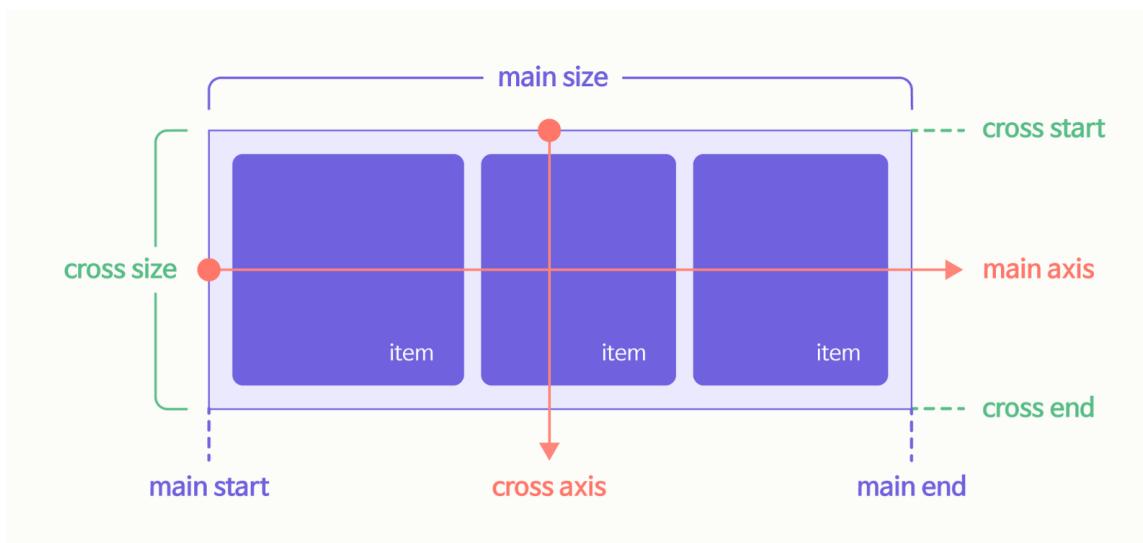
축이라는 개념이 있어서 이 축을 기준으로 정렬을 할 수도 있고, 흐름도 제어할 수 있다. 주축(main axis)에 따라 정렬되며, 방향은 `flex-direction` 속성으로 결정한다.

1.5. axis와 cross-axis

1.5.1. axis란?

Flex는 부모 요소의 속성을 통해 자식들의 방향을 제어할 수 있다. 이 속성을 통해 자식들을 가로(행 방향)로 배열할 수도 있고, 세로(열 방향)로 배열할 수도 있고 심지어 반대 방향으로도 배치가 가능하다.

Axis는 축을 의미하는데, 축이란 어떠한 것이 움직이는 데에 방향의 기준이 되는 것을 뜻한다. 지구의 자전축이 보이지 않듯, Flex에는 보이지 않지만 두 가지 축이 존재한다. Flex의 이 두 가지 축은 자식들이 어느 방향으로 정렬이 될지 정해주는 기준이 된다.



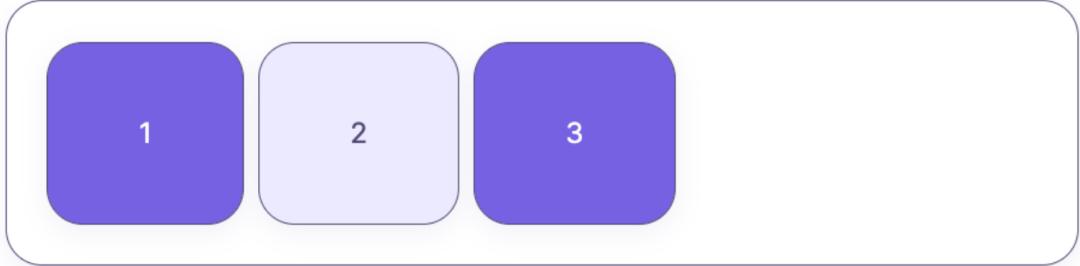
1.5.2. 주축과 교차축

- Main Axis

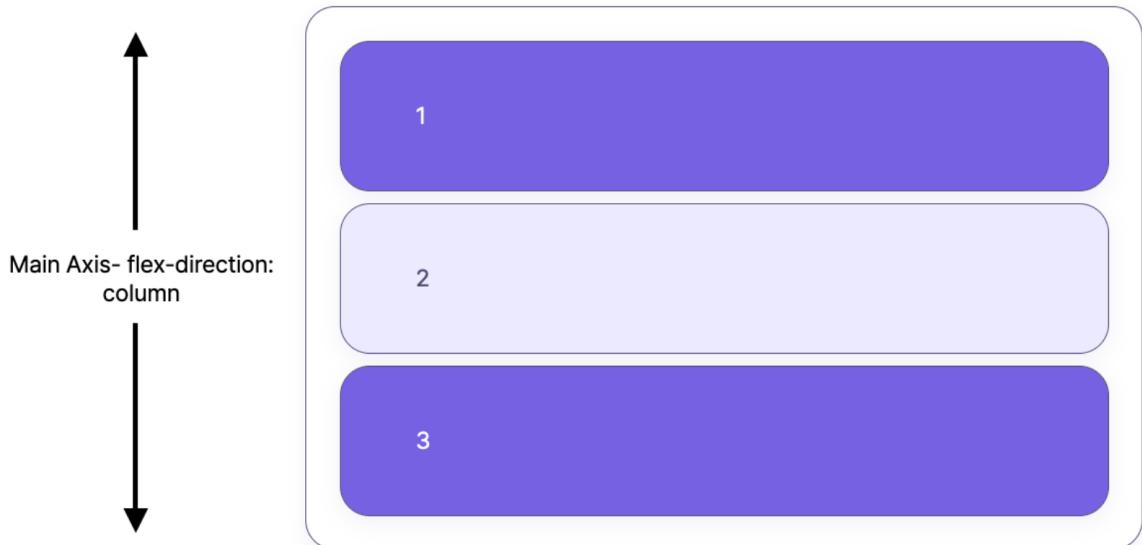
Flex의 주축이 되는 main axis의 방향은 `flex-direction`이라는 속성에 의해 결정된다. 이는 다음과 같이 네 가지의 경우가 있다. 자세한 설명과 코드는 flex-direction 챕터에서 다루게된다.

- `flex-direction: row` : `flex-direction`의 기본값으로 아이템들이 행 방향, 가로로 배치된다.
- `flex-direction: row-reverse` : 아이템들이 역순으로 가로로 배치된다.
- `flex-direction: column` : 아이템들이 열 방향으로, 세로로 배치된다.
- `flex-direction: column-reverse` : 아이템들이 역순으로 세로로 배치된다.

Main Axis- flex-direction: row



flex-direction:row



flex-direction:column

- **Cross Axis**

Cross axis는 Main axis 방향의 수직 방향이다. Main axis가 `flex-direction`에 의해 결정되었다면, Cross axis는 main axis 방향에 따라 결정이 된다.

1.5.3. dir 특성

 1.5.3 챕터와 1.5.4 챕터는 건너 뛰셨다가 필요할 때 찾아보기를 권고해드립니다.

html 요소 텍스트의 방향성을 나타내는 열거된 속성이다. 기본값은 `auto` 이다.

```
<div dir = "ltr / rtl / auto">
...
</div>
```

- `ltr` : 글자를 읽는 방법이 왼쪽에서 오른쪽으로 쓰이는 언어이다.
- `rtl` : 글자를 읽는 방법이 오른쪽에서 왼쪽으로 읽는 아랍권 국가에서 사용된다.
- `auto` : 기본 알고리즘을 사용하여 강한 방향성을 가진 문자를 찾을 때까지 요소 내부의 문자를 구문 분석하고 전체 요소에 해당 방향성을 적용한다.

dir 속성에 따른 flex-Axis의 변화

`flex-direction:row` 와 `flex-direction:row-reverse` 는 컨테이너의 방향성에 따라 영향을 받기 때문에 `dir` 전역 속성을 확인해야 한다.

ltr

- `flex-direction:row` : flex-Axis는 왼쪽의 flex-start에서 오른쪽 flex-end이다.
- `flex-direction:row-reverse` : flex-Axis는 오른쪽 flex-start에서 왼쪽 flex-end이다.
- container에 `dir="ltr"` 입력

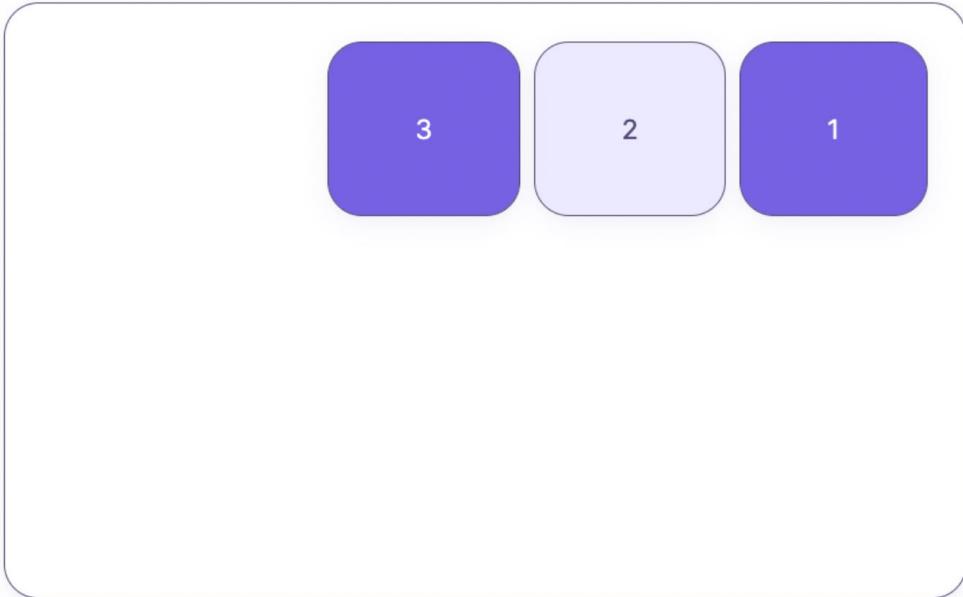
rtl

- `flex-direction:row` : flex-Axis는 오른쪽의 flex-start에서 왼쪽 flex-end이다.
- `flex-direction:row-reverse` : flex-Axis는 왼쪽 flex-start에서 오른쪽 flex-end이다.
- container에 `dir="rtl"` 입력

```
<body>
  <div dir="rtl" class="container"> //container가 rtl로 변환된다.
    <div class="item">1</div>
    <div class="item even">2</div>
    <div class="item">3</div>
  </div>
</body>
```

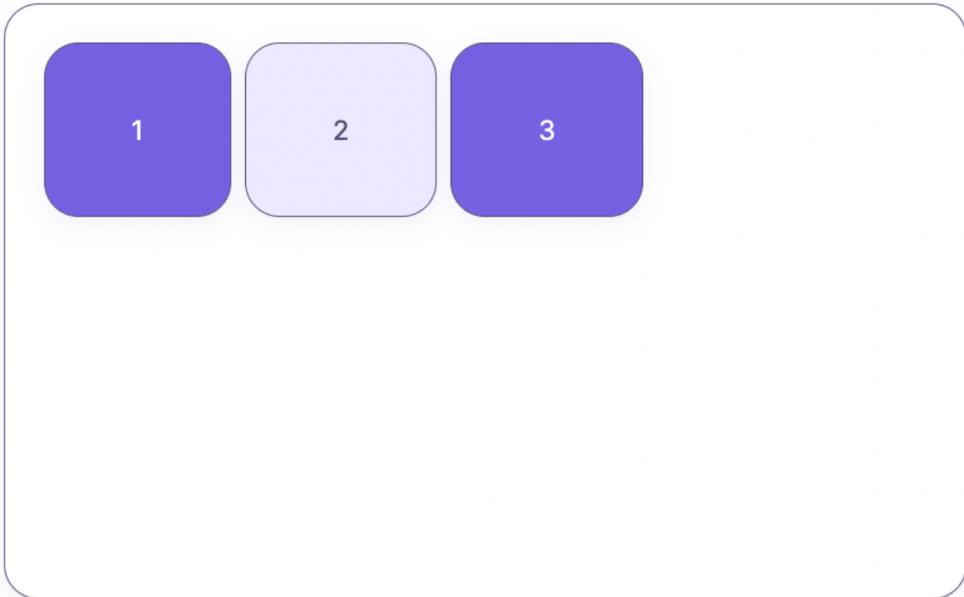
Flex_1 Flex

- `flex-direction:row`



```
.container {  
  display: flex;  
  flex-direction: row;  
  align-items: flex-start;  
}
```

- `flex-direction:row-reverse`



```
.container {
  display: flex;
  flex-direction: row-reverse;
  align-items: flex-start;
}
```

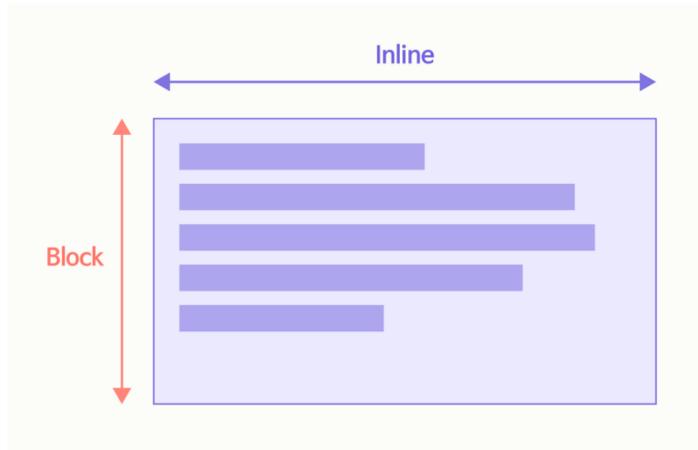
1.5.4. writing mode에 따라 바뀌는 축

writing mode(읽기 모드)란?

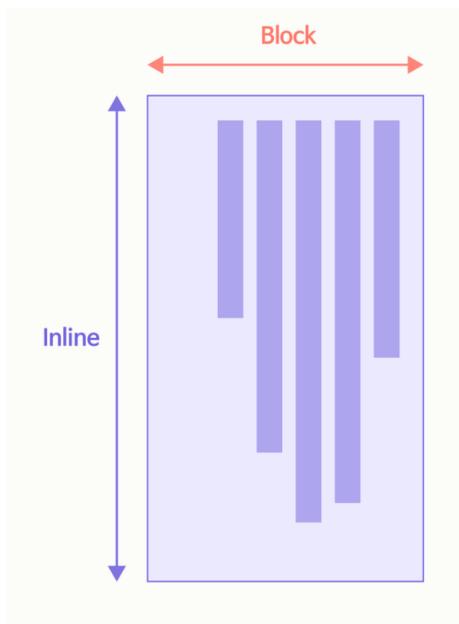
문서를 구성하는 텍스트 흐름 방향을 `writing mode`라고 한다.

writing mode의 중요성

최근 `flex` 박스와 `grid` 템플릿을 사용한 레이아웃이 많아지고 있다. 이때 우리는 자연스럽게 인라인 방향(텍스트 방향)을 왼쪽에서 오른쪽, 블록 방향(단락 방향)은 위에서 아래라고 생각한다. 표준 방향 `row`도, 인라인 방향이 가로인 것도 이상하다고 느끼지 않는다. 하지만 국가마다 설정된 `writing mode`가 다를 수 있다. 만약 우리가 만든 홈페이지를 전 세계에서 보게 된다면, `writing mode`를 고려해야 하는 상황이 생길 수도 있다. 아직은 영어권 국가에서 사용하는 `horizontal-tb` 방향 이외의 `writing mode`를 많이 사용하지 않지만, `flex`와 `grid` 시대를 맞이하며 `writing mode`에 대한 이해를 바탕으로 논리적 사고를 확장하는 것은 중요한 일이다.



가로 방향으로 글을 작성할 때 일반적으로 적용되는 블록(단락)과 인라인(텍스트) 방향이다.



세로 방향으로 글을 작성할 때 적용되는 블록(단락)과 인라인(텍스트) 방향이다.

writing mode 종류

1) horizontal-tb

텍스트를 왼쪽에서 시작하여 오른쪽으로 작성된다. 영어나 아랍어에서 주로 사용된다.

2) vertical-rl

텍스트가 오른쪽에서 왼쪽으로 세로로 작성된다. 한국, 일본, 중국 등의 아시아권 국가에서 세로 방향으로 글을 작성할 때에만 적용된다.

3) vertical-lr

텍스트가 왼쪽에서 오른쪽으로 세로로 작성된다. 대표적으로 몽골어가 있다.

4) sideways-lr, side-rl

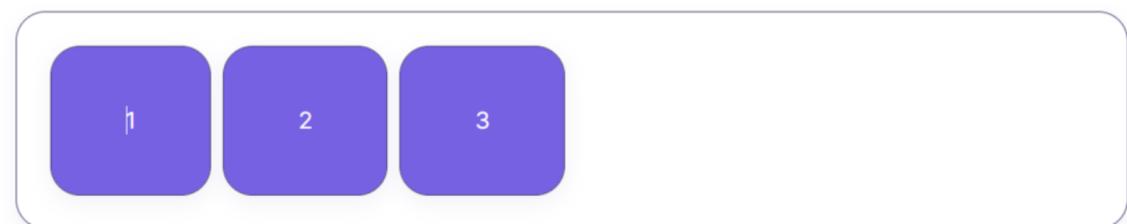
horizontal-tb 속성처럼 텍스트가 가로로 작성된다. 하지만 lr 와 rl 방향 변화는 파이어폭스에서만 확인 가능하다.

writing mode에 따라 바뀌는 axis 방향

1) flex-direction: row

horizontal-tb

인라인과 블록 방향이 모두 가로이며, Main Axis 는 가로, Cross Axis 세로이다.

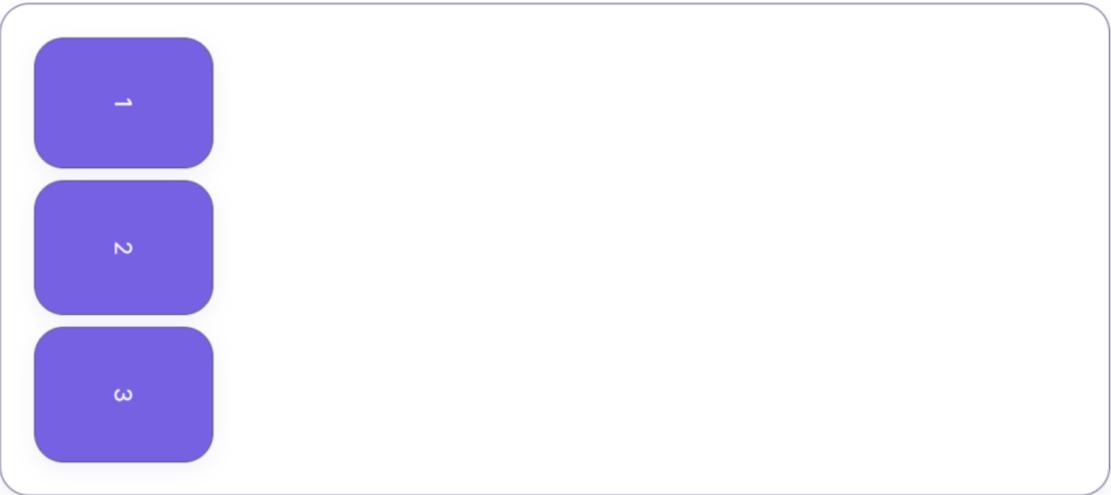


```
.container {
    writing-mode: horizontal-tb;
}
```

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

`vertical-lr`

인라인 방향과 블록 방향이 모두 세로이며 **Main Axis** 는 세로, **Cross Axis** 는 가로이다. 글씨가 눕혀지는 이유는 writing mode가 전환되면서 인라인과 블록 방향이 모두 세로로 바뀌었기 때문이다.



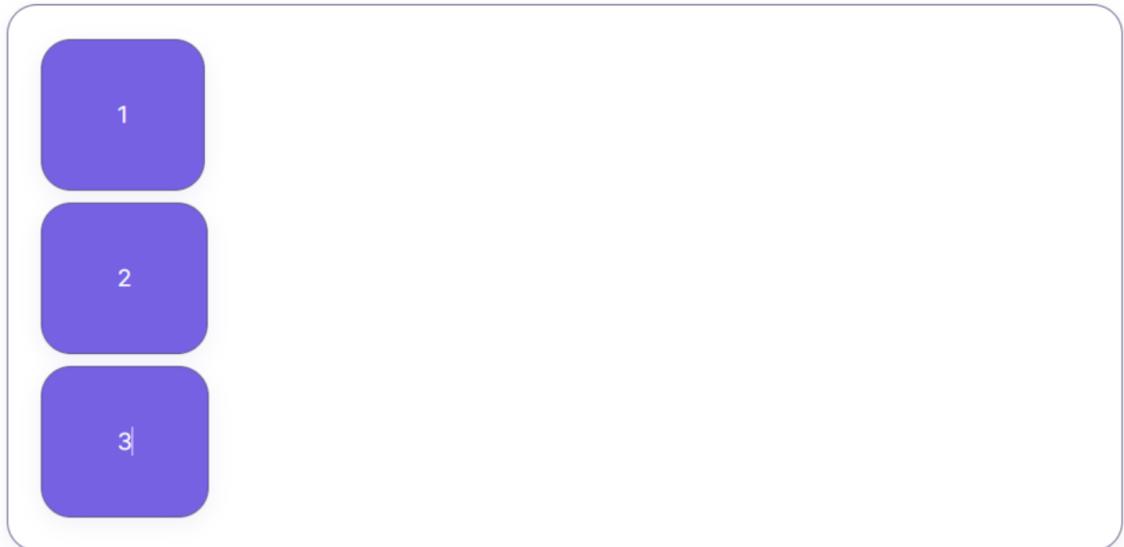
```
.container {  
    writing-mode: vertical-lr;  
}
```

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

2) flex-direction: column

horizontal-tb

인라인 방향은 가로, 블록 방향은 세로이며 **Main Axis** 는 세로, **Cross Axis** 가로이다.

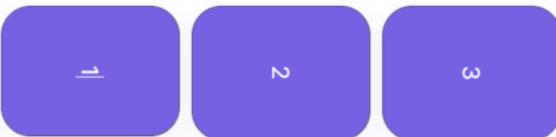


```
.container {  
  flex-direction: column;  
  writing-mode: horizontal-tb;  
}
```

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

vertical-lr

인라인 방향은 세로, 블록 방향은 가로이며 Main Axis는 가로, Cross Axis는 세로이다. **row**의 **vertical-lr** 과 마찬가지로 세로 쓰기로 전환되어 글씨가 눕혀지게 보인다.



```
.container {  
    flex-direction: column;  
    writing-mode: vertical-lr;  
}
```

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

1.6. 기본 속성 정보

1.6.1. Flex 알아보기

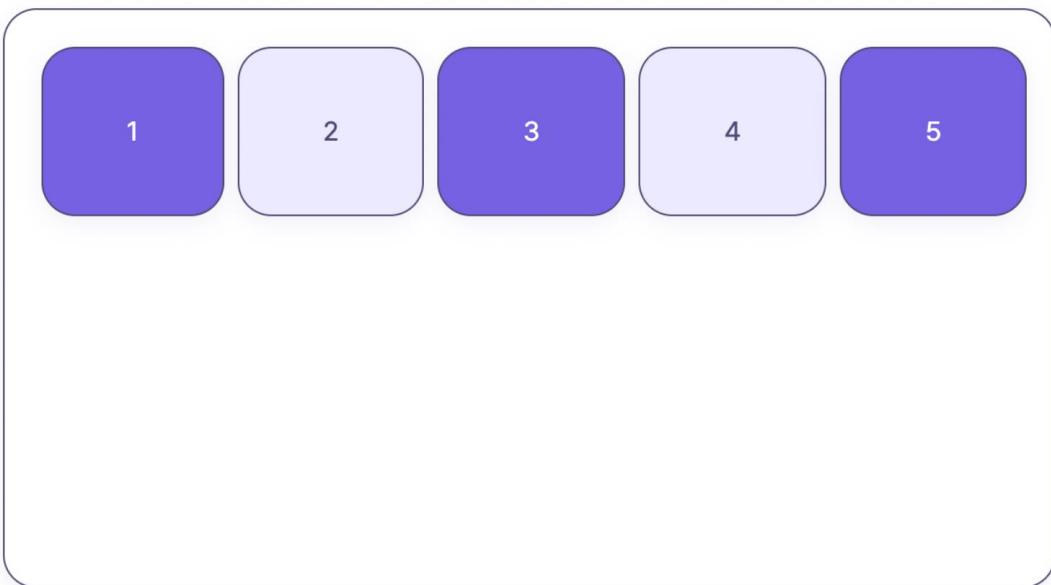
Flex는 flexbox, flexiblebox 라고도 한다. Flex는 행(row)과 열(column)의 형태의 레이아웃을 만들기 위해서 만들어졌다. Flex를 사용할 때, 자식요소(아이템)는 부모 요소(컨테이너)의 크기에 맞춰 변형될 수 있다. 또한, 많은 속성을 사용하면 더욱 다양한 형태로 사용이 가능하다.

Flex는 자식요소(아이템)이 그 자식요소(아이템)을 포함하는 부모 요소(컨테이너)의 공간에 맞춰 그 크기를 줄이거나, 늘이고 또는 부모 요소(컨테이너)에 맞춰 그 정렬을 맞추는 css display의 하나의 속성이다. flex를 보여주는 간단한 예로 살펴보자.



```
<body>
  <div class="container">
    <div class="item">1</div>
    <div class="item even">2</div>
    <div class="item">3</div>
    <div class="item even">4</div>
    <div class="item">5</div>
  </div>
</body>
```

위의 그림과 코드를 살펴보면, 코드에는 부모 요소(컨테이너) 안에 자식요소(아이템)들이 들어 있지만, 그림을 살펴보면 부모 요소(컨테이너)와 안에 있는 블록은 아무런 연관관계가 없어보인다. 하지만 여기서 display: flex를 넣는다면 아래와 같이 정렬된다.

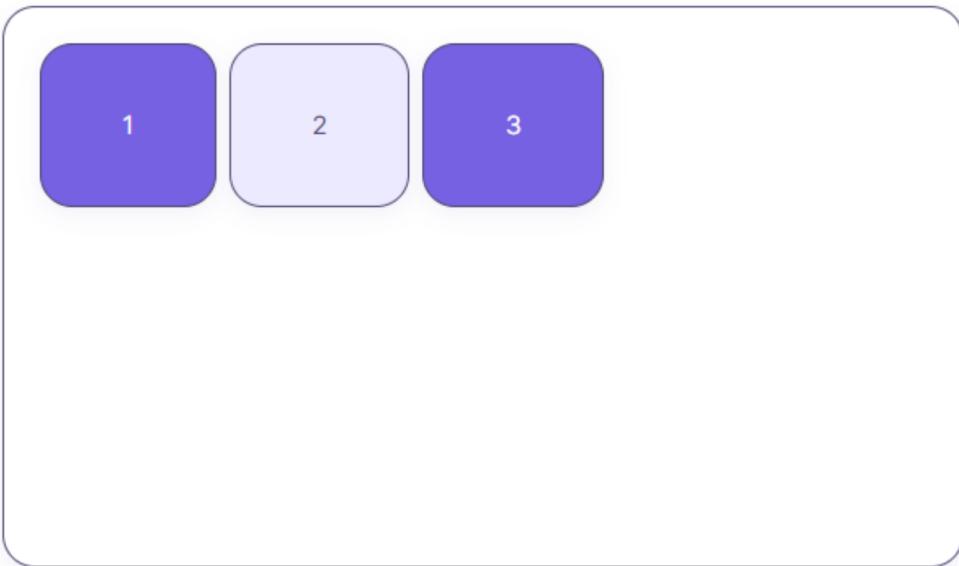


```
.container {  
    display: flex;  
}
```

```
<body>  
  <div class="container">  
    <div class="item">1</div>  
    <div class="item even">2</div>  
    <div class="item">3</div>  
    <div class="item even">4</div>  
    <div class="item">5</div>  
  </div>  
</body>
```

자식요소(아이템)들이 부모 요소(컨테이너)에 맞게 정렬된 모습을 볼 수 있다. 이렇게 flex는 부모 요소(컨테이너)의 공간에 맞게 자식요소(아이템)들의 크기나 정렬을 맞춰주는 속성이다. 그렇기 때문에 잘 사용한다면 부모 요소(컨테이너)와 자식요소(아이템)의 배치를 효율적으로 사용할 수 있는 편리한 기능이 될 것이다. 다음으로 flex를 어떻게 사용할 수 있는지 알아보고자 한다.

1.6.2. Flex 사용법



Flex는 크게 아이템과 컨테이너, 두 가지의 요소로 나누어진다. 위 그림에서는 box1, box2, box3이 **아이템**, 이를 감싼 보라색 테두리가 **컨테이너**이다. 이때 아이템은 자식요소가 되고 컨테이너는 부모요소가 된다. **컨테이너**는 큰 틀에서 전체적인 레이아웃에 영향을 주는 공간이다. 자식요소를 가로로 정렬할지, 세로로 정렬할지 정하는 것도 컨테이너에서 정한다. **아이템**은 각자가 어떤 식으로 정렬될지를 정하기 위해 존재한다. 일괄적으로 모두에게 동일한 속성을 적용하는 것이 아닌, 너비나 배치 순서를 바꾸는 등의 속성을 부여하여 커스텀 하는 것이 가능하다.

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

Flex는 새롭게 등장한 `display` 속성이기 때문에 `display: flex;`로 적용한다. 이후에는 Flex가 제공하는 속성을 활용해 레이아웃을 변화시킬 수 있다.

Flex의 속성도 어떤 요소에 적용되는지에 따라 두 가지로 구분된다. 따라서 적용을 원하는 요소에 적절한 속성을 사용하는 것이 중요하다. 이때 속성의 영향은 컨테이너의 직계자식까지만 영향을 준다는 특징이 있으니 주의해야 한다.

1.6.3. Flex 기본 속성

Flex는 기본적으로 **블록 레벨 요소**의 성질을 가지며 주로 부모의 속성을 통해 자식들을 컨트롤한다. 이때 부모를 **Flex-container**, 영향을 받는 자식들을 **Flex-item**이라고 부른다. Flex를 사용하다 보면 기본 속성값에 대해서 한 번쯤 생각하게 된다. 원하는 대로 요소들이 움직이지 않는다면 아래에 첨부한 표를 참조하여 기본 속성에 대해서 찾아보기를 권장한다.

이제 Flex의 다양한 속성을 배우게 될 텐데, 해당 속성이 어떤 축을 기준으로 적용이 되는지가 모두 다르기 때문에, 반드시 축에 대한 이해를 바탕으로 Flex를 학습해나가는 것이 중요하다. Flex는 기본적으로 **블록 레벨 요소**의 성질을 가지며 주로 부모의 속성을 통해 자식들을 컨트롤한다. 이때 부모를 **Flex-container**, 영향을 받는 자식들을 **Flex-item**이라고 부른다. 원하는 대로 요소들이 움직이지 않는다면 아래에 첨부한 표를 참조하여 기본 속성에 대해서 찾아보기를 권장한다.

	기본 속성	적용 대상
flex-direction	row	flex containers
justify-content	normal	flex containers
align-items	normal	all elements
align-content	normal	multi-line flex containers
flex-wrap	nowrap	flex containers
flex-basis	auto	flex items
flex-grow	0	flex items
flex-shrink	1	flex items

어떠한 속성을 적용해야 하는지 헷갈리는 경우, 컨테이너에 flex 속성을 적용한 뒤 크롬 개발자 도구의 스타일 탭을 확인하면 좋다. 주요한 속성 다섯 가지를 GUI 기능으로 제공하여 적용된 모습을 확인해볼 수 있기 때문이다.

2. Flex-direction

`flex-direction` 이란 flex container 내의 item들의 정렬 방향을 결정하는 속성이다. `flex-direction`은 자식요소(아이템)의 부모요소인 (컨테이너)에 적용하여 사용하며 종류는 `row`, `row-reverse`, `column`, `column-reverse` 네 가지가 있다.

2.1. flex-direction : row

왼쪽에서 오른쪽으로 글씨가 써지는 **행 방향 정렬**이며, `flex-direction` 을 따로 설정하지 않으면 초기값으로 설정된다.



```
.container {  
  display: flex;  
  flex-direction: row;  
}
```

2.2. flex-direction : row-reverse

row 와 동일하게 보여지지만 시작점과 끝점이 반대로 위치한다.

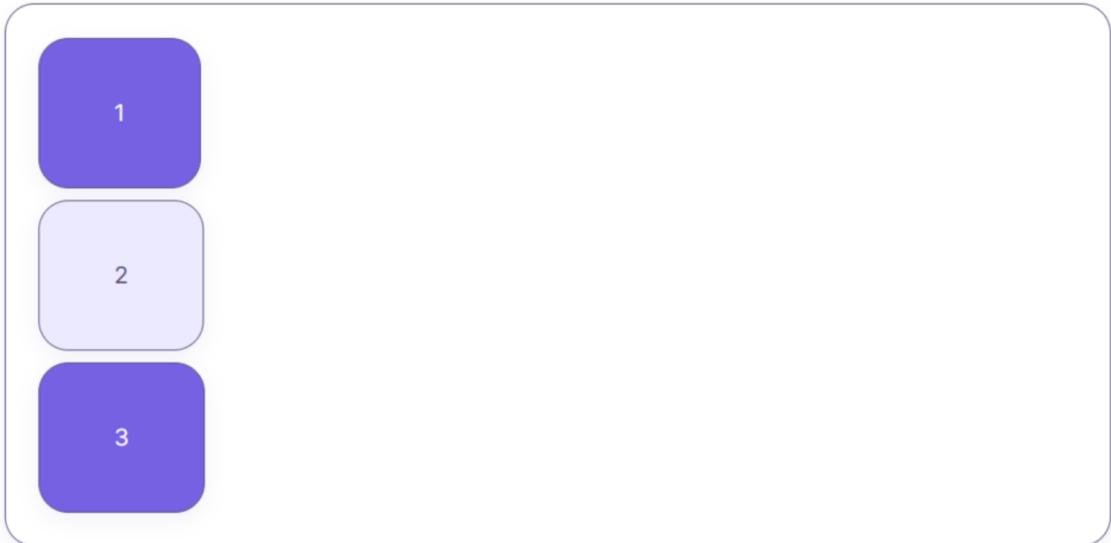


```
.container {  
    display: flex;  
    flex-direction: row-reverse;  
}
```

컨테이너 내에 위치한 아이템들을 유연하게 정렬 할 수 있으며, 역순으로 정렬한다. 컨테이너에게 **display: flex** 속성을 부여하지 않으면 아이템은 유연하게 움직이지 않는다.

2.3. flex-direction : column

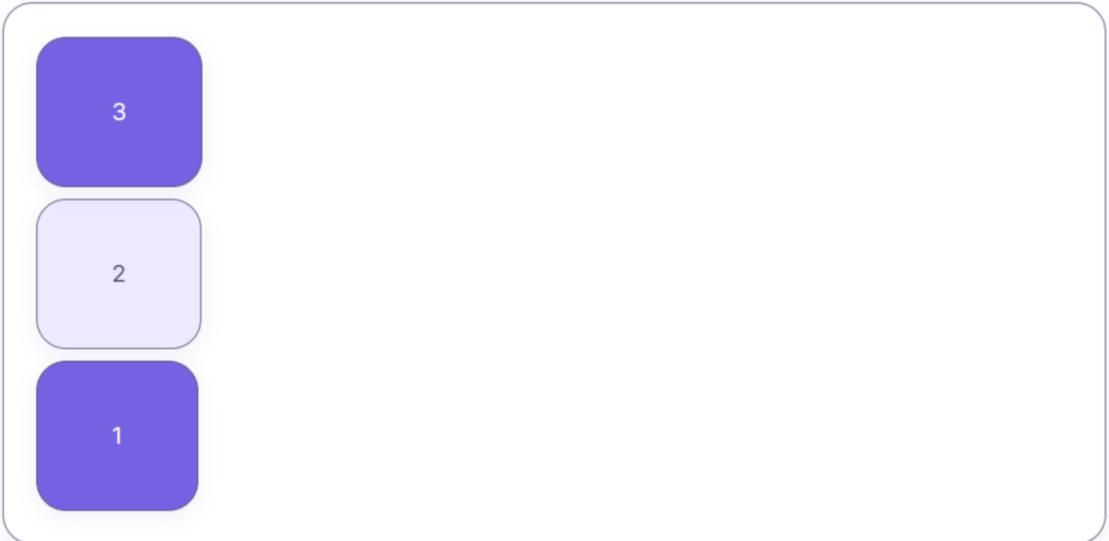
Flex 내 box들을 수직으로 정렬한다. 컨테이너 내부의 요소들이 아이템이 아닌 경우 해당 속성의 값은 적용되지 않는다. `flex-direction : column;` 일 때 주축은 세로 방향이 되고, 교차 축은 가로 방향이 된다. 아이템을 추가하는 경우 마크업 순서에 맞게 위에서 아래로 흐르듯이 정렬된다.



```
.container{  
    display : flex;  
    flex-direction : column;  
}
```

2.4. flex-direction : column-reverse

`column` 과 동일하게 보여지지만 시작 점과 끝 점이 반대로 위치한다.



```
.container{  
    display : flex;  
    flex-direction : column-reverse;  
}
```

2.5. 접근성 고려사항

`flex-direction` 속성에 `row-reverse` 또는 `column-reverse` 값을 사용하면 DOM 구조와 시각적 표현에 차이가 발생하여 스크린 리더로 접근하는 사용자에게 잘못된 정보를 전달 할 가능성이 있다.

3. Justify-content

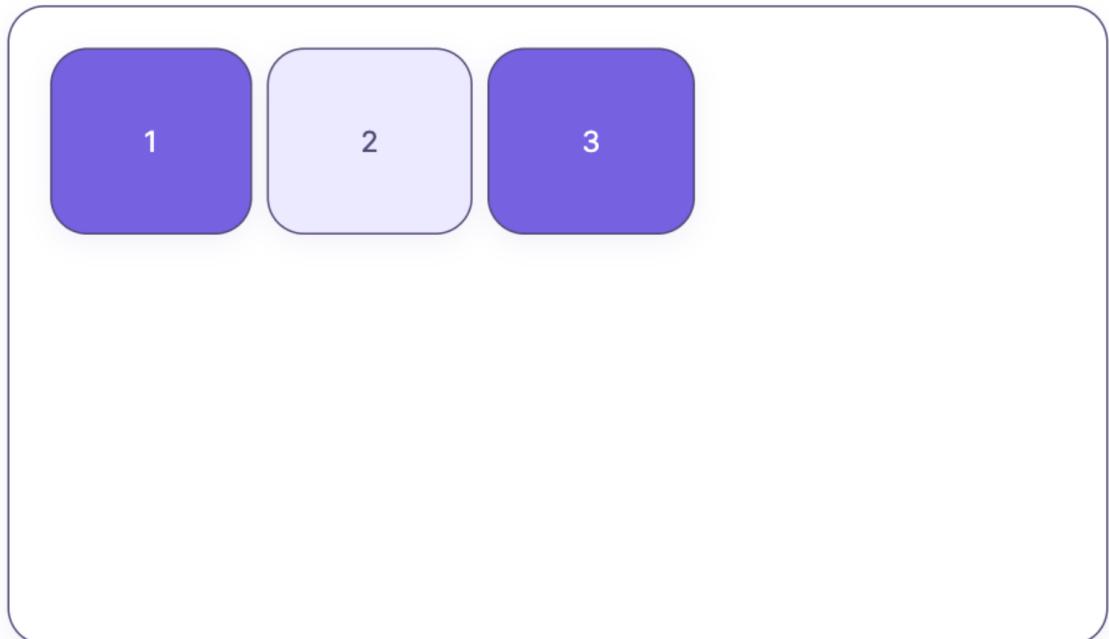
3.1. flex-start

부모 요소(컨테이너)에 display: flex, justify-content: flex-start 값을 주면 된다.

```
.container {  
    display: flex;  
    justify-content: flex-start;  
}
```

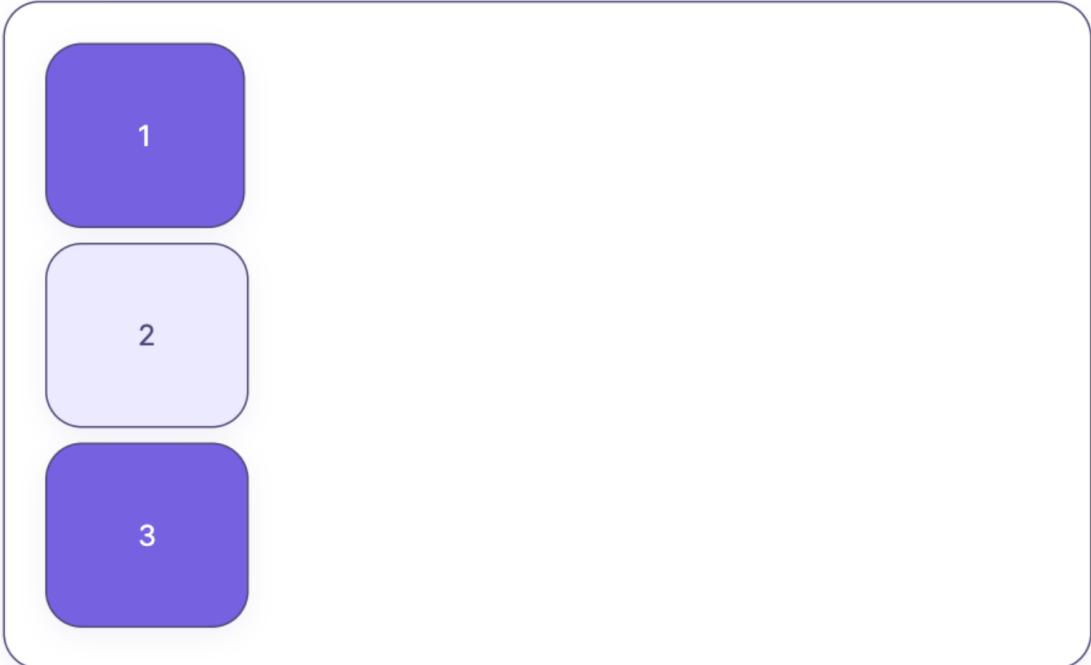
justify-content: flex-start; 는 기본값으로 Flex 아이템들을 시작점을 기준으로 정렬한다. 기본값일 때, 기준이 되는 점은 왼쪽 축의 상단이다.

flex-direction: row; 인 경우에는 가로(row=행)로 정렬 하며 왼쪽 상단이 기준점이 되어서 왼쪽 상단부터 차례대로 1, 2, 3번 순으로 아이템이 배치된다.



Flex_3 Justify-content

그렇지만 `flex-direction: column;` 을 준 경우에는, 컨테이너의 상단이 기준점인 것은 같지만 flex 방향이 row(행)가 아닌 column(열)이기 때문에 세로(열)로 아이템이 배치된다.



```
.container {  
  display: flex;  
  justify-content: flex-start;  
  flex-direction: column;  
}
```

3.2. flex-end

부모 요소(컨테이너)에 display: flex;, justify-content: flex-end; 값을 주면 된다.

```
.container {  
    display: flex;  
    justify-content: flex-end;  
}
```

justify-content: flex-end; 는 자식 요소(아이템)를 맨 끝점으로 배치하는데, 기본값(왼쪽 축)의 반대인 오른쪽 축의 상단이 기준이 된다.

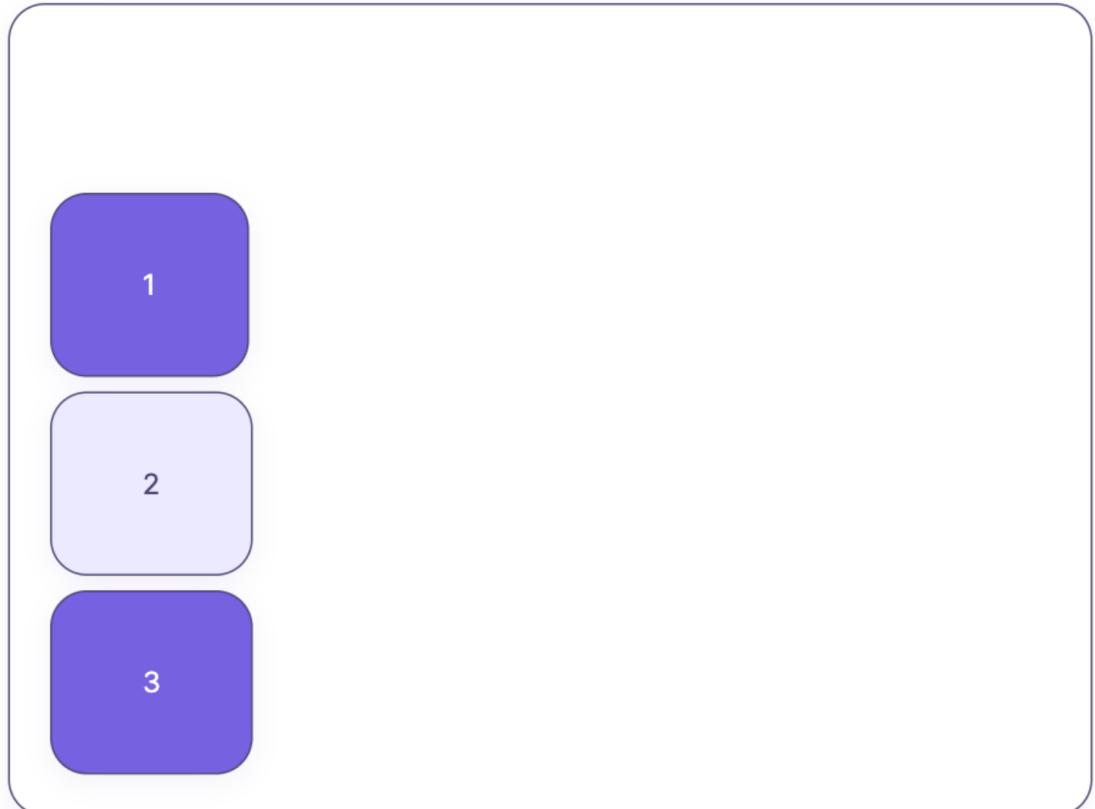
flex-direction: row; 인 경우일 때에는 row(행)로 정렬하기 때문에 자식 요소(아이템)인 1, 2, 3번의 아이템들이 오른쪽 축에 붙어 배치된다.



Flex_3 Justify-content

`flex-direction: column;` 을 준 경우에는, flex 방향이 row(행)가 아닌 column(열)이 되기 때문에 세로(열)로 자식 요소(아이템)가 배치된다.

그리고 `justify-content: flex-end;` 의 경우, 자식 요소(아이템)를 배치하는 기준이 기본값(상단 축)의 반대인 하단 축이 되기 때문에 1, 2, 3번의 아이템이 세로로 정렬되어 하단에 붙어 배치된다.



```
.container {  
    display: flex;  
    justify-content: flex-end;  
    flex-direction: column;  
}
```

3.3. center

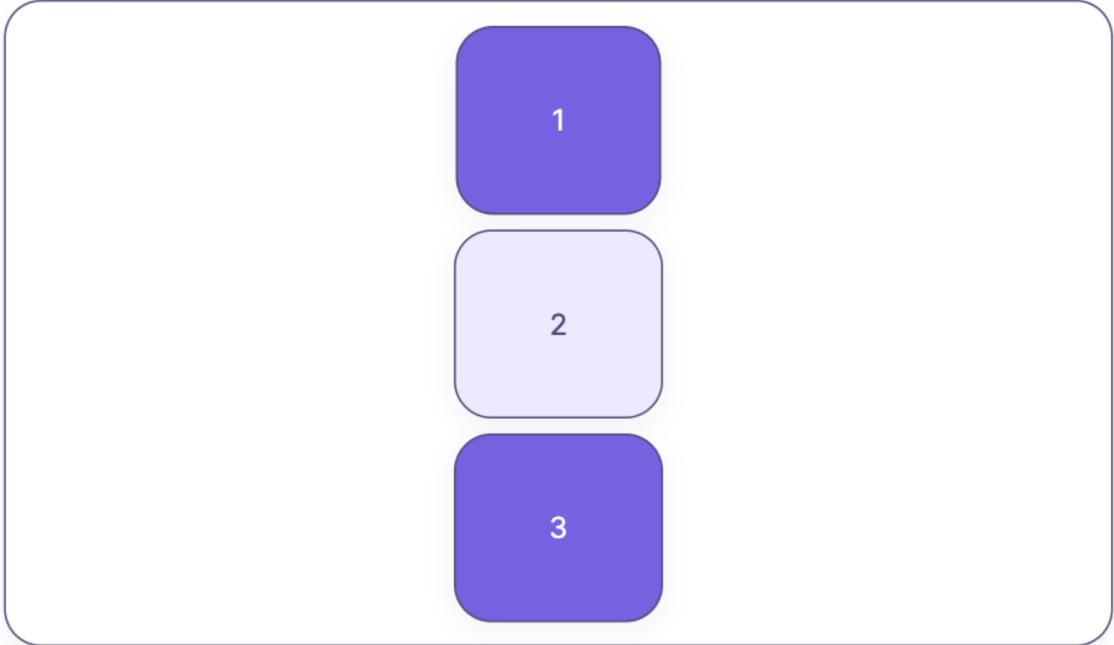
`justify-content: center` 를 준 경우, Flex-box 내의 요소들을 주축(main-axis)의 중앙을 기준으로 정렬한다. 즉, 요소들을 가운데 정렬해준다. 이때 요소의 간격은 균등 분할된다. 개별적으로 빈 공간을 제어하고 싶다면 해당 요소에 별도의 margin 값을 주어야 한다. `space-between`, `space-around` 역시 동일하다.

```
.container {  
    display: flex;  
    justify-content: center;  
}
```



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

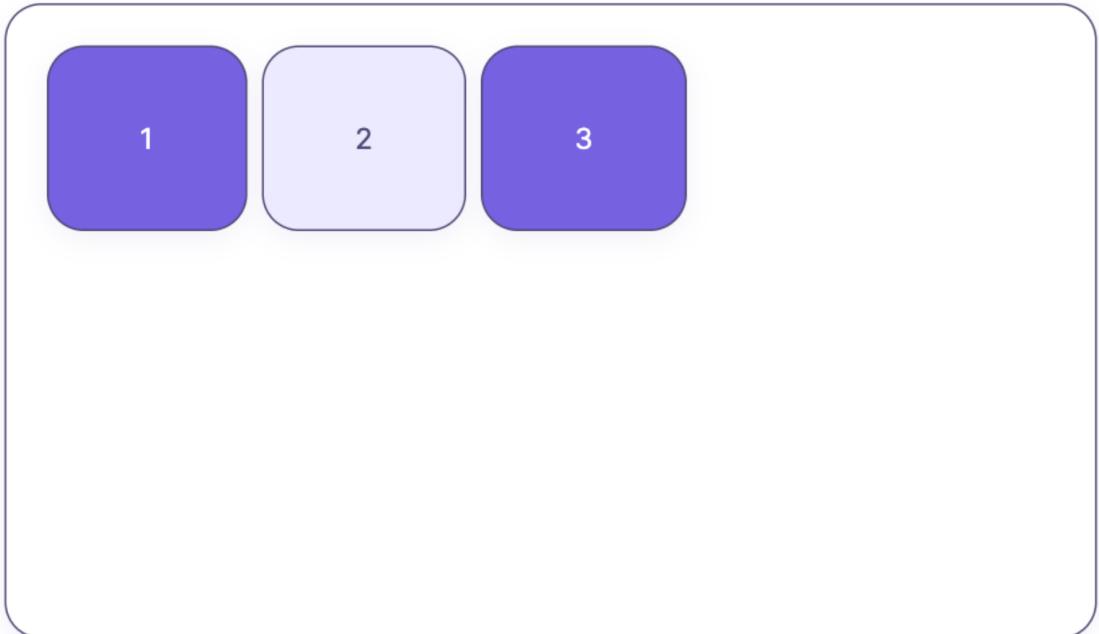
Flex_3 Justify-content



```
.container {  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  align-items: center;  
}
```

3.4. space-between, space-around, space-evenly

다음으로 `space-between`, `space-around`, `space-evenly`의 비슷하면서도 다른 점들을 알아보자. 위셋의 공통점은 자식 요소들을 메인 축으로 정렬하며, 일정한 간격을 준다는 것이다. `between`, `around`, `evenly`의 뜻은 차례로 '사이에', '둘레에', '균등하게'라는 뜻이다. 그럼 뜻을 생각하며 아래 그림으로 차이점을 확인해 보자.



```
.container {  
  display: flex;  
  justify-content: flex-start;  
}
```

아무 정렬도 지정해주지 않았을 때의 기본값이다.

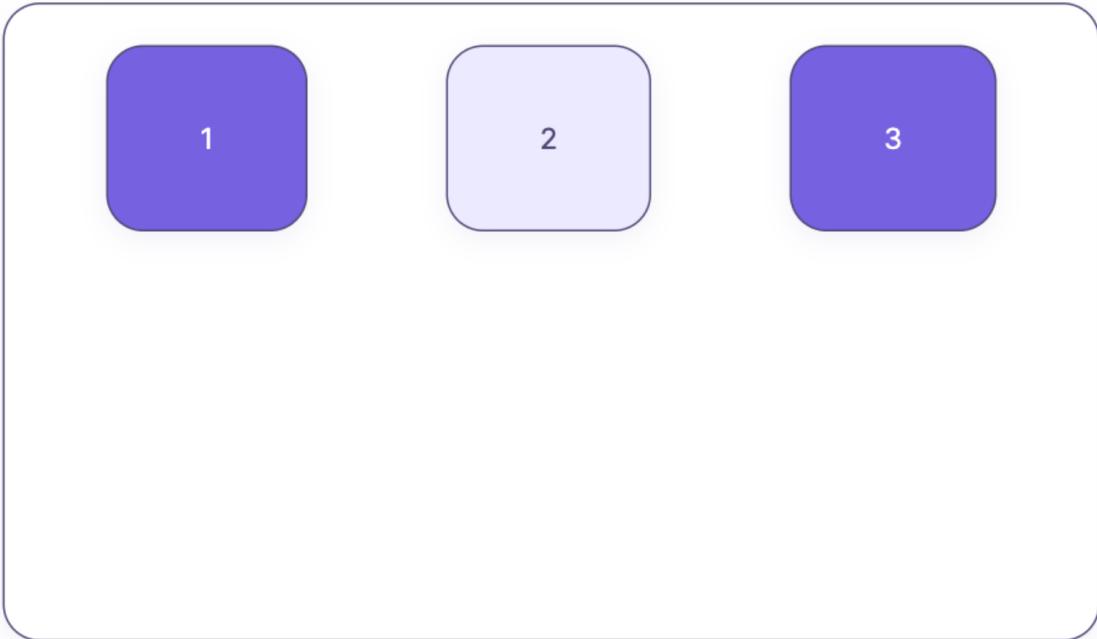
3.4.1. space-between



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

`justify-content: space-between;` 을 주었을 때 자식 요소들의 '사이에' 같은 간격이 들어간 것을 볼 수 있다. 컨테이너의 패딩 값을 제외하면 컨테이너의 너비에 꽉 차게 배치된다.

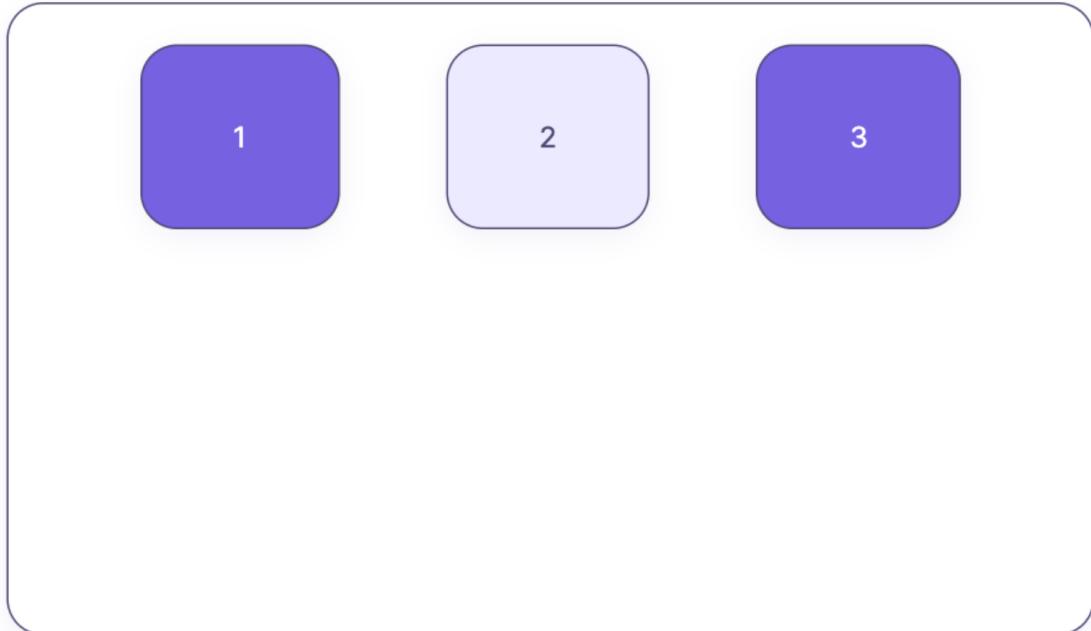
3.4.2. space-around



```
.container {  
    display: flex;  
    justify-content: space-around;  
}
```

아이템의 '둘레에' 같은 간격이 들어간 것을 볼 수 있다. 아이템 사이에는 같은 간격이 2번씩 들어가게 되므로 양 끝 컨테이너 사이의 간격은 아이템 사이의 간격의 1/2이 된다.

3.4.3. space-evenly



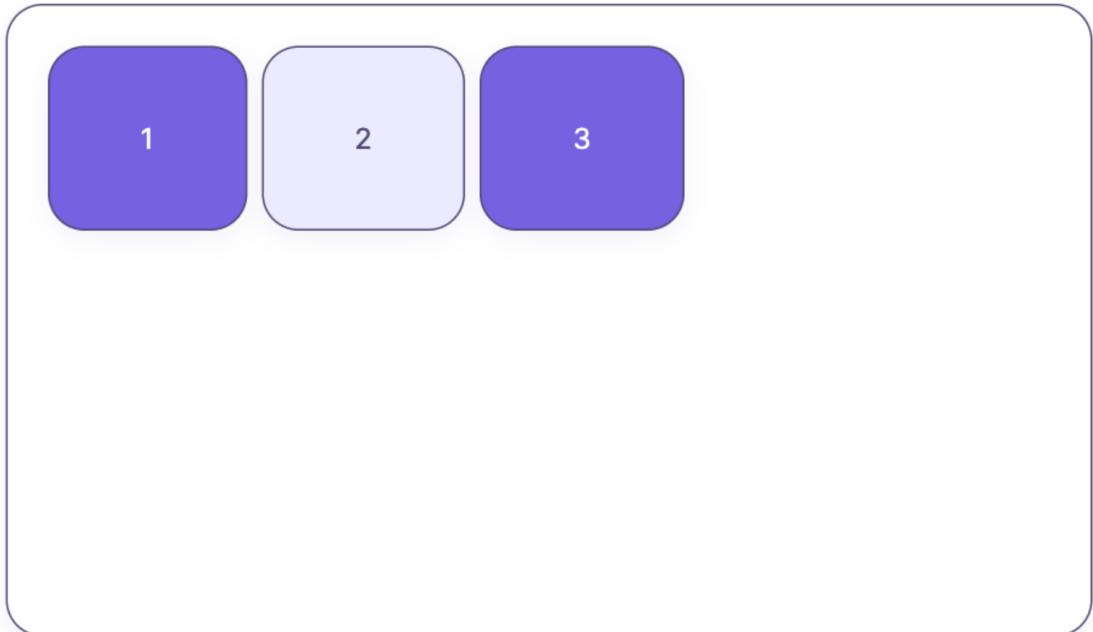
```
.container {  
    display: flex;  
    justify-content: space-evenly;  
}
```

모든 간격이 '균등하게' 배치된 것을 볼 수 있다. space-around와 비슷하게 보이지만 양 끝의 간격과 자식 요소 사이의 간격이 같은 것을 확인할 수 있다.

3.5. 자주 사용하지 않는 속성들

3.5.1. justify-content : normal

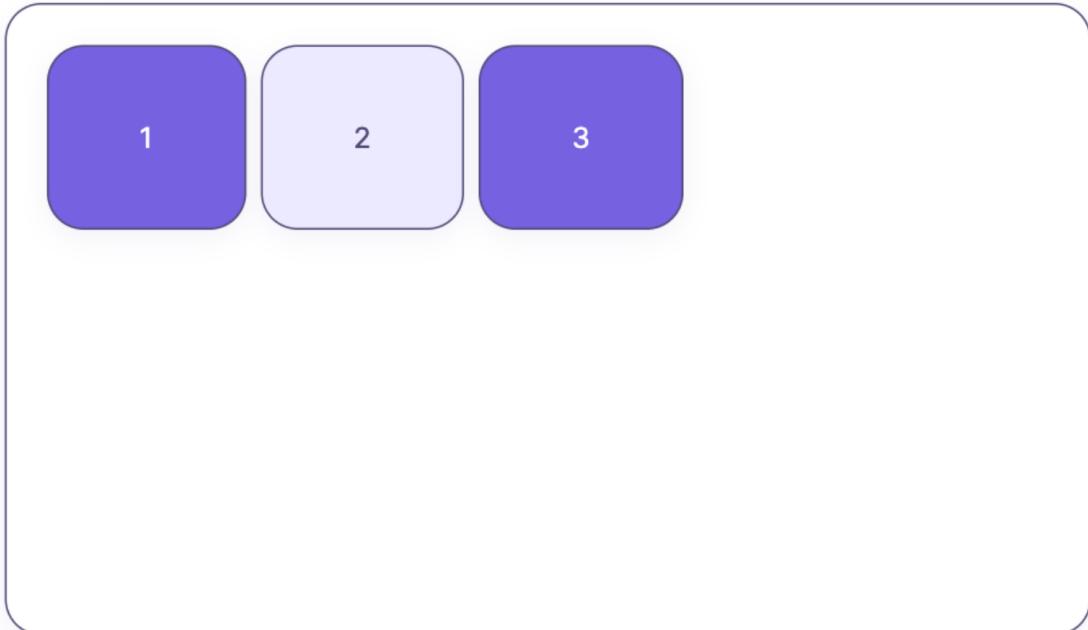
값이 설정되지 않은 것처럼 기본 위치에 정렬한다. 기본값이므로 생략 할 수 있다.



```
.container {  
    display: flex;  
    justify-content: normal;  
    align-items: flex-start;  
}
```

3.5.2. justify-content : initial

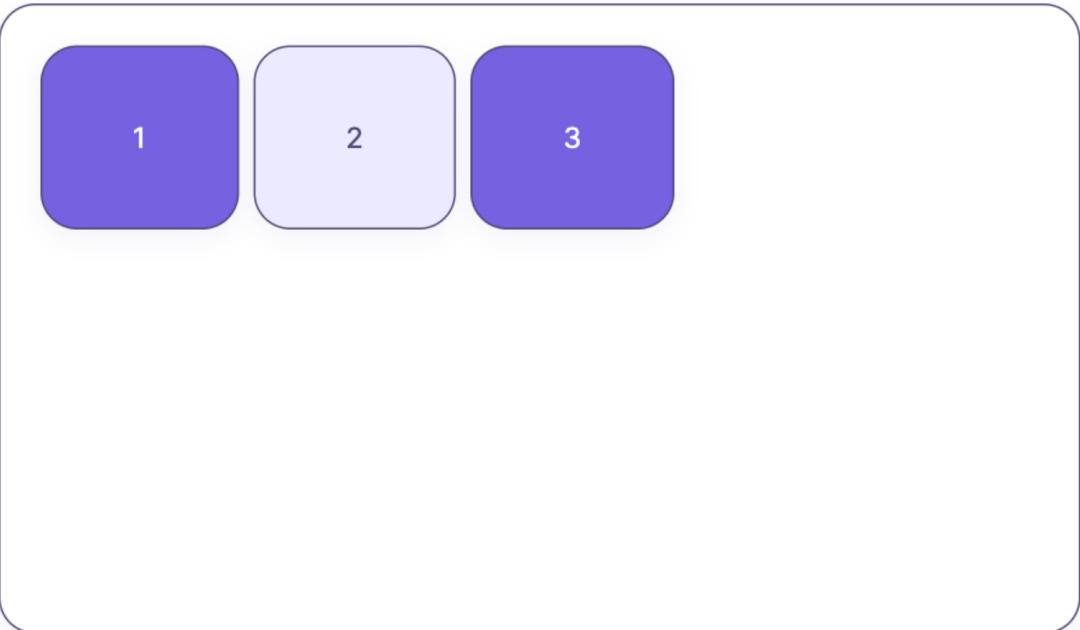
기본값으로 정렬한다. justify-content의 기본값은 normal이므로 위의 justify-content:normal과 결과가 동일하게 나온다.



```
.container {  
    display: flex;  
    justify-content: initial;  
    align-items: flex-start;  
}
```

3.5.3. justify-content : inherit

상속되는 값에 따라 정렬한다. container에 class 명이 “wrap-container”인 부모 요소를 만들어 주었다. wrap-container의 justify-content:center 속성이 상속되어 자식 요소인 container의 justify-content도 center로 적용된다.



```
.wrap-container {  
    display: flex;  
    justify-content: center;  
}  
  
.container {  
    display: flex;  
    justify-content: inherit;  
}
```

```
<body>  
    <div class="wrap-container">  
        <div class="container">  
            <div class="item">1</div>  
            <div class="item even">2</div>  
            <div class="item">3</div>  
        </div>  
    </div>  
</body>
```



3.5.4. justify-content : revert

상속된 속성으로부터 기본값으로 되돌린다. 특정 사이트만의 css 속성이 지정되어 있거나 사용자가 정의한 스타일이 적용된 속성을 부모 속성 또는 사용자 에이전트의 기본 스타일로 되돌린다. 사용자 에이전트란 브라우저마다 정해놓은 css 기본 규칙이다.

3.5.5. justify-content : unset

부모 요소로부터 상속받는 값이 있으면 inherit로 적용되고 상속되는 값이 없다면 initial 속성으로 정렬한다.

3.5.6. firefox에서만 지원하는 속성

`justify-content: revert-layer;`

이전 레이어에 지정된 스타일로 되돌릴 수 있다. revert와 달리 작성자 원본에 적용된 스타일을 제거하고 사용자 원본 또는 사용자 에이전트의 기본값으로 되돌린다.

`justify-content: safe;`

다른 정렬 속성과 함께 사용된다. 항목이 컨테이너를 오버플로 하여 데이터가 손실된다면 항목이 start 속성처럼 정렬된다.

`justify-content: unsafe;`

다른 정렬 속성과 함께 사용된다. 컨테이너의 상대적 크기와 관계없이, 데이터 손실을 일으키는 오버플로가 발생할 수 있는지와 관계없이 지정된 정렬 값이 적용됩니다.

4. Align-items, Align-content

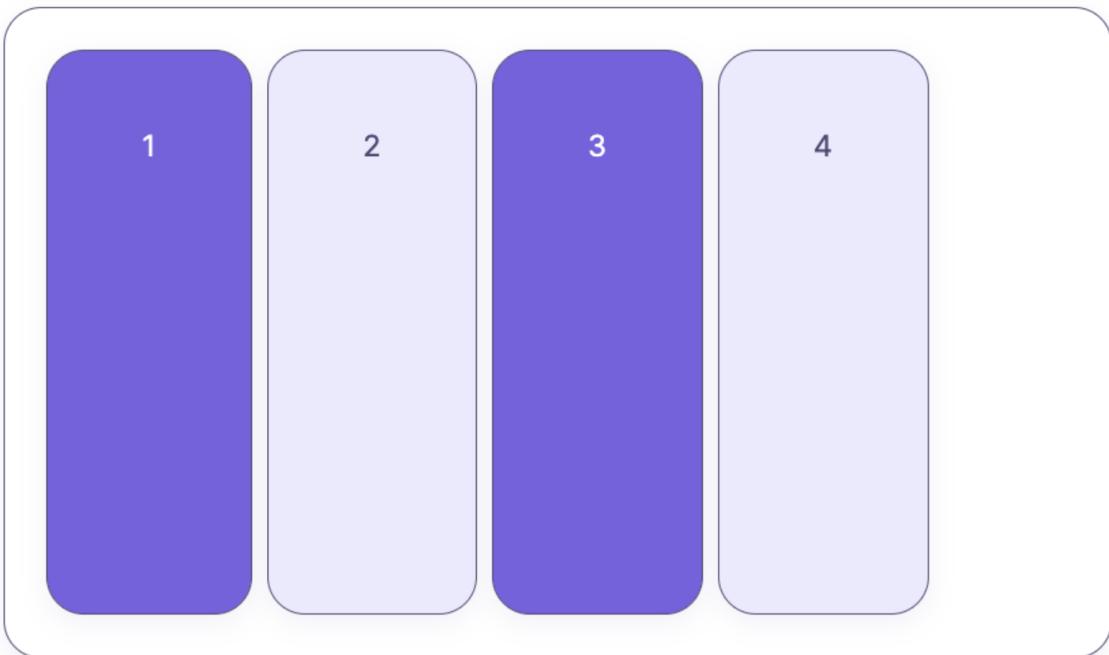
4.1. align-items

align의 사전적 정의는 '일직선으로 맞추다'이다. 즉, flex에서 align이란 축의 수직 방향 정렬을 의미하며, 부모요소(컨테이너) 안에서 교차 축에 따라 자식요소(아이템)가 정렬되는 속성이다.

속성

- stretch

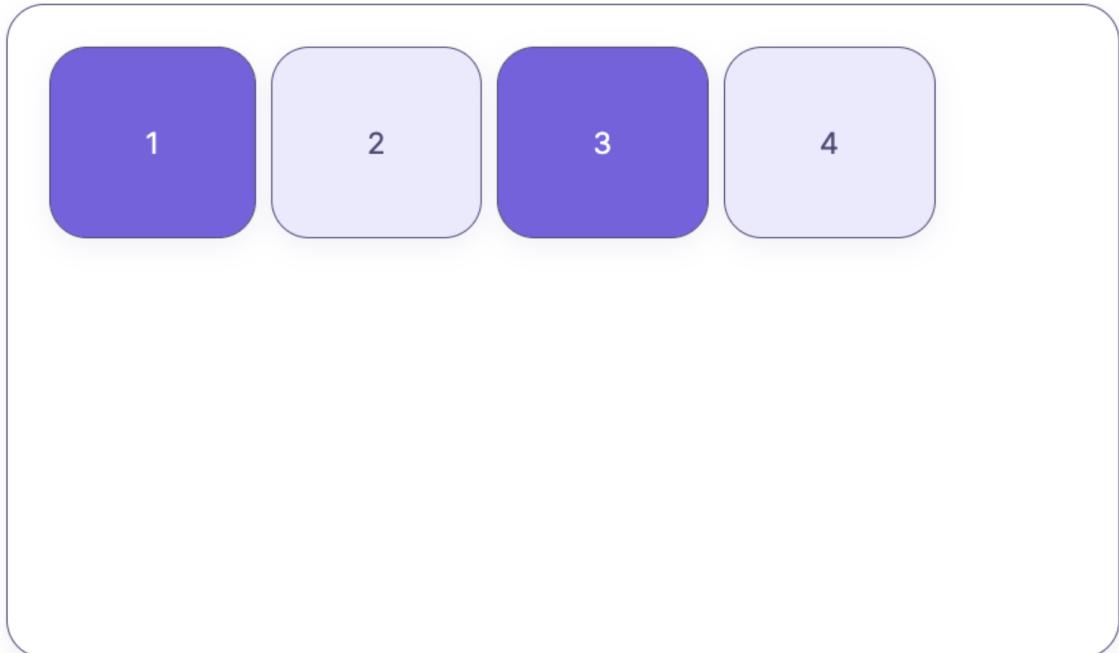
align-items의 기본값이다. 부모요소(컨테이너)의 높이만큼 자식요소(아이템) 높이가 같이 늘어난다.



```
.container {  
  display: flex;  
  align-items: stretch;  
}
```

- flex-start

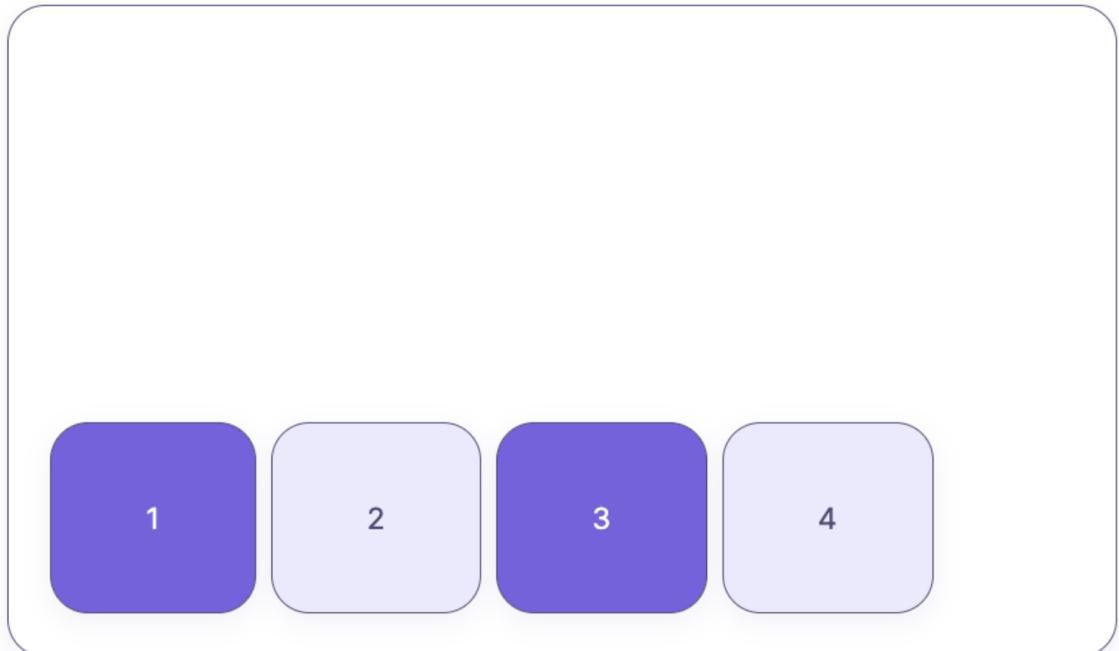
자식요소(아이템)는 교차 축 시작점에서 정렬된다. 기본적으로 가로축은 수직을 의미하며, 자식요소(아이템)는 상단에서 수직으로 정렬된다.



```
.container {  
    display: flex;  
    align-items: flex-start;  
}
```

- flex-end

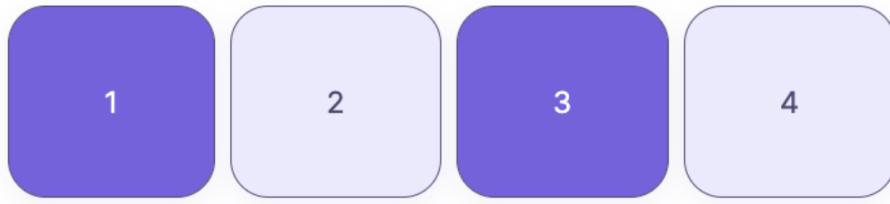
자식요소(아이템)는 교차 축 끝점에서 정렬된다. 기본적으로 가로축은 수직을 의미하며, 자식요소(아이템)는 하단에서 수직으로 정렬된다.



```
.container {  
    display: flex;  
    align-items: flex-end;  
}
```

- center

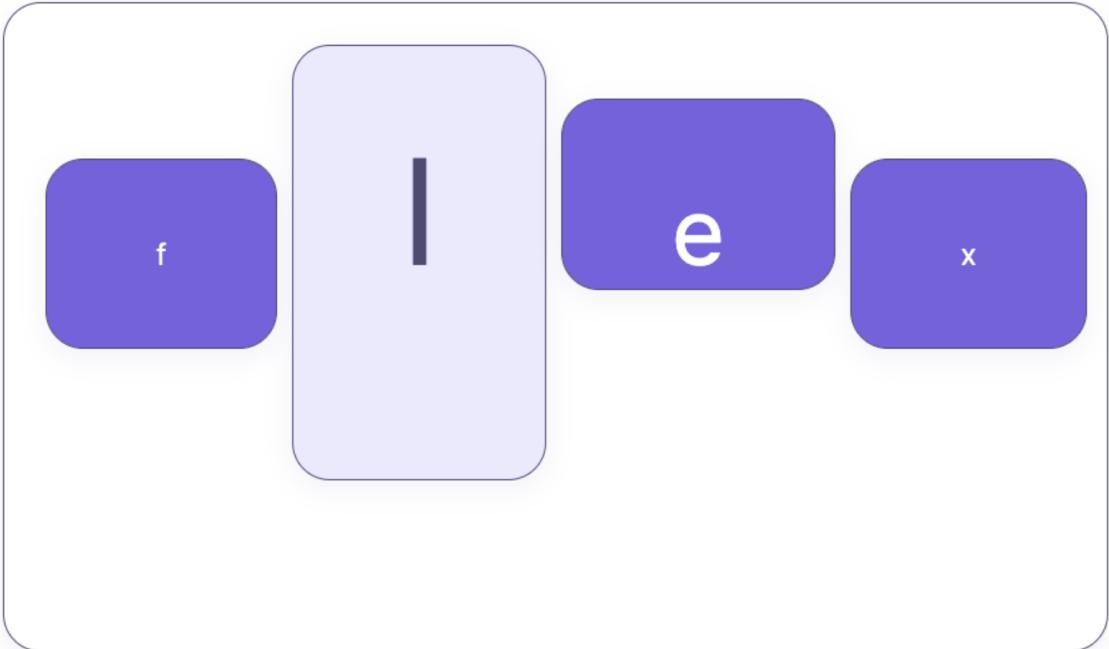
자식요소(아이템)는 교차 축 중앙으로 정렬된다.



```
.container {  
  display: flex;  
  align-items: center;  
}
```

- **baseline**

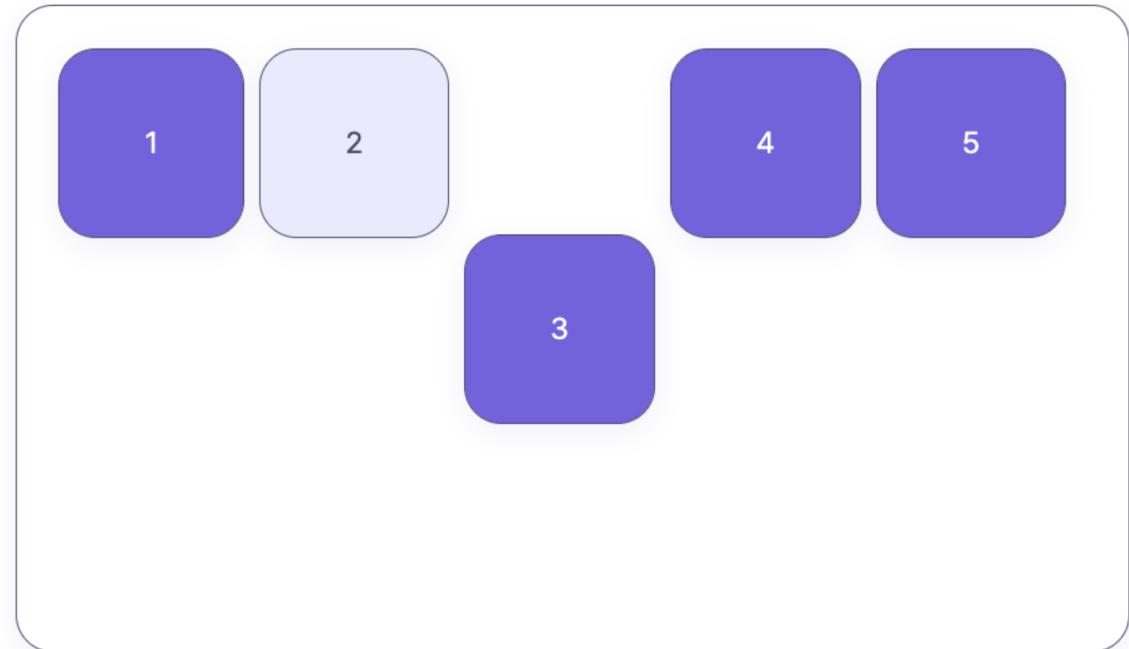
자식요소(아이템)는 교차 축의 기준선에 정렬이 된다. 텍스트의 기준선이 수평선을 따라 아이템이 정렬된다.



```
.container {  
    display: flex;  
    align-items: baseline;  
}
```

3. align-self와 다른 점은?

align-items는 부모 요소(컨테이너) 내에 있는 자식 요소를 대상으로 적용하는 반면에, align-self는 부모 요소(컨테이너) 안에서 특정 아이템만 교차 축으로 정렬된다. 즉, 특정 아이템의 정렬을 따로 하고 싶다면 이 속성을 사용하면 된다.



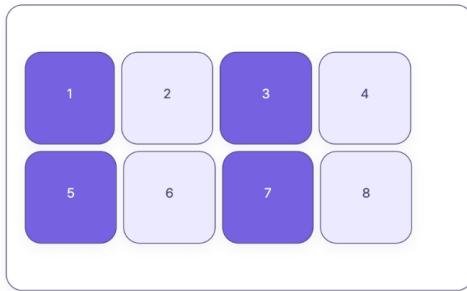
```
.container {  
    display: flex;  
    align-items: flex-start;  
}  
  
.third_item {  
    align-self: center;  
}
```

플렉스 박스 컨테이너에 `align-items : flex-start` 를 적용하고 'third_item' 이름을 가진 특정 아이템만 `align-self : center` 를 적용했다. 위와 같이 3번 아이템만 중앙정렬 속성이 적용되는 것을 볼 수 있다.

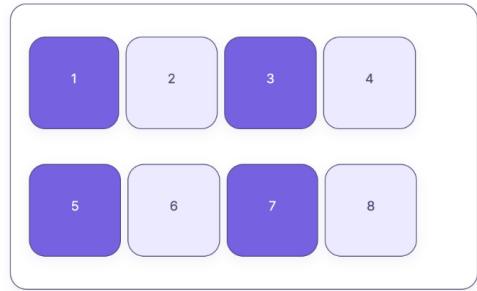
4.2. align-content

`align-items` 는 한 줄을 기준으로 정렬하지만, `align-content`는 `flex-wrap : wrap;` 이 설정된 경우 두 줄부터 기준으로 정렬한다. 아래 예시를 보면, 더 쉽게 알 수 있다.

`align-content: center;`



`align-items: center;`



좌측은 `align-items : center;`, 우측은 `align-content : center;`로 정렬한 결과화면이다. `align-items`는 아이템 한 줄마다 속성이 적용되었기 때문에 두 줄 사이에 간격이 생겨 정렬된 것을 볼 수 있다. 반면에, `align-content`는 두 줄 이상의 아이템을 수직축 방향으로 라인 자체를 정렬하기 때문에 두 줄 사이에 간격이 생기지 않은 것을 볼 수 있다.

속성

- `stretch`

부모요소(컨테이너)의 높이만큼 자식요소(아이템) 높이가 같이 늘어난다.



```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: stretch;  
}
```

- **flex-start**

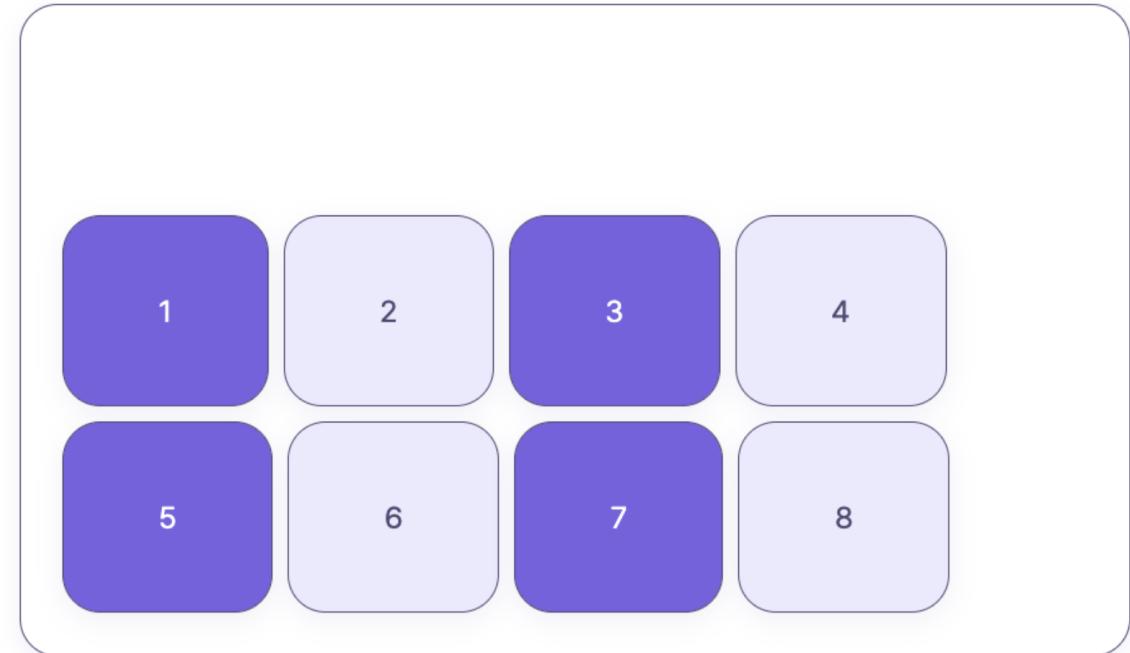
부모요소(컨테이너)의 최상단에서부터 정렬을 시작한다.



```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: flex-start;  
}
```

- flex-end

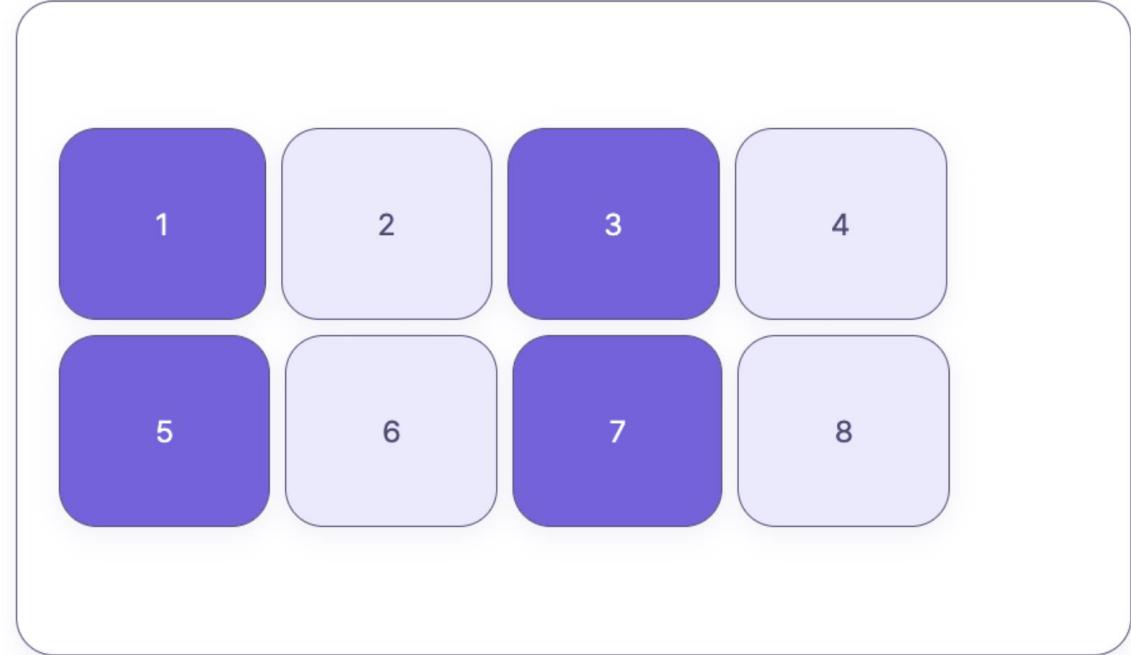
부모요소(컨테이너)의 최하단에서부터 정렬을 시작한다.



```
.container {  
  display: flex;  
  flex-wrap: wrap;  
  align-content: flex-end;  
}
```

- center

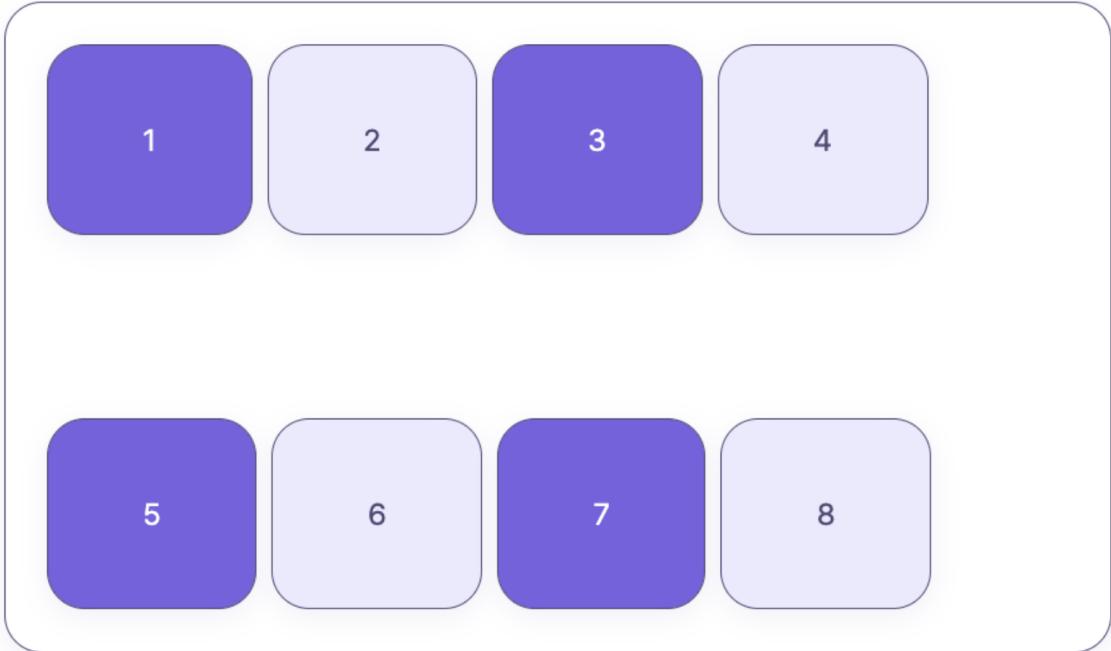
부모요소(컨테이너) 교차 축의 중앙에 정렬합니다.



```
.container {  
  display: flex;  
  flex-wrap: wrap;  
  align-content: center;  
}
```

- **space-between**

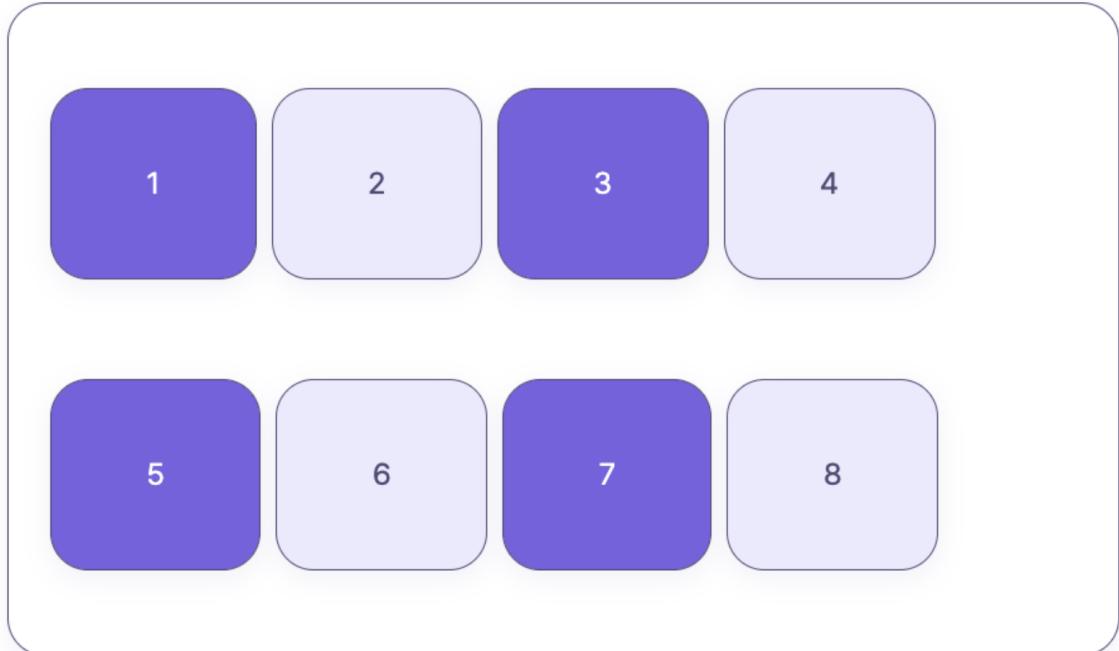
부모요소(컨테이너) 교차 축의 양 끝(시작점, 종료점)에 맞춰 아이템 간에 일정한 간격을 두고 정렬한다.



```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: space-between;  
}
```

- space-around

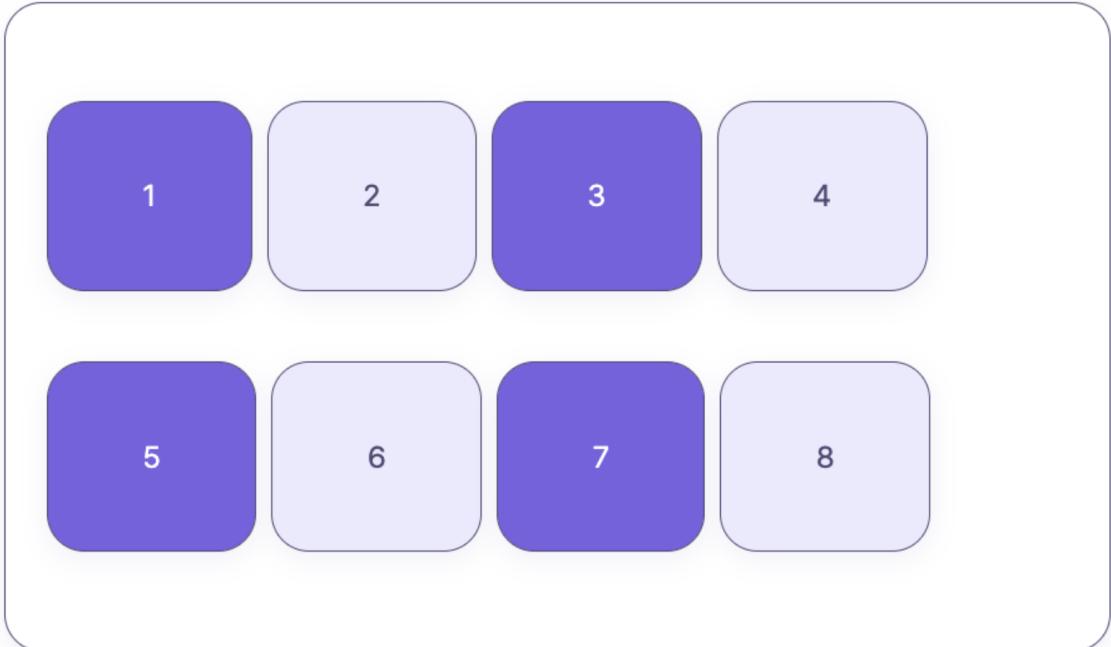
space-between과 마찬가지로 부모요소(컨테이너) 교차 축의 양 끝(시작점, 종료점)에 맞춰 아이템 간에 일정한 간격을 두고 정렬한다. 이때 첫 아이템 앞 여백과 마지막 아이템 뒤 여백은 각 아이템 간 거리의 절반 크기를 차지하여 정렬된다.



```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: space-around;  
}
```

- space-evenly

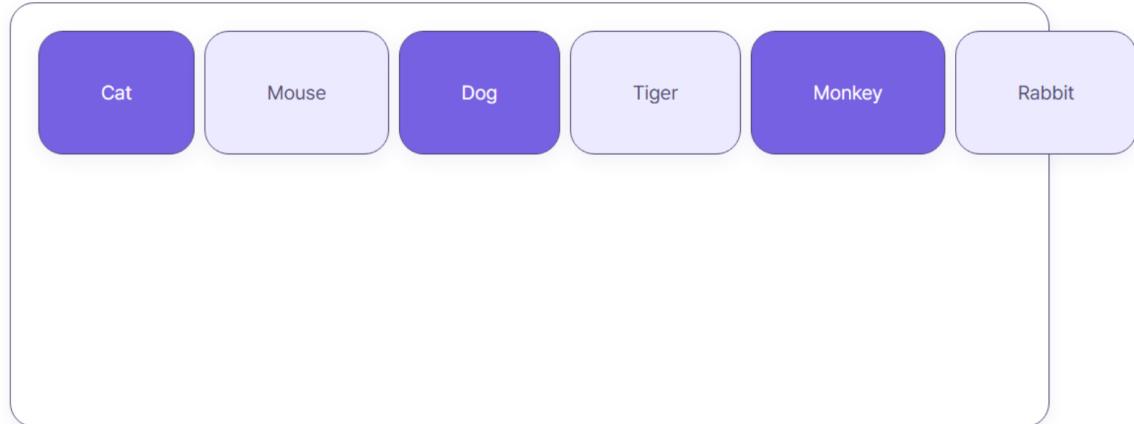
space-between과 마찬가지로 부모 요소(컨테이너) 교차 축의 양 끝(시작점, 종료점)에 맞춰 아이템 간에 일정한 간격을 두고 정렬한다. 이때 아이템 간의 거리, 첫 아이템 앞 여백과 마지막 아이템 뒤 여백의 크기가 모두 동일하게 정렬된다. 본 속성은 IE, Edge에서는 지원되지 않는다.



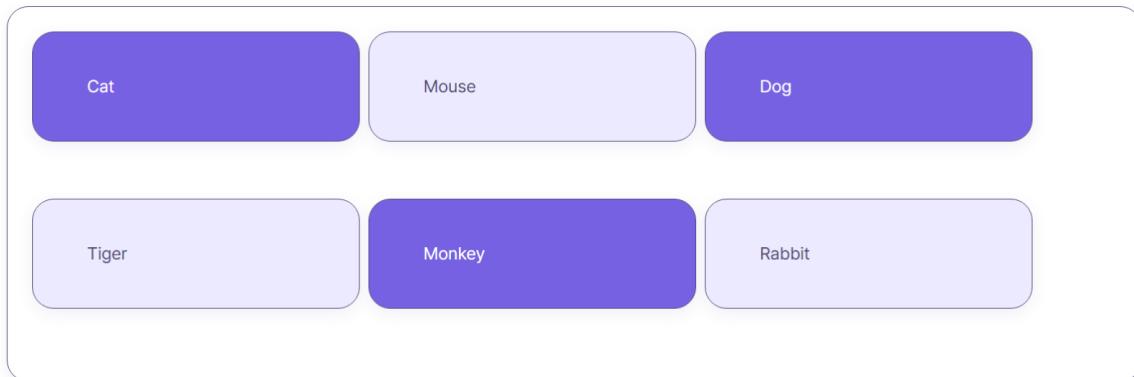
```
.container {  
    display: flex;  
    flex-wrap: wrap;  
    align-content: space-evenly;  
}
```

5. Flex-wrap

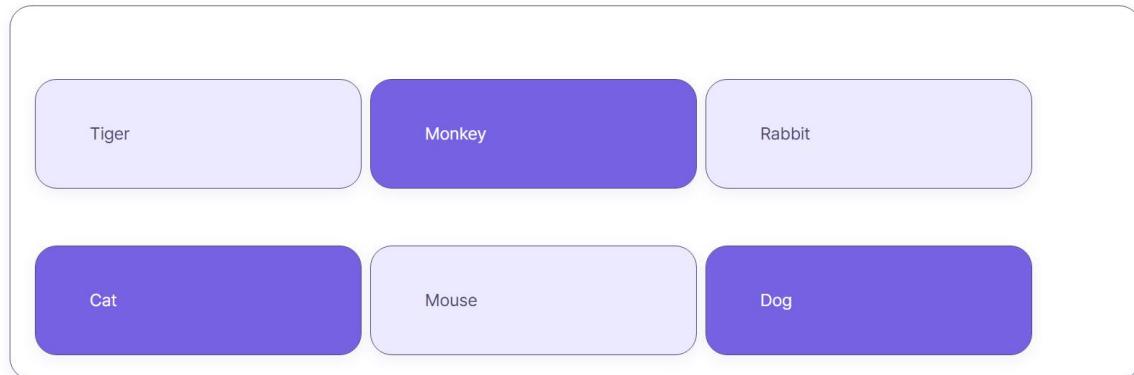
웹페이지 레이아웃을 구성할 때 부모 요소(컨테이너) 안의 자식 요소(아이템)들이 많으면 고민할 때가 있다. 예를 들어 영어 단어 웹페이지를 만든다고 가정해보자. 단어 카드들을 한 줄에 길게 배치할 것인지 아니면 한 줄이 아닌 두 줄 이상으로 배치할지 고민이 들 때가 있다.



no-wrap



wrap



wrap-reverse

```
.container {
  display: flex;
  /* flex-wrap: nowrap; */
  /* flex-wrap: wrap; */
  /* flex-wrap: wrap-reverse; */
}
```

```
<body>
  <div class="container">
    <div class="item">Cat</div>
    <div class="item even">Mouse</div>
    <div class="item">Dog</div>
    <div class="item even">Tiger</div>
    <div class="item">Monkey</div>
    <div class="item even">Rabbit</div>
  </div>
</body>
```

flex-wrap 속성을 사용하게 되면, 부모 요소(컨테이너)의 크기만큼 단어 카드들의 크기를 조절할 수도 있고, 단어 카드들의 크기만큼 영역을 차지하고 부모 요소(컨테이너)의 영역을 벗어나게 되면 단어 카드를 밑으로 내려서 정렬도 할 수 있게 된다.

flex-wrap 속성은 Flex 레이아웃 모듈의 하위 속성이다. 정렬된 요소들의 총 너비가 부모 너비보다 클 때, 요소들을 강제적으로 한 줄로 할 것인지 여러 줄로 할 것인지를 정의한다. 두 줄 이상인 경우 새로운 라인이 쌓일 방향을 결정하는 교차 축도 정의된다. 가로축은 메인축에 수직인 축이다. 부모 요소에 `display:flex;` 를 꼭 작성해야 한다.

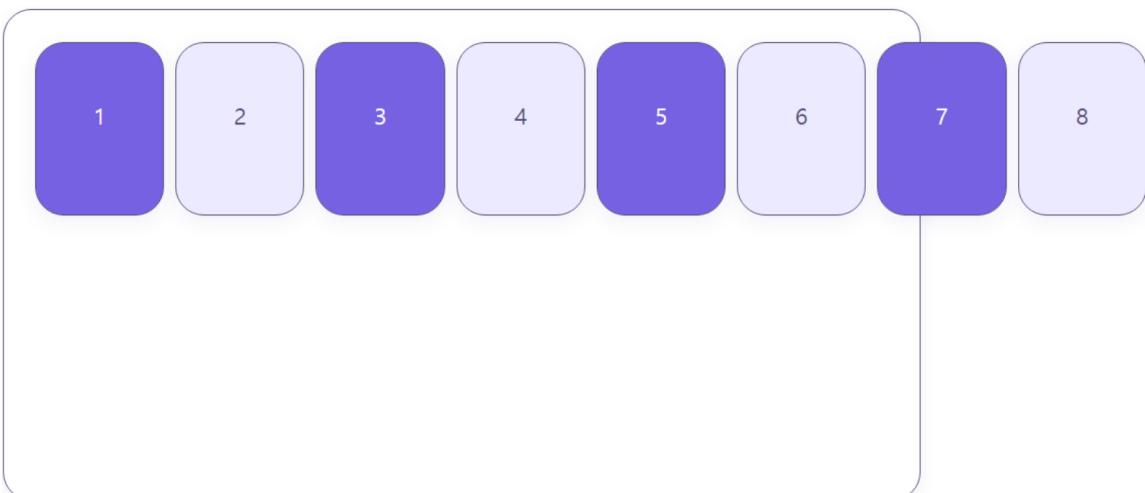
5.1. flex-wrap : nowrap

flex-wrap 속성 중 nowrap의 속성에 대해서 알아보자. flex-wrap는 작성자가 따로 설정하지 않는다면 기본값인 nowrap으로 설정된다. flex-wrap: nowrap을 사용하였을 때, 아래 그림과 같이 자식 요소(아이템)의 크기가 부모 요소(컨테이너)의 크기에 맞춰 일정 비율만큼 줄어드는 것을 확인할 수 있을 것이다.



하지만, 검색을 통해서 wrap의 속성을 찾아봤다면, 아래 그림처럼 부모영역(컨테이너)을 넘어가는 그림을 보았을 것이다. 이렇게 되는 이유는 8장에서 다루게 될 flex-shrink의 속성 때문이다. 기본적으로 flex-shrink의 초기값은 1로 설정되어 있어서 일정 비율만큼 줄어들게 되는데 이 속성을 0으로 하면 아래 그림처럼 부모영역(컨테이너)을 넘어가게 된다.

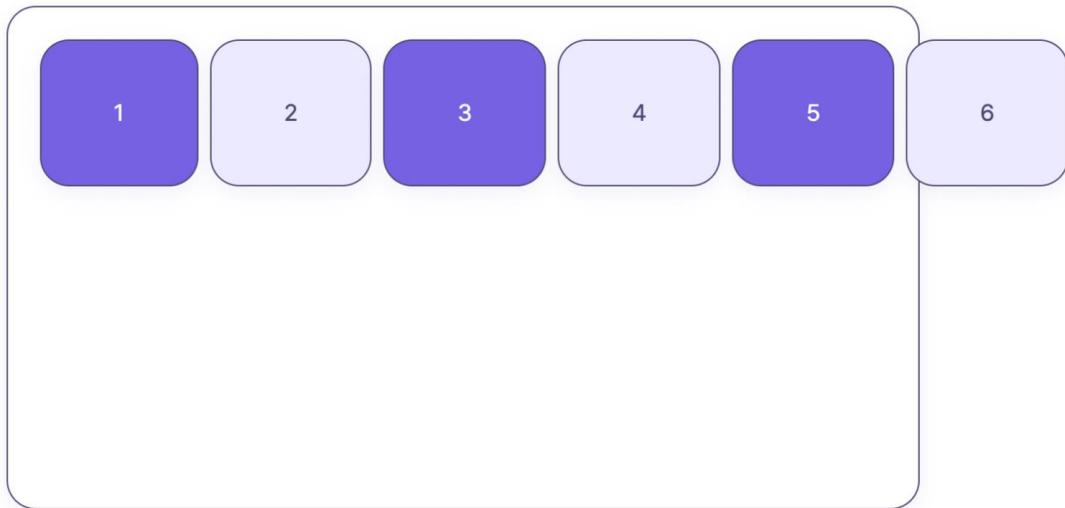
따라서 flex-wrap : nowrap은 자식 요소(아이템)의 전체 크기가 부모영역(컨테이너)을 넘어가지 않는다면 자기 자신의 크기를 유지하게 되고, 부모영역(컨테이너)을 넘어가게 된다면 일정한 비율로 줄어들게 된다. Flex를 사용하다 보면 nowrap이라는 속성을 자주 볼 일은 없지만, 이 속성을 사용하면 어떻게 변하는지 알아두는 것이 좋다.



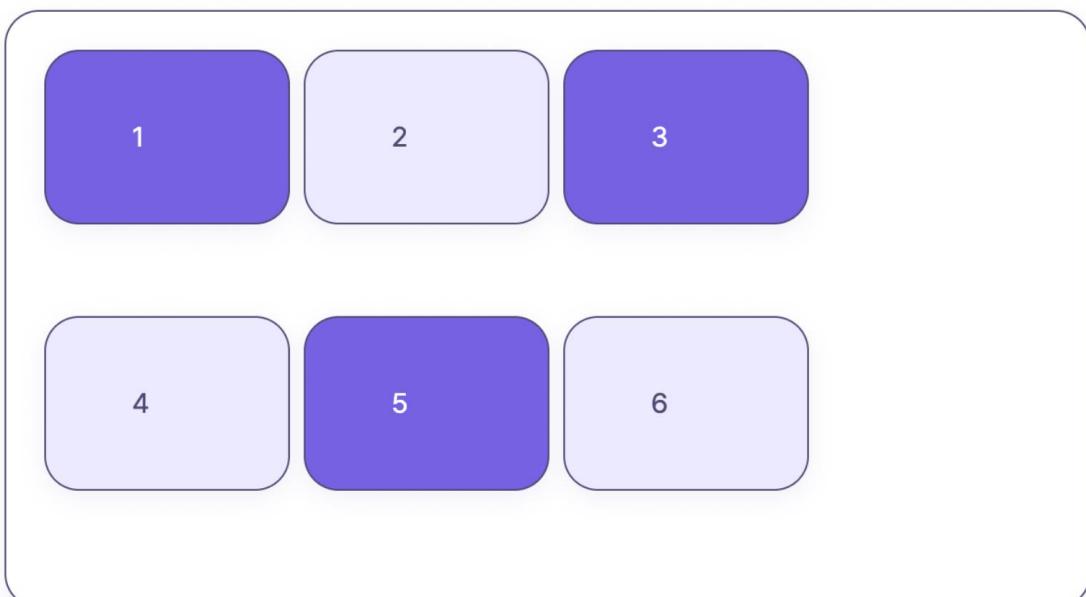
5.2. flex-wrap : wrap

다음으로 flex-wrap 속성 중에 wrap의 속성에 대해서 알아보자. Flex-wrap의 wrap속성은 각 자식 요소(아이템)의 총 넓이가 부모 요소(컨테이너)의 넓이 보다 클 때, 부모 요소(컨테이너) 안에 자식 요소(아이템)이 들어갈 수 있도록 다음 줄에 이어서 정렬해 주는 속성을 말한다.

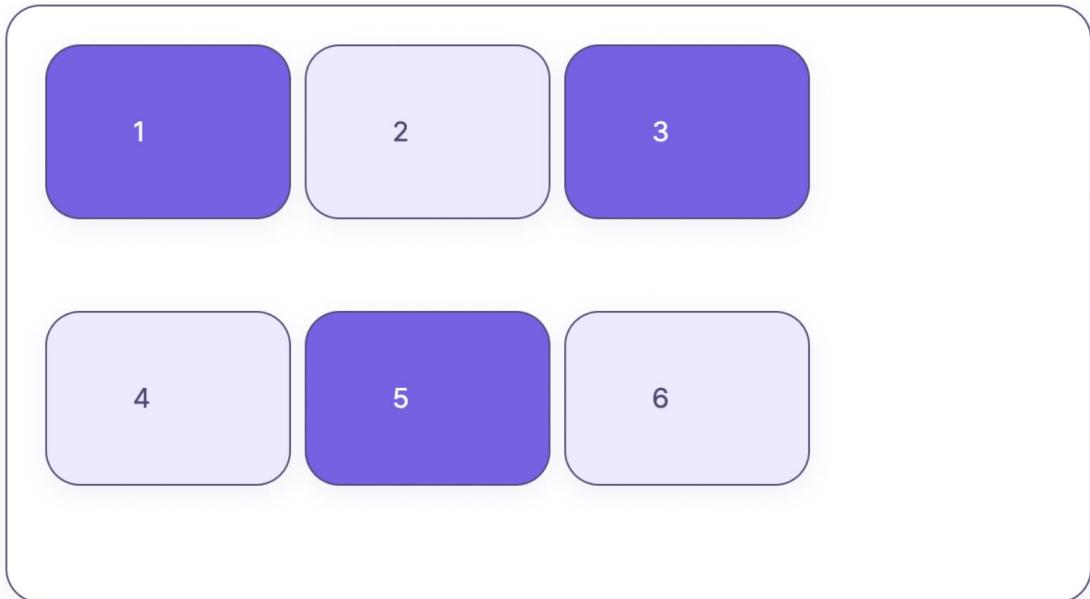
앞서 말한 flex-wrap에서 no-wrap을 사용하면 자식 요소(아이템)의 총 넓이가 부모 요소(컨테이너)의 넓이보다 큰 경우 정렬된 요소들은 부모 요소(컨테이너)의 넓이에 맞춰 자동 축소 되게 된다.



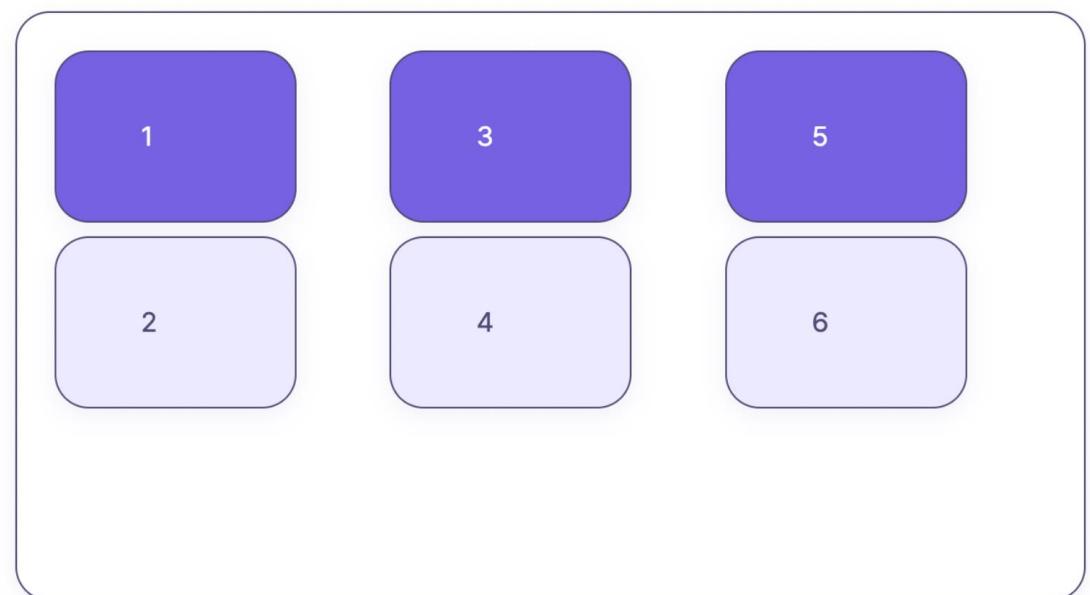
반면, flex-wrap:wrap의 경우에는 자식 요소(아이템)의 넓이를 유지한 채로 다음 줄로 이어서 정렬 되게 한다. flex-wrap:wrap를 사용하는 경우, 자식 요소(아이템)의 크기와 부모 요소(컨테이너)에 크기에 따라 다음 줄로 정렬이 이어지게 되는데 이는 flex-direction의 방향을 따르게 된다,



만약 flex-direction의 방향이 row(행)라면 row(행)에 맞춰 주축인 수평으로 자식 요소(아이템)들이 정렬 되게 되고, 교차축인 수직축으로 아이템이 쌓이게 된다.



flex-direction의 방향이 column(열)이라면 column(열)에 맞춰 주축인 수직으로 자식 요소(아이템)들이 정렬 되게 되고, 교차축인 수평축으로 자식 요소(아이템)줄이 쌓이게 된다.



이러한 자동 맞춤 같은 flex-wrap:wrap의 성질에 따라서 화면이 조정되는 경우(특히 미디어 쿼리를 구현하는 경우), 아이템의 크기가 변하지 않고 위치만 조정될 수 있게 하기 위해서 많이 사용하게 된다.

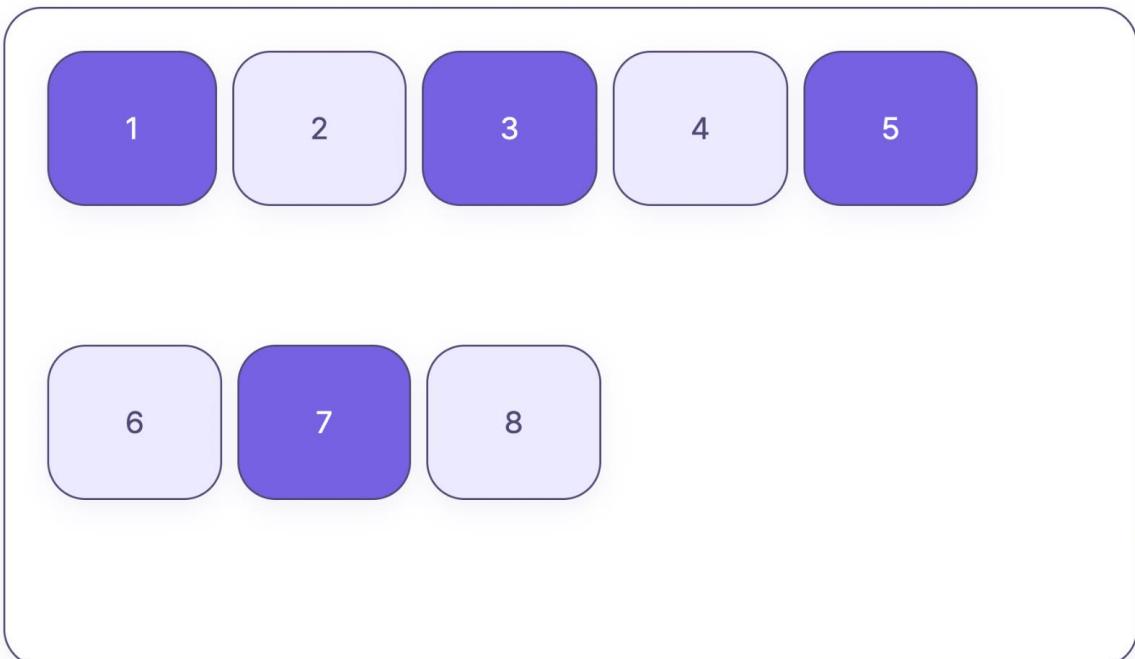
5.3. flex-wrap : wrap-reverse

`flex-wrap : wrap-reverse;` 는 `wrap-reverse : wrap;` 과 마찬가지로 자식 요소(아이템)들이 필요한 경우 여러 줄에 걸쳐 배치되지만, 다른 점은 wrap-reverse이기 때문에 자식 요소(아이템) 요소들이 정렬되는 기준점이 반대 방향으로 바뀌어서 배치된다.

적용하는 방법은 부모 요소(컨테이너)에 `display: flex`를 주고, `flex-wrap: wrap-reverse` 값을 설정하면 된다.

```
.container {
  display: flex;
  flex-wrap: wrap-reverse;
}
```

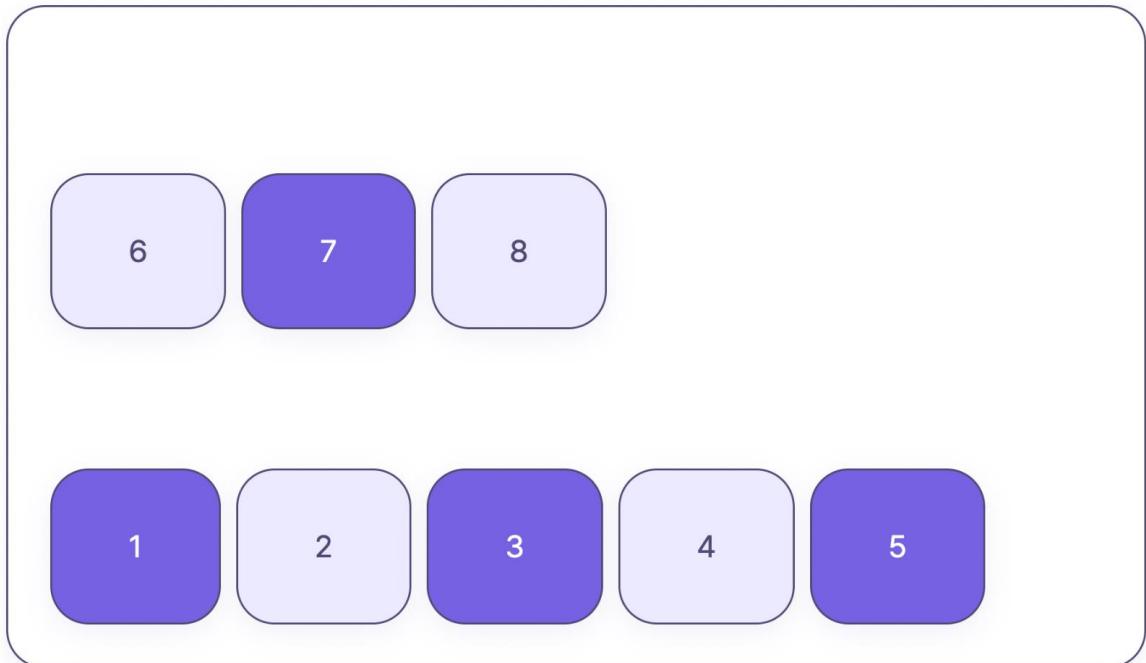
- `flex-wrap: wrap;` 과 `flex-wrap: wrap-reverse;` 비교



```
.container {
  display: flex;
  flex-wrap: wrap;
}
```

`display: flex;` 의 경우 `flex-direction: row;` 가 기본값이다. 그렇기 때문에 `flex-wrap: wrap;` 인 경우에는 부모 요소인 컨테이너의 왼쪽 축의 상단부터 차식 요소인 아이템들이 순차적으로 배치된다. 즉, 배치가 시작되는 기준이 주축(main-axis)의 main-start 지점이 부모(컨테이너)의 상단, 교차 축(cross-axis)의 cross-start 지점은 부모(컨테이너) 왼쪽이 된다.

하지만, `flex-wrap: wrap-reverse;` 의 경우 부모 요소(컨테이너)의 하단인 바닥 행의 왼쪽에서부터 1, 2, 3... 번 순으로 아이템이 배치 되고 나머지 요소들은 그 행 위에 배치 된다. 또, 전체적으로 보면 `flex-wrap: wrap;` 과는 반대로 차식 요소(아이템)들이 컨테이너의 하단에 불어 배치 되는 것을 확인 할 수 있다.

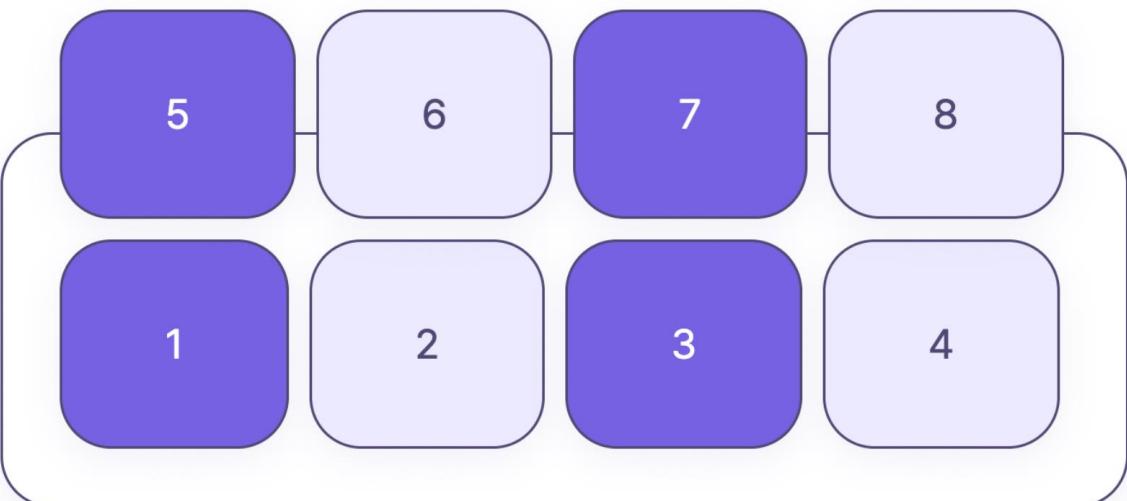


```
.container {
  display: flex;
  flex-wrap: wrap-reverse;
}
```

reverse(역순)이기 때문에 배치 되는 기준점이 바뀌기 때문이다. 즉, `flex-wrap: wrap-reverse;` 를 주게 되면, 주축(main-axis)의 main-start 지점이 부모 요소(컨테이너)의 하단이 된다. 그래서 부모 요소(컨테이너)의 하단 왼쪽이 기준점이 되어 차식 요소들(아이템들)이 순서대로 배치 되게 된다. 차식요소(아이템)는 부모 요소(컨테이너)의 width에 맞춰 정렬되기 때문에 부모의 width를 줄이면, 각 줄에 들어가는 아이템의 개수 역시 아래 예시와 같이 변경되게 된다.



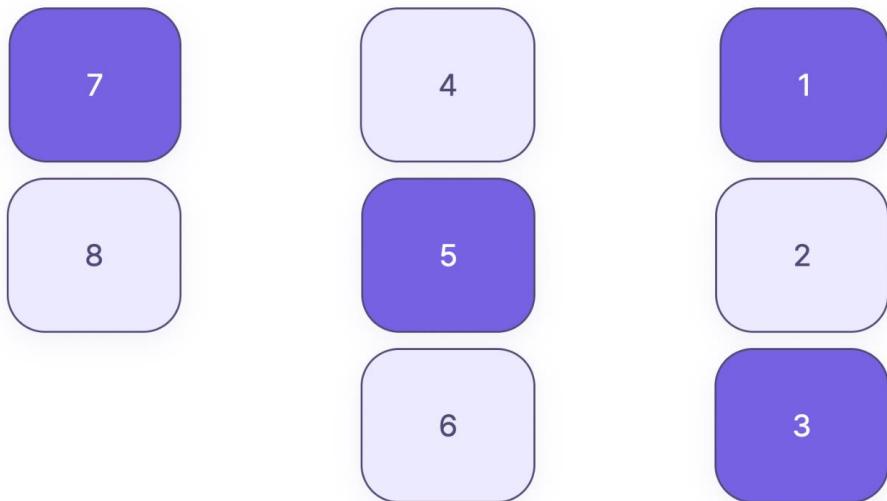
그리고 `flex-wrap: wrap;` 과 마찬가지로, `flex-item` 요소들의 크기를 컨테이너 크기에 맞춰 줄이지 않기 때문에, `flex-item`(자식 요소)이 부모(컨테이너)에 들어가기에 더 많은 부피를 차지하게 되면 아래 이미지처럼 아이템들이 컨테이너 밖으로 넘치는 것을 확인할 수 있다.



- flex-direction: column; 값을 준 경우 flex-wrap: wrap-reverse; 는 어떻게 될까?

`display: flex;` 를 준 경우, `flex-direction`의 기본값은 row지만, `flex-direction: column;` 을 준다면 위에서 확인한 것들과는 다른 결과를 확인할 수 있다. flex의 방향이 row(행)이 아닌 column(열)으로 변경되기 때문에 주축(main-axis)이 수평(가로)이 아닌 수직(세로) 방향이 된다.

그렇기 때문에, `flex-direction: column;` 인 상태에서 `flex-wrap: wrap-reverse;` 를 주게 되면, 주축(main-axis)의 main-start 지점이 부모 요소(컨테이너)의 오른쪽 끝 지점이 되고, 오른쪽 끝의 열(column)부터 아이템이 차례대로 배치된다.



6. Flex-basis

6.1. flex-basis란?

flex-basis는 자식 요소(아이템)에 사용하는 속성이다.

```
.item {  
    flex-basis: auto; /* 기본값 */  
}
```

부모 요소인 컨테이너의 주 축(main axis)이 되는 방향으로 **flex 아이템의 초기 크기(default size)**를 정해줄 수 있다. 컨테이너의 아이템 배치 방향(**flex-direction**)이 가로(row) 축일 경우 넓이를, 세로(column) 축일 경우 높이를 지정한다.

6.2. flex-basis에 들어가는 값

flex-basis의 값으로는 우리가 사용하는 각종 단위가 들어갈 수 있다. 길이를 나타내는 단위들인 **em**, **rem**, **px**이나 퍼센티지(**%**) 값이 될 수도 있고, **content**, **min-content**, **max-content**, **fit-content**, **fill**, **auto** 등의 키워드가 될 수도 있다. 상수는 0 외에 다른 값을 사용할 수 없다.

6.3. flex-basis의 유연성

flex-basis 속성은 **유연한(flexible)** 크기를 가진다. 즉, 고정적인 width, height 값을 지정해 줄 때와 달리 축의 방향에 따라, 또 내부 콘텐츠에 따라 크기가 가변적, 유동적으로 변할 수 있다.

**Lorem ipsum dolor sit amet consectetur
adipisicing elit. Modi nulla dolores optio
dolorum earum esse incidunt obcaecati deleniti.**

**Lorem ipsum dolor sit amet consectetur,
adipisicing elit. Incidunt aliquid eum suscipit
quis quaerat. Natus.**

Lorem.

아이템이 `height: 40px;` 일 때(내부 콘텐츠의 크기는 고려하지 않고 40px에 맞춤)

Lore*m ipsum dolor sit amet consectetur adipisicing elit. Modi nulla dolores optio dolorum earum esse incidunt obcaecati deleniti itaque. Facilis quia voluptatum recusandae itaque labore.*

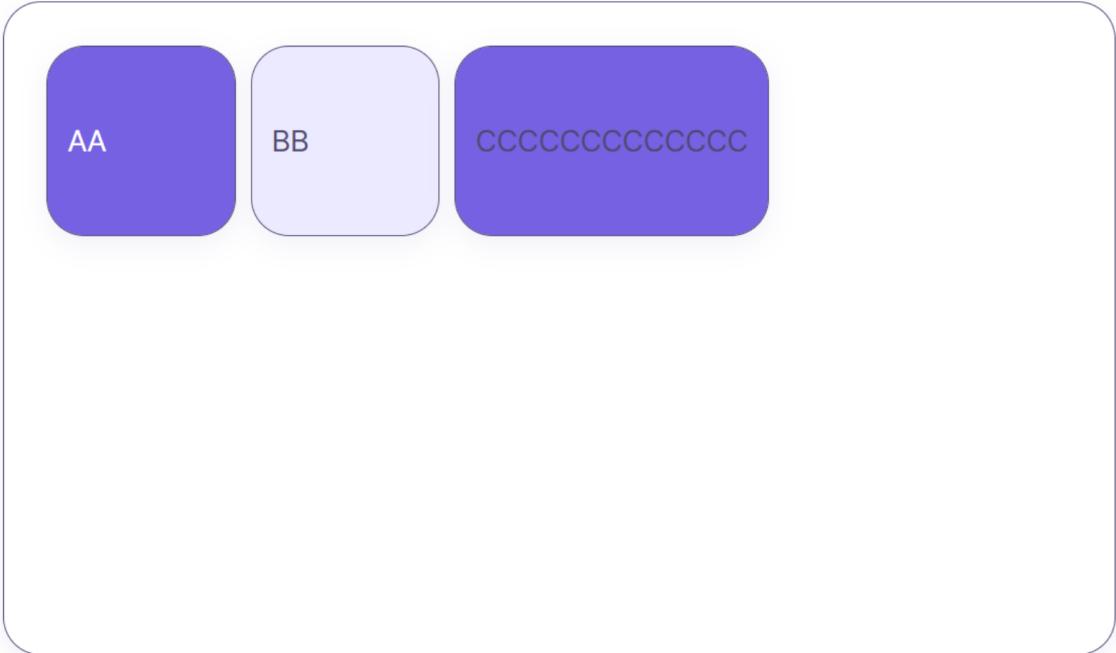
Lore*m ipsum dolor sit amet consectetur, adipisicing elit. Incidunt aliquid eum suscipit quis quaerat. Natus.*

Lore*m.*

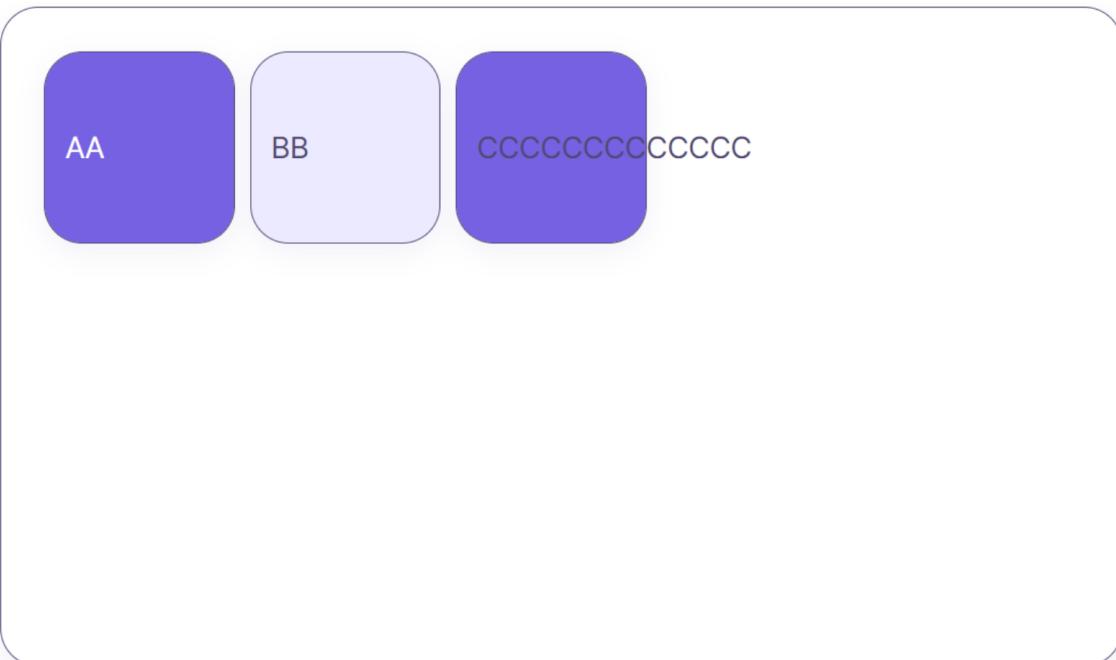
아이템이 `flex-basis: 40px;` 일 때(콘텐츠의 크기에 맞춤)

6.4. flex-basis와 width/height의 관계

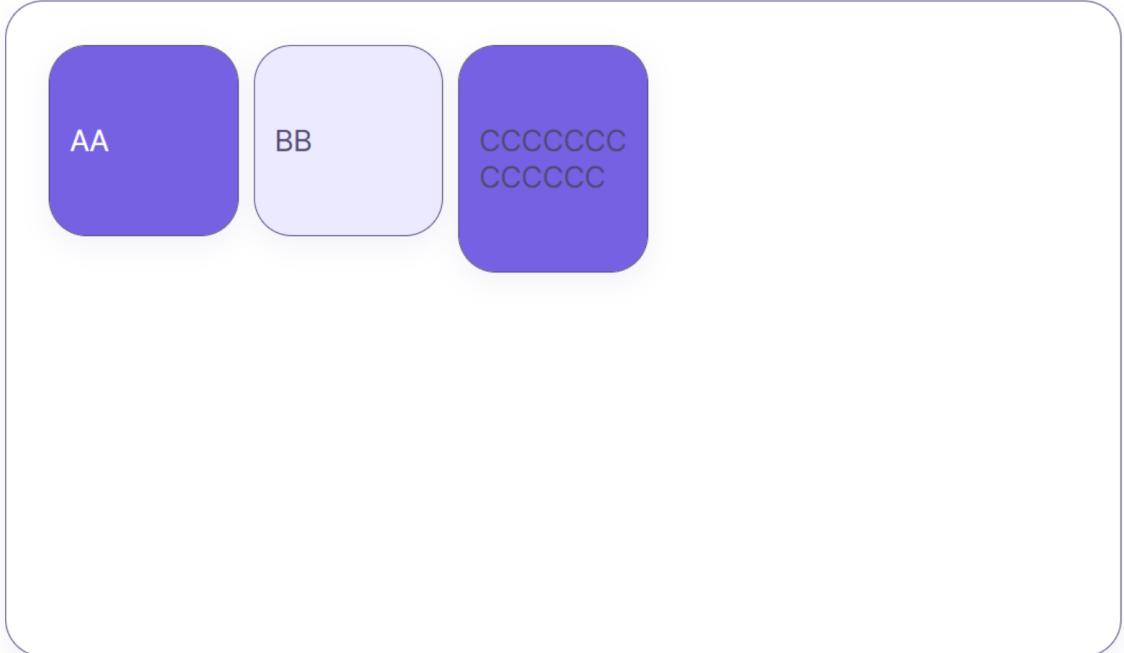
기본값 auto는 해당 아이템의 width(또는 height) 값을 사용하게 된다. 만약 width값이 없다면 콘텐츠의 크기가 기본으로 잡히게 된다.



위 그림에서는 basis의 값을 100px로 설정했다. 콘텐츠의 크기가 적으면 100px로 아이템들의 width가 결정되지만, 콘텐츠가 많아지게 되면 basis보다 넓어지게 된다.



반면, width값을 100px로 설정하게 되면 basis와 달리 컨텐츠의 크기가 많아지면 아이템의 width는 100px로 고정되고, 컨텐츠가 밖으로 나가게 된다. 이를 방지하려면 `word-wrap: break-word;` 값을 주면 해결된다.



기본적으로 flex-basis 속성이 width(또는 height) 값보다 우선하게 된다. 즉, width를 100px를 주고 basis를 300px로 주게 되면 basis값이 우선시되어 아이템들의 기본너비가 300px로 잡히게 된다. 주의 할 점은 서로 다른 값일 경우 기본적으로 flex-basis가 우선하지만, 똑같은 값을 flex-basis와 width(또는 height)에 주고 동시에 적용할 경우엔 width(또는 height)의 적용이 우선시된다는 것이다. 또한, 언어가 한글인지 영어인지와 `word-break` 등 다른 속성값에 따라 flex-basis는 예상과 다르게 동작할 수도 있다.

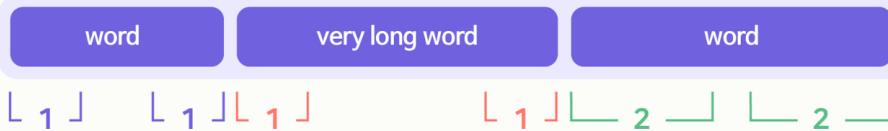
6.5. flex-basis : auto; 와 flex-basis : 0;

`flex-basis: auto;` 가 기본값이다. 이때는 지정해 준 width(또는 height) 값을 사용하거나, 다른 박스가 늘어날 때 같이 늘어난다(stretch). 또한, 추가 공간이 flex-grow 값에 따라 분배된다. `auto` 일 때와는 달리 `flex-basis: 0;` 으로 설정하면 내용 주위의 추가 공간이 고려되지 않는다.

flex-basis : 0



flex-basis : auto



flex-basis 값이 0일 때와 auto일 때의 차이

6.6. flex-basis : content;

`flex-basis: content;` 는 콘텐츠 크기에 맞게 자동으로 크기가 조절된다. 고유 크기 조정 키워드로는 `fill`, `min-content`, `max-content`, `fit-content` 가 있다.

auto	max-content	min-content	fit-content
1. flex-basis test	2. flex-basis test	3. flex-basis test	4. flex-basis test
content		fill	
5. flex-basis test		6. flex-basis test	

이 키워드는 잘 지원되지 않기 때문에 테스트하기 어렵다 (2022년 기준). 따라서 `min-content`, `max-content` 및 `fit-content` 의 기능을 완전히 파악하여 원하는 의도대로 사용하기엔 아직 무리가 있다.

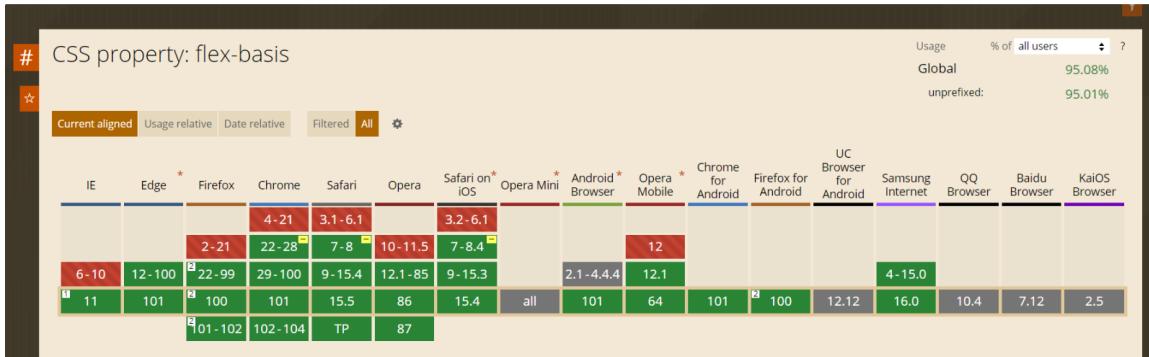
6.7. flex-basis보다는 flex 축약 속성을! (W3C)

CSS 표준을 관리하는 W3C에 따르면, flex-basis 속성을 직접 사용하기보다는 flex 축약 속성으로 사용하는 것을 권장하고 있다.

Authors are encouraged to control flexibility using the '`flex`' shorthand rather than with '`flex-basis`' directly, as the shorthand correctly resets any unspecified components to accommodate common uses.

출처: <https://www.w3.org/TR/css-flexbox-1/#flex-basis-property>

6.8. flex-basis 호환성



출처: <https://caniuse.com/?search=flex-basis>

7. Flex-grow

7.1. flex-grow란?

`flex-grow` 는 자식 요소인 `flex-item`이 차지하는 비율을 조절하는 속성이다. 컨테이너 크기에 맞춰 늘어나는 비율이 여백을 차지하면서 이 속성을 가진 다른 요소들과 동일하게 분배한다. 쉽게 말해서 `flex-grow`의 값은 아이템들의 `flex-basis`를 제외한 여백 부분을 `flex-grow`의 비율로 각각 나누어 가진다.

7.2. flex-grow 기능 : 확장 또는 고정

```
.item {
    flex-shrink: 0; /* 기본값 */
}
```

```
/* <number> values */
flex-grow: 1 | 0.5;

/* Global values */
flex-grow: inherit | initial | unset;
```

출처: MDN

`flex-grow` 속성은 기본값이 0 이다. `flex-grow:0` 일때는 `flex item`을 확장하지 않고 원래의 크기를 유지한다.



flex-grow: 0;

예시1. `flex-grow:0` (기본 값)

`flex-grow:1` 일 때는 `flex item`이 유연한 박스의 형태로 바뀌며 빈 공간을 채운다.



flex-grow: 1;

예시2. `flex-grow:1`

각각의 아이템별로 `flex-grow` 값을 줄 수도 있다. `flex-grow`에 정해준 비율만큼 여분의 컨테이너비를 분배하여 갖는다.

`flex-grow: 1;`

`flex-grow: 2;`

`flex-grow: 1;`

예시3. 각각의 아이템별로 다른 `flex-grow`

7.3. flex-grow 사용 시 주의할 점

`flex-grow` 속성은 0 또는 양의 정수의 값으로 설정해야 한다. 음수의 값으로 설정했을 경우, 기본값인 `flex-grow: 0` 일 때와 같다. 또한 `flex-grow`의 증가너비 비율은 width로 인해 영향을 받는다. 때문에 `flex-basis`로 flex의 가변 범위에 입력하여 기본값을 정하는 것이 좋다.

`flex-grow: 0;`

`flex-grow: 0;`

`flex-grow: -1;`

예시4. `flex-grow: -1`

7.4. flex 축약 속성

```
flex: flex-grow | flex-shrink | flex-basis;
```

일반적으로는 모든 값이 설정되었음을 보장하기 위하여 flex 속성을 이용해 **축약형**으로 사용한다. 축약 속성의 순서는 `flex-grow`, `flex-shrink`, `flex-basis` 순으로 적용되며 띄어쓰기로 구분한다.

Flex **축약 속성의 기본값**은 `flex: 0 1 auto;`이다. `flex-grow`를 제외한 개별 속성은 생략할 수 있다. `flex: 1;` 할 경우 grow는 1로 변하고 shrink의 값은 0으로 basis의 값을 명시적으로 작성하지 않으면 0으로 입력이 된다.

7.5. flex-shrink보다는 flex 축약 속성을! (W3C)

CSS 표준을 관리하는 W3C에 따르면, flex-shrink속성을 직접 사용하기보다는 flex 축약 속성으로 사용하는 것을 권장하고 있다.

Authors are encouraged to control flexibility using the '`flex`' shorthand rather than with '`flex-grow`' directly, as the shorthand correctly resets any unspecified components to accommodate [common uses](#).

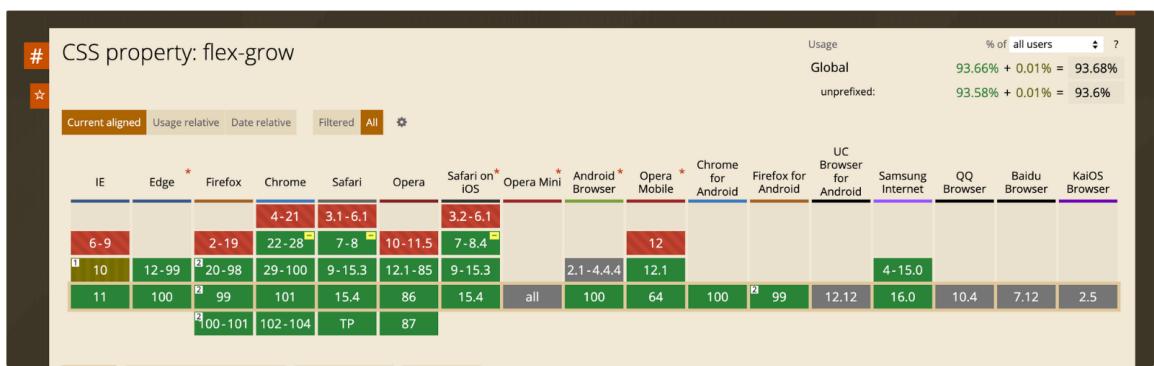
<https://www.w3.org/TR/css-flexbox-1/#propdef-flex-grow>

7.6. flex-grow 호환성

지원버전

	□										■		
	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	
flex-grow	✓ 29	✓ 12	✓ 20	✓ 11	✓ 12.1	✓ 9	✓ 4.4	✓ 29	✓ 20	✓ 12.1	✓ 9	✓ 2.0	
<0 animate⚠	✓ 49	✓ 79	✓ 32	✗ No	✓ 36	✗ No	✓ 49	✓ 49	✓ 32	✓ 36	✗ No	✓ 5.0	

출처: [mdn](#)



출처: <https://caniuse.com/?search=flex-grow>

8. Flex-shrink

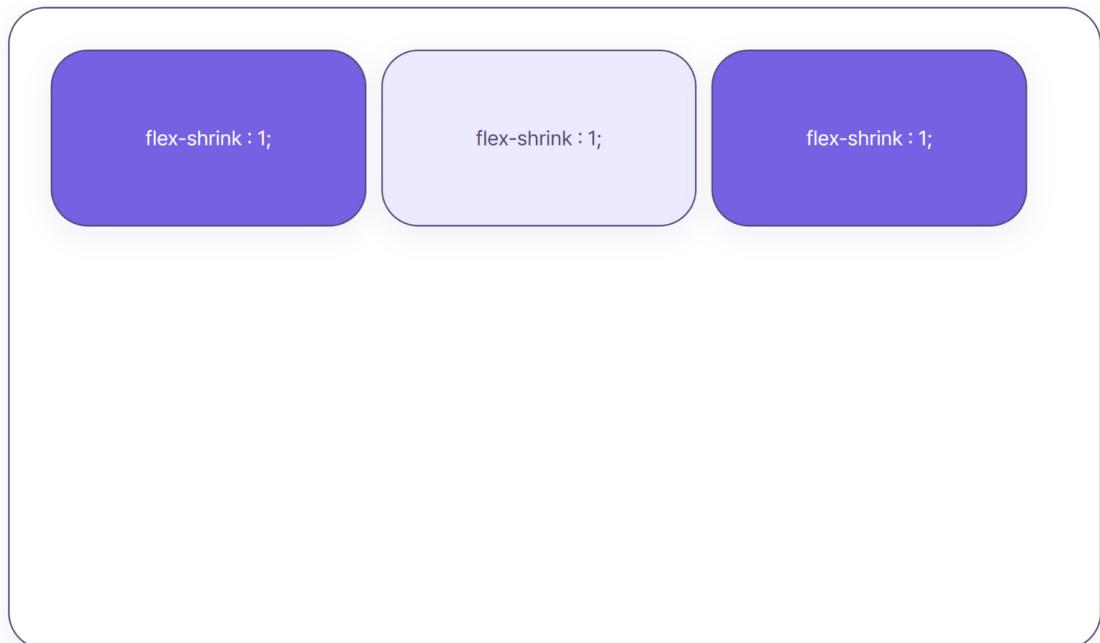
8.1. Flex-shrink 란?

`flex-shrink` 는 자식 요소에 사용하는 속성이다. flex 컨테이너 안에서 자식 요소인 flex 아이템 요소를 자동으로 줄여서 적절한 크기로 배치해 유연한 레이아웃을 만들 수 있다는 장점이 있다. `flex-basis` 속성으로 지정된 아이템의 기본 크기를 설정한 숫자 값에 비례해 수축시킬 수 있다. `flex-grow` 와는 반대되는 개념이다.

8.2. Flex-shrink 기능: 축소 또는 고정

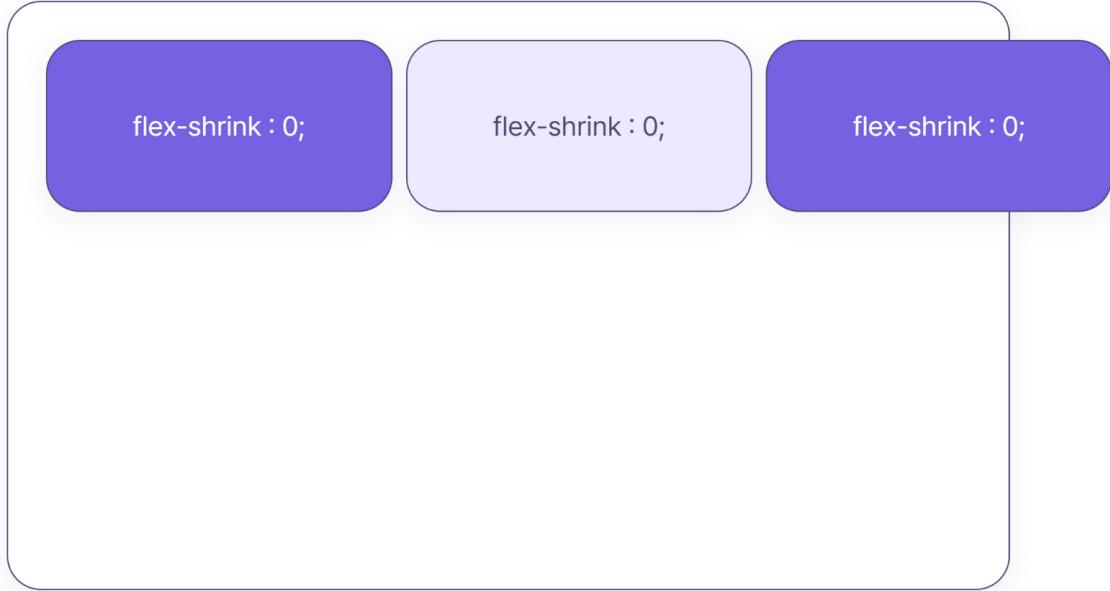
```
.item {  
    flex-shrink: 1; /* 기본값 */  
}
```

`flex-shrink` 속성의 **기본값은 1**이다. 즉, 정의하지 않아도 자동으로 축소된다. 숫자가 클수록 상대적으로 더 작은 크기를 갖게 된다. 값을 0으로 선언할 경우 너비 혹은 높이를 **고정해서** 항상 유지할 수 있다.

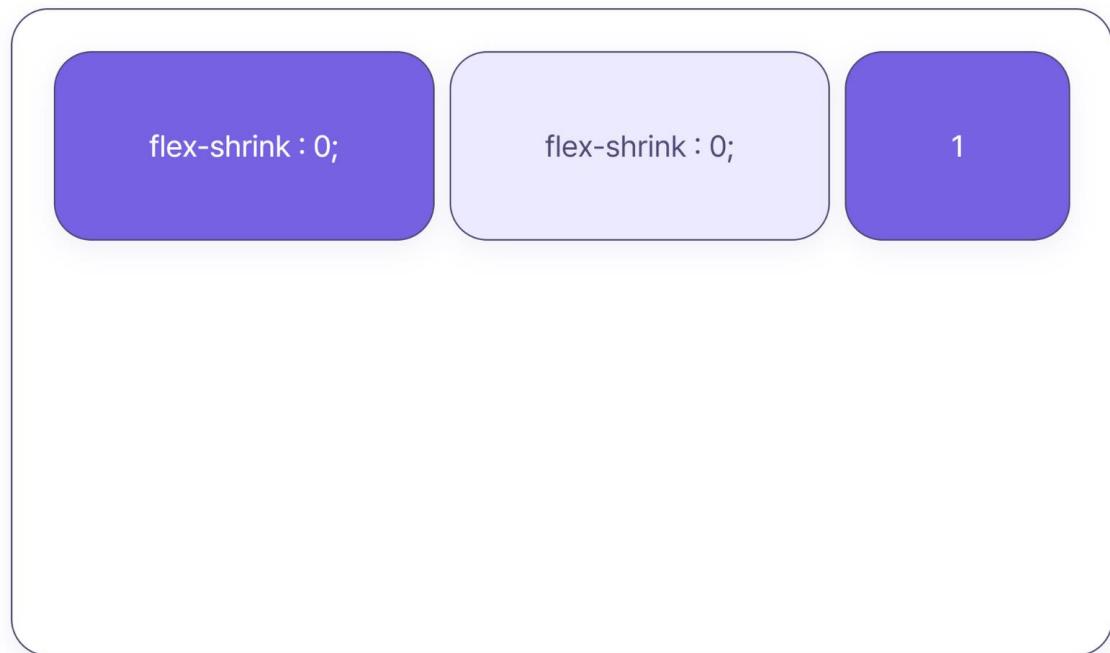


예시1. `flex-shrink:1` (기본값)

Flex_8 Flex-shrink



예시2. `flex-shrink:0`

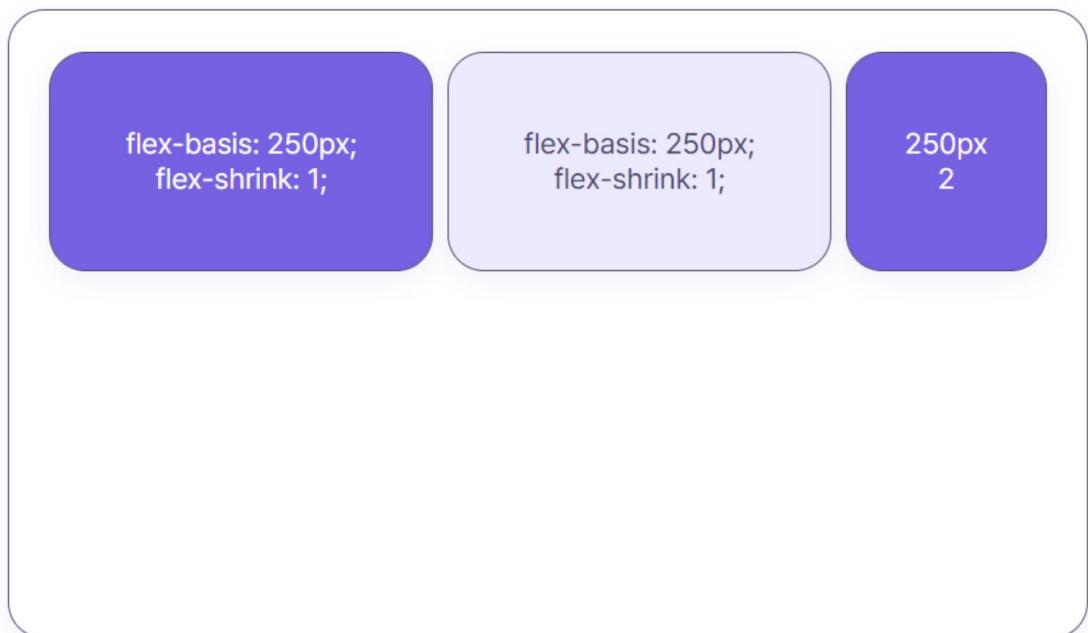


예시3. 아이템 별로 다른 경우

8.3. Flex-shrink 사용 시 주의할 점

`flex-shrink` 속성은 부모 컨테이너에 `flex-wrap: wrap;` 속성을 부여한 경우 적용되지 않는다. 따라서 적용을 위해서는 `wrap` 을 정의하지 않거나, `nowrap` 속성이 부여되어야 한다.

 flex-basis: 250px 통일, 3번째 아이템만 flex-shrink: 2 일 때 (예시3, 4)



예시3. 부모 컨테이너 `flex-wrap: nowrap;`



예시4. 부모 컨테이너 `flex-wrap: wrap;` (\Rightarrow flex-shrink 적용되지 않음)

8.4. Flex 축약 속성

```
flex: <flex-grow> <flex-shrink> <flex-basis>
```

`flex-grow`, `flex-shrink`, `flex-basis` 속성의 값을 단축해서 사용할 수 있는 것이 `flex` 축약 속성이다. 순서는 grow, shrink, basis 순으로 적용된다.

- `flex-grow` 는 flex 아이템이 팽창하는 비율을 설정한다. 기본값은 1이다.
- `flex-shrink` 는 flex 아이템이 수축하는 비율을 설정한다. 기본값은 1이다.
- `flex-basis` 는 flex 아이템이 팽창하고 수축하는 기준 크기를 설정한다. 기본값은 0이다.

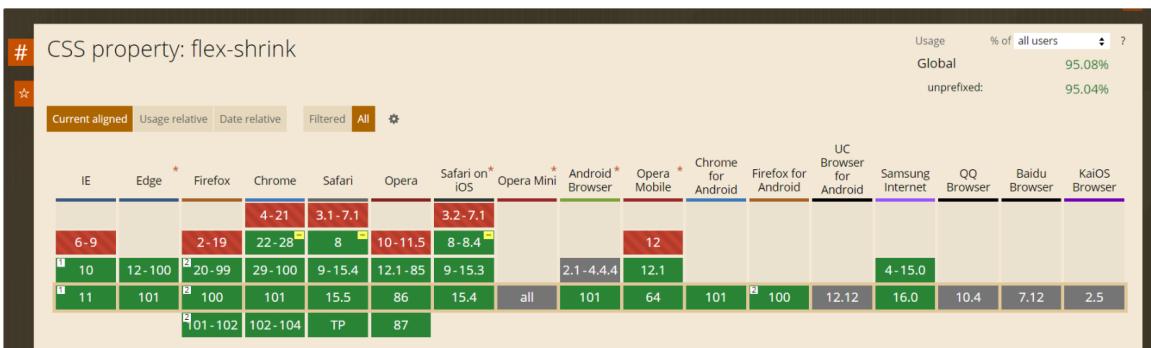
8.5. Flex-shrink보다는 Flex 축약 속성을! (W3C)

CSS 표준을 관리하는 W3C에 따르면, flex-shrink 속성을 직접 사용하기보다는 `flex` 축약 속성으로 사용하는 것을 권장하고 있다.

Authors are encouraged to control flexibility using the '`flex`' shorthand rather than with '`flex-shrink`' directly, as the shorthand correctly resets any unspecified components to accommodate [common uses](#).

<https://www.w3.org/TR/css-flexbox-1/#propdef-flex-shrink>

8.6. Flex-shrink 호환성 (caniuse)



출처: <https://caniuse.com/?search=flex-shrink>

9. 그 밖의 Flex-item에 사용하는 속성들

기타 flex-item에 사용하는 속성에는 `align-self`, `order`, 그리고 `z-index`가 있다.

9.1. align-self

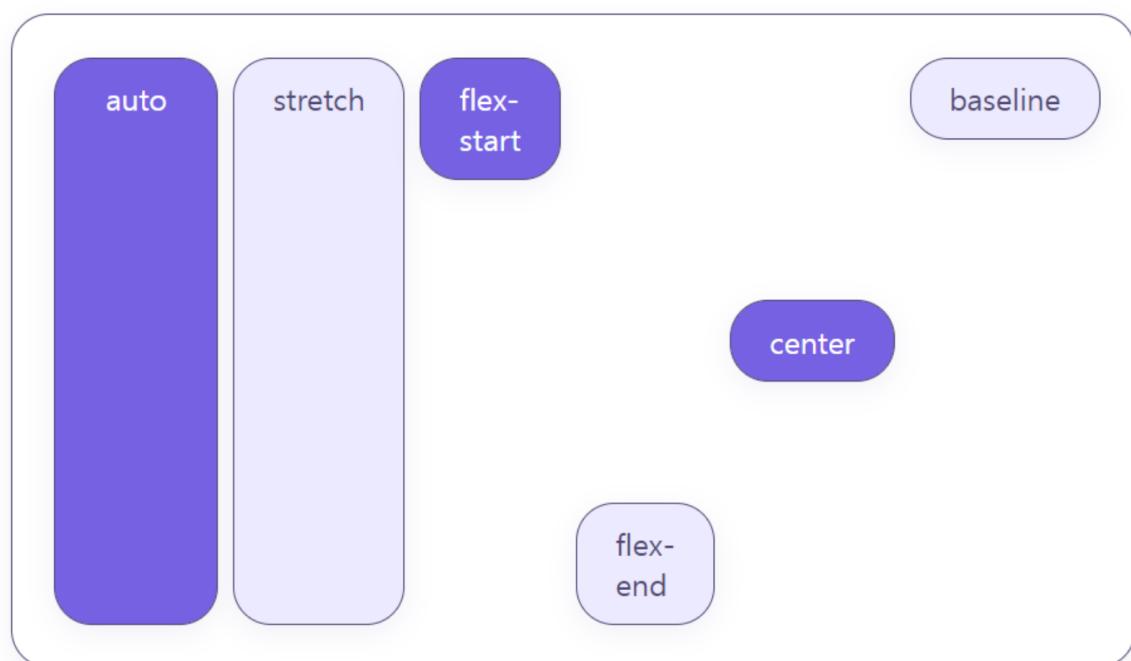
`align-self` 란 교차 축(cross-axis)을 기준으로 개별아이템 요소의 정렬 방법을 결정하는 속성이다. Flex 컨테이너에 적용하여 내부에 있는 전체 아이템을 정렬하는 `align-items` 와 달리, `align-self`는 인접한 Flex 아이템에 영향을 주지 않고, 각각의 Flex 아이템 위치를 자유롭게 변경할 수 있다. 따라서 전체 설정인 `align-items` 보다, 개별 아이템에 적용되는 `align-self` 속성이 우선한다는 특징이 있다.

align-self의 속성값

`align-self`의 속성값은, auto 값을 제외하면 `align-items`의 속성값과 같다.

- auto : 기본값으로, 플렉스박스 컨테이너의 `align-items` 속성을 상속받는다. `align-items`의 기본 속성은 stretch이기 때문에, `align-items`가 지정되어있지 않는 경우 stretch 속성을 상속받는다.
- stretch : 교차 축을 기준으로, 컨테이너의 높이를 채우기 위해 플렉스 아이템이 늘어난다.
- flex-start : 교차 축을 기준으로, 플렉스 아이템이 컨테이너의 시작점에 정렬된다.
- flex-end : 교차 축을 기준으로, 플렉스 아이템이 컨테이너의 끝점에 정렬된다.
- center : 교차 축을 기준으로, 플렉스 아이템이 컨테이너의 중앙에 정렬된다.
- baseline : 교차 축을 기준으로, 플렉스 아이템이 컨테이너의 문자 기준선에 정렬된다.

아래는 아이템에 auto, stretch, flex-start, flex-end, center, baseline 속성값을 부여한 모습이다. 컨테이너가 `flex-direction:row;` 인 경우 교차 축인 세로축을 기준으로, 아이템들이 컨테이너를 채운다.



Flex_9 그 밖의 Flex-item에 사용하는 속성들

```
.container{  
    display: flex;  
    flex-direction: row;  
}  
.item:nth-child(1){align-self: auto;}  
.item:nth-child(2){align-self: stretch;}  
.item:nth-child(3){align-self: flex-start;}  
.item:nth-child(4){align-self: flex-end;}  
.item:nth-child(5){align-self: center;}  
.item:nth-child(6){align-self: baseline;}
```

`flex-direction:column;` 일 때는 교차 축인 가로축을 기준으로, 아이템들이 컨테이너를 채운다.

auto

stretch

flex-start

flex-end

center

baseline

```
.container{  
    display: flex;  
    flex-direction: column;  
}
```

align-self의 활용 예시

아래는 Flex 컨테이너의 아이템들을 align-self 속성을 사용하여 간단하고 쉽게 정렬한 모습이다. 컨테이너에 `flex-direction:column;` 을 부여하고, 첫 번째 Flex 아이템에 `align-self:flex-end;` 를, 네 번째 Flex 아이템에 `align-self:flex-start;` 을 주어 배치하였다. 값이 부여되지 않은 두 번째와 세 번째 아이템 요소는 `align-self:auto;` 가 되어, 컨테이너의 `align-items` 의 기본값인 stretch 속성을 부여받은 것을 확인할 수 있다.



```
.container{  
    display: flex;  
    flex-direction: column;  
}  
.item:nth-child(1){align-self: flex-end;}  
.item:nth-child(4){align-self: flex-start;}
```

9.2. order

`order` 속성은 아이템 요소들의 순서를 결정하는 속성이다. 기본적으로 `flex` 는 작성한 순서대로 나열되지만, `order` 속성을 사용하여 아이템들의 순서를 변경할 수 있다.

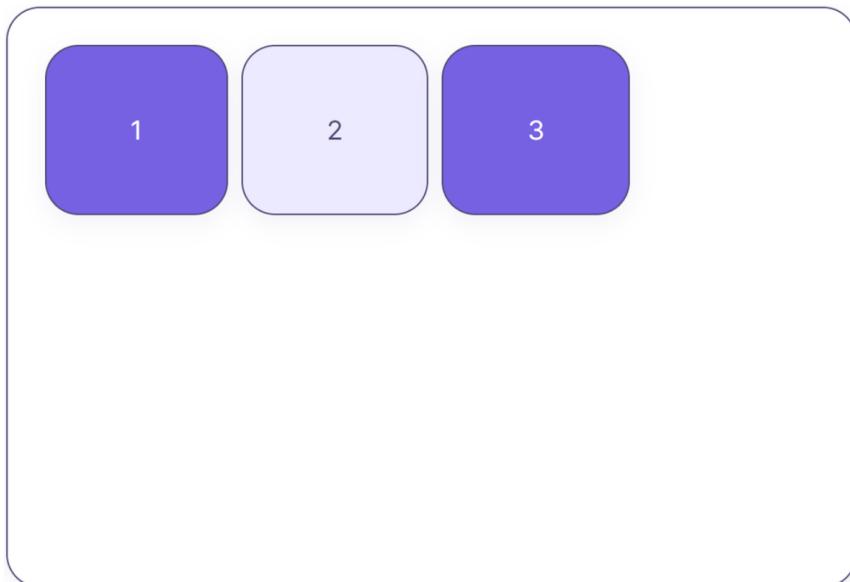
order의 특징

- 기본값은 0이며, 음수와 양수를 사용할 수 있다. 값이 작을수록 우선순위가 적용되어 `음수 → 0 → 양수` 순서로 표시된다.
- `flex-direction` 속성의 방향값(row, row-reverse, column, column-reverse)을 기준으로 낮은 숫자를 먼저 배치한다.
- HTML 구조와 상관없이 순서를 시각적으로 변경할 수 있지만, HTML 자체의 구조를 바꾸는 것은 아니다. 따라서 스크린리더가 읽을 때는 `order` 값이 적용되지 않고 HTML 마크업 순서대로 읽힌다.

아래는 `flex-direction: row;` 와 `flex-direction: column;` 에서 `order` 값을 적용하지 않았을 때와 적용했을 때의 모습이다.

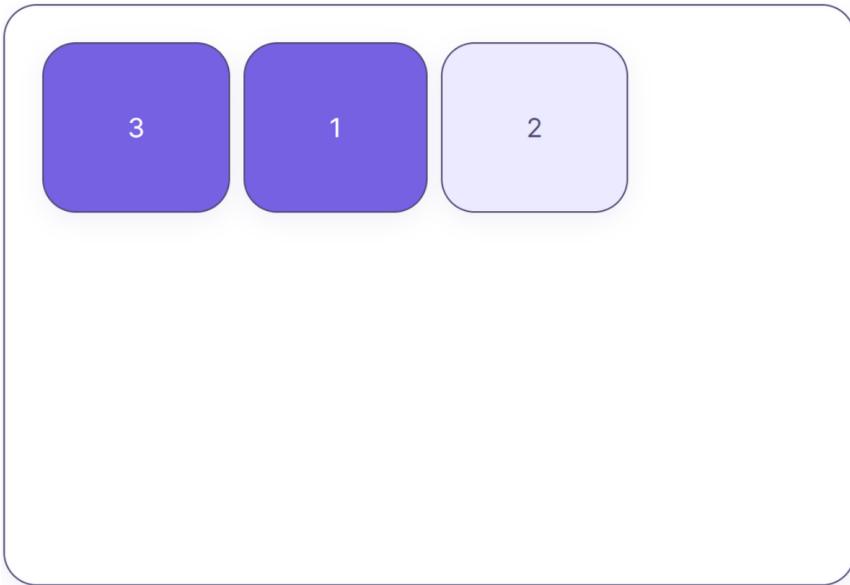
`flex-direction: row;`

- order 값을 적용하지 않은 모습



Flex_9 그 밖의 Flex-item에 사용하는 속성들

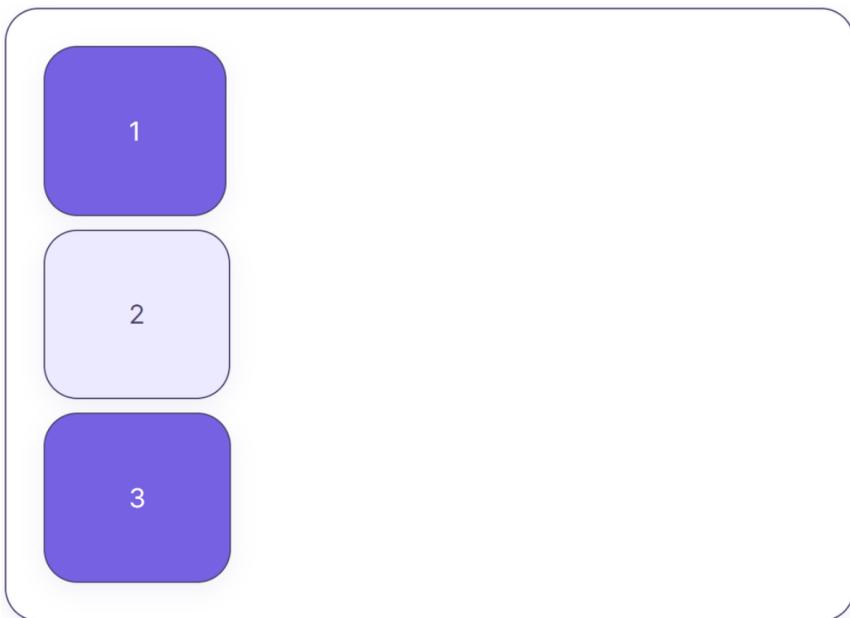
- order값을 적용한 모습



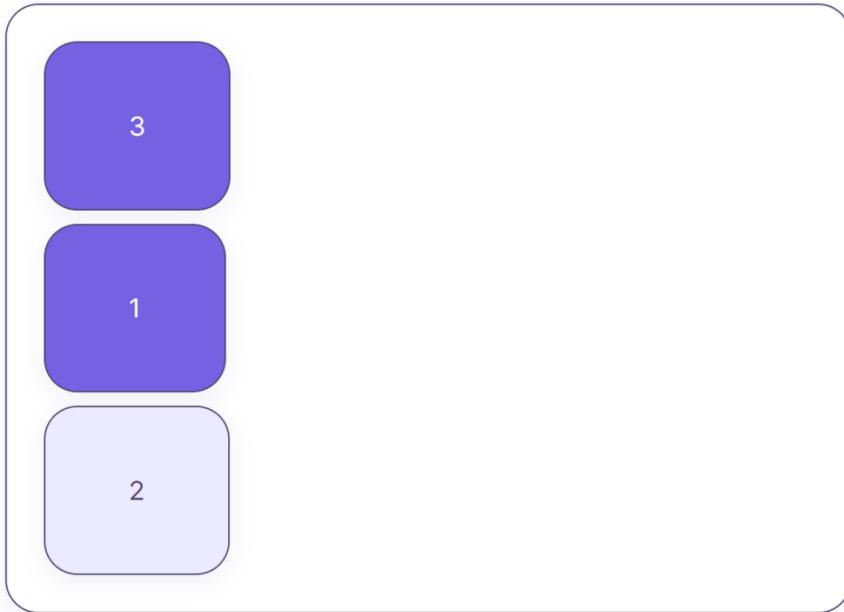
```
.item:nth-child(1) {order: 2;}  
.item:nth-child(2) {order: 3;}  
.item:nth-child(3) {order: 1;}
```

`flex-direction: row;`

- order 값 적용하지 않은 모습

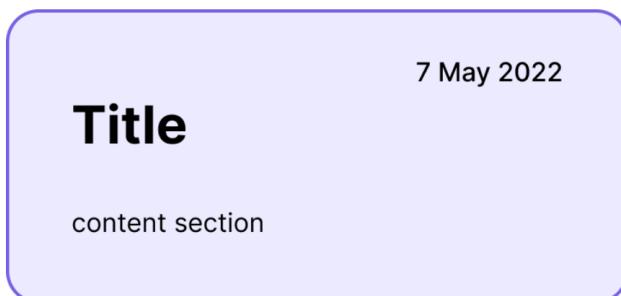


- order값을 적용한 모습



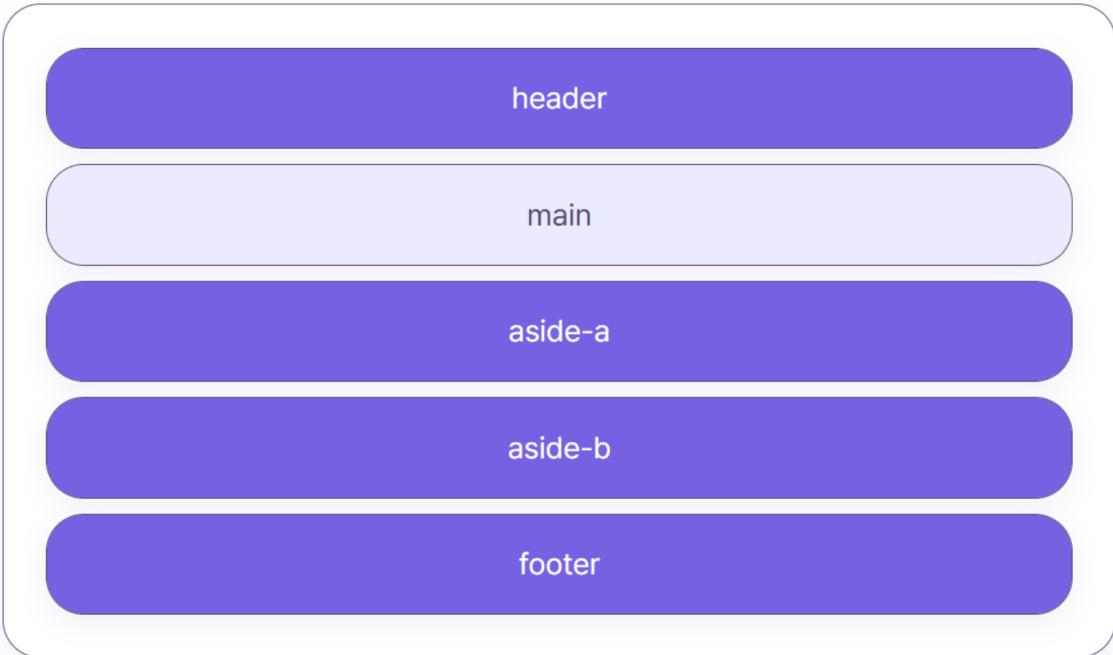
order의 사용 예시

1. 날짜와 제목, 내용이 있는 카드 디자인을 만든다고 가정하자. 아래의 이미지처럼 날짜가 제목보다 먼저 위치하게 할 때, 기본 순서로 배치한다면 스크린 리더는 날짜, 제목, 내용 순으로 읽을 것이다. 하지만 사용자는 가장 중요한 제목을 먼저 읽은 후 날짜를 읽기를 선호한다. 이럴 경우, 날짜에 `order:-1` 을 주어 시각적 순서만 바꿔서 스크린 리더가 읽는 순서를 변경하지 않도록 할 수 있다.



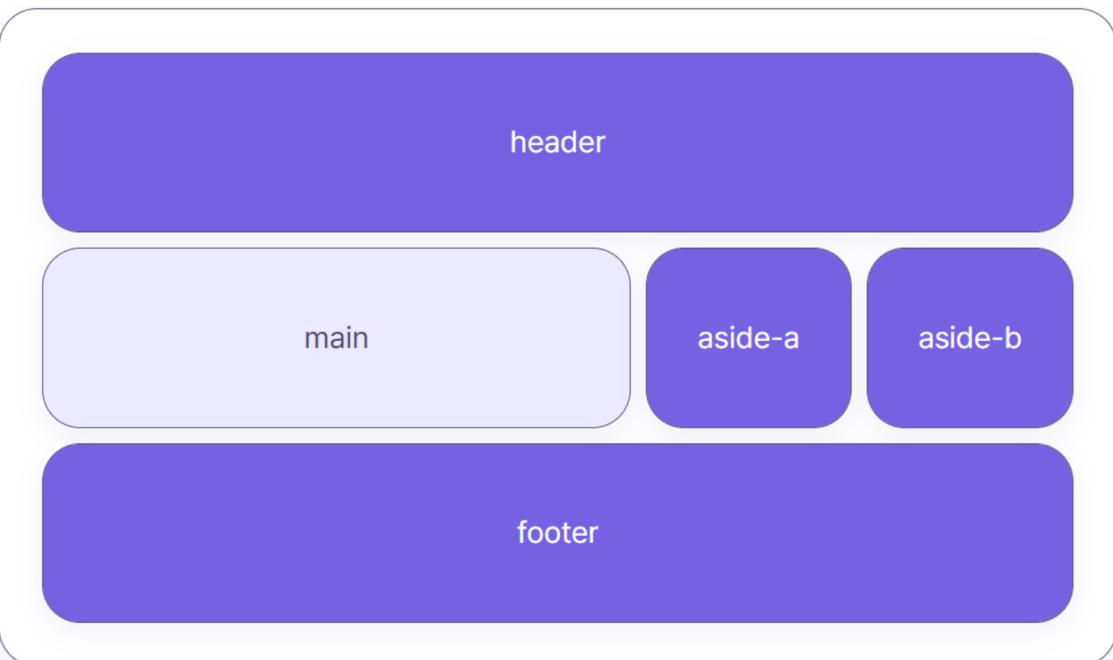
2. 아래와 같은 경우에도 order 속성을 사용할 수 있다.

이미 만들어진 모듈을 사용하여 웹을 구성한다면 초기에는 아래와 같은 배치가 된다. 마크업 순서가 header부터 footer까지 차례대로 되어있으므로, 각 요소의 크기를 조정해주어도 기본적으로 순서는 바뀌지 않는다.



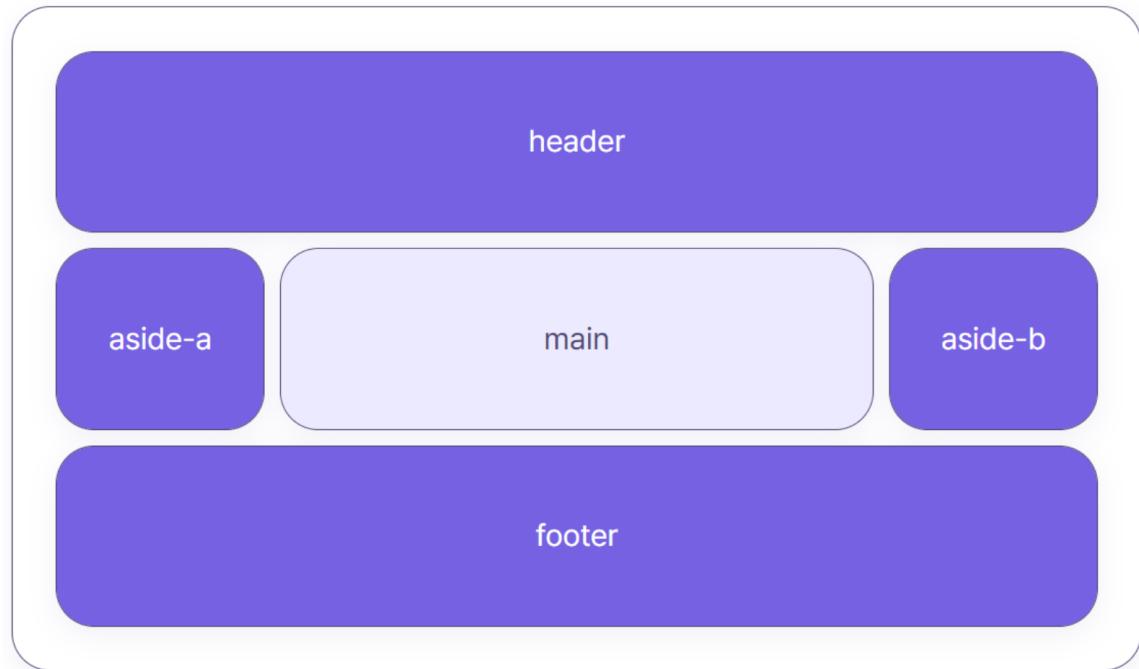
- **order 속성을 주지 않았을 때**

요소들의 크기를 조정해주었지만, 마크업 순서대로 배치되기 때문에 order 속성을 주지 않는다면 기본적인 순서를 변경할 수 없다.



- **order 속성을 주었을 때**

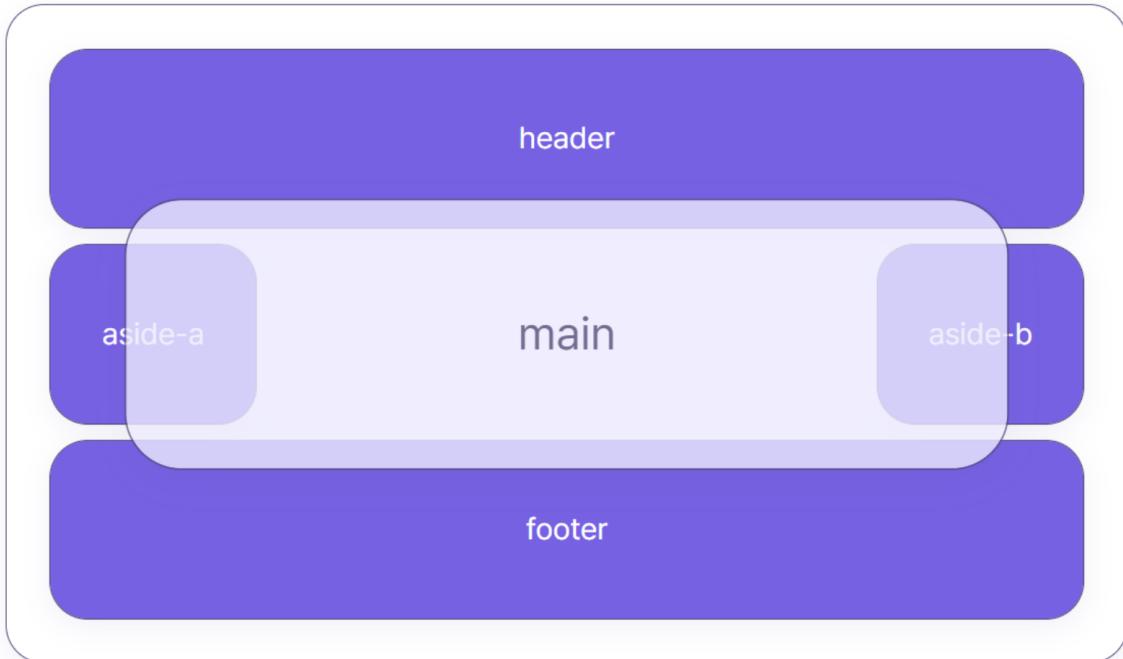
현재 main 부분을 가운데로 옮기려고 한다. 그러므로 aside-a, main, aside-b 요소에 각각 order를 주면 배치가 되지만, 기본적으로 order 값을 1이라도 주게 되면, order 값이 없는 속성이 더 우선시되기 때문에 마크업상 아래에 있는 footer 부분에도 order 값을 주어야 한다. 즉 우리가 원하는 순서대로 각 요소에 order를 1, 2, 3, 4를 주게 되면, 우리가 원하는 레이아웃을 만들 수 있다.



```
.main {  
    width: 60%;  
    order: 2;  
}  
  
.aside {  
    width: 20%;  
}  
  
.aside.aside-a {  
    order: 1;  
}  
  
.aside.aside-b {  
    order: 3;  
}  
  
.footer {  
    order: 4;  
}
```

9.3. z-index

flex에서도 z-index 속성을 사용할 수 있다. 어떤 요소에 z-index의 값이 1이라도 주어진다면 마치 다른 요소를 덮어 씌우는 것과 같은 모습을 표현할 수 있다.



```
.main {  
    width: calc(60% - 16px);  
    order: 2;  
    z-index: 1;  
    transform: scale(1.5);  
    opacity: 0.8;  
}
```

10. IE 지원을 위한 Flex

10.1. -ms- prefix 사용 예시

IE에서 지원하지 않는 Flex 속성

`flex-flow`, `justify-content: space-around`, `align-self`, `align-content` 의 4가지 속성들은 현재 지원하고 있지 않다. 또 `flex-grow`, `flex-shrink`, `flex-basis` 의 3가지 속성들은 단축 속성이 `-ms-flex: 0 1 auto;` 를 통해 지원하고 있다. (IE10버전에서는 `-ms-flex: 0 0 auto;` 를 통해 지원하고 있다.)

Flex 속성 IE 지원 대응표

속성	IE 지원
<code>display: flex;</code>	<code>display: -ms-flexbox;</code>
<code>flex-grow</code>	<code>-ms-flex</code>
<code>flex-direction</code>	<code>-ms-flex-direction</code>
<code>flex-wrap</code>	<code>-ms-flex-wrap</code>
<code>order</code>	<code>-ms-flex-order</code>
<code>justify-content</code>	<code>-ms-flex-pack</code>
<code>align-items</code>	<code>-ms-flex-align</code>

11. 성배 레이아웃 그리기

11.1. Flex 속성을 이용하여 기본 성배 레이아웃 구현하기

최종코드

```
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>성배레이아웃1</title>
    <link rel="stylesheet" as="style" crossorigin
        href="https://cdn.jsdelivr.net/gh/orioncactus/pretendard/dist/web/static/pretendard.css" />
    <style>
        :root {
            --dark: #504975;
            --white: #ffffff;
            --background: #fbfbff;
            --primary: #7661e2;
            --light: #eceaef;
        }

        * {
            font-family: Pretendard, -apple-system, BlinkMacSystemFont, system-ui, Roboto, 'Helvetica Neue', 'Segoe UI', 'Apple SD Gothic Neo', 'Noto Sans KR', 'Malgun Gothic', sans-serif;
        }

        body {
            height: 100%;
            padding: 10%;
            box-sizing: border-box;
        }

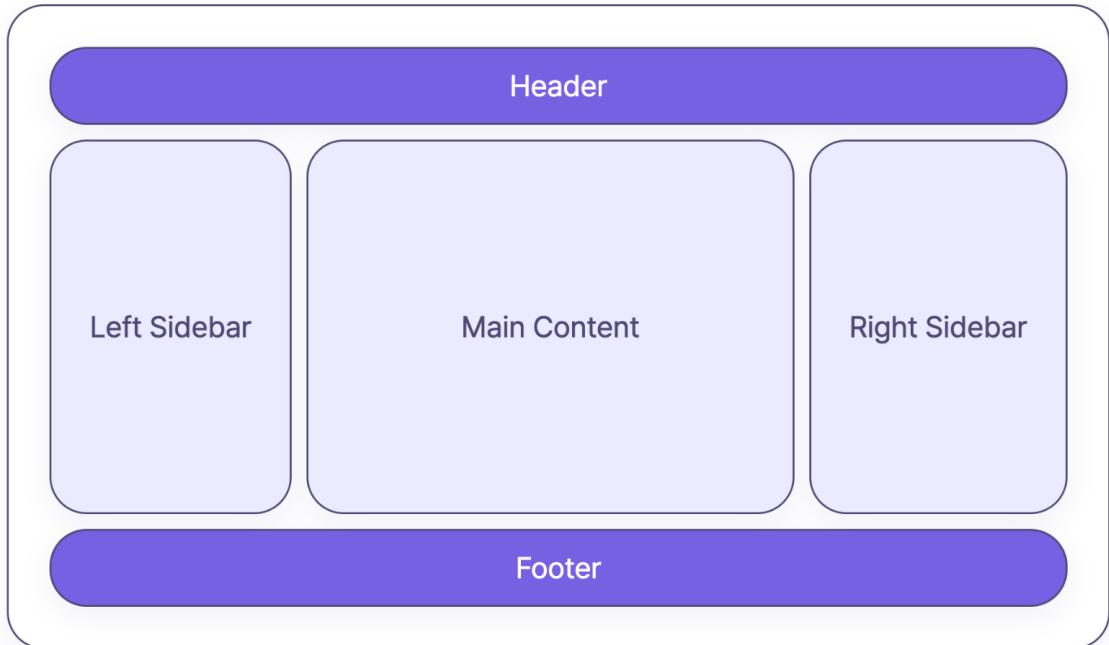
        .container {
            width: 70%;
            height: 300px;
            padding: 22px;
            margin: 0 auto;
            border: 1px solid var(--dark);
            border-radius: 20px;
            background: #fff;
            box-shadow: 0px 4px 12px 0px #51459f14;
        }
    </style>

```

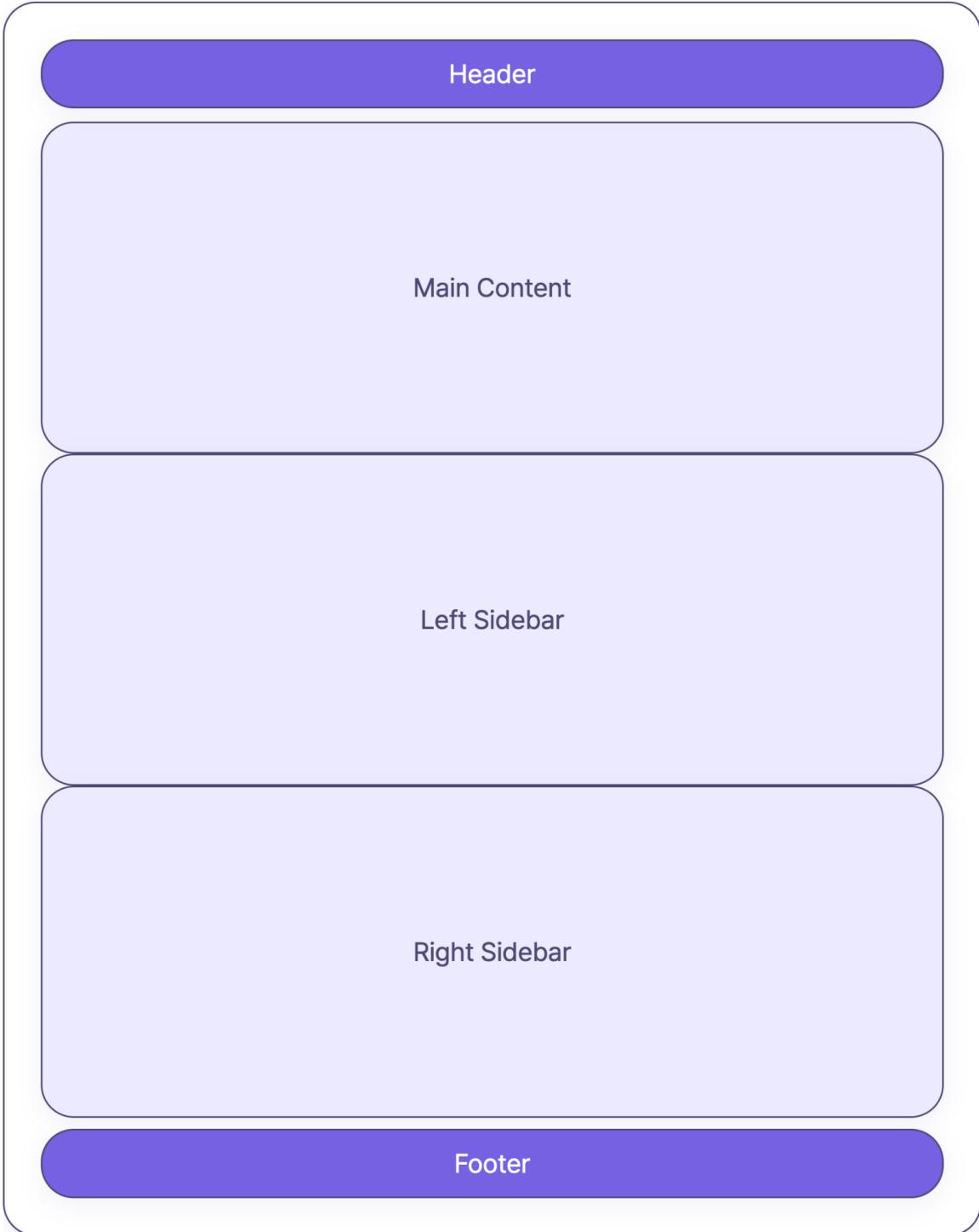
```
.item {  
    height: 40px;  
    /* padding: 0px ; 콘텐츠에 따라 조절해주세요 */  
    border: 1px solid var(--dark);  
    border-radius: 20px;  
    background-color: var(--primary);  
    color: var(--white);  
    box-shadow: 0px 4px 15px 0px #51459F14;  
    text-align: center;  
    vertical-align: middle;  
    line-height: 2.5;  
    word-break: keep-all;  
}  
.wrap{  
    display: flex;  
    flex-flow: row wrap;  
    gap: 0.5rem;  
    margin: 0.5rem 0;  
}  
.main-content, .left-sidebar, .right-sidebar{  
    background-color: var(--light);  
    color: var(--dark);  
    padding: 80px 0;  
}  
.main-content{  
    flex-grow: 2;  
    order: 2;  
}  
.left-sidebar{  
    flex-grow: 1;  
    order: 1;  
}  
.right-sidebar{  
    flex-grow: 1;  
    order: 3;  
}  
</style>  
</head>
```

Flex_11 성배 레이아웃 그리기

```
<body>
  <div class="container">
    <div class="item header">Header</div>
    <div class="wrap">
      <div class="item main-content">Main Content</div>
      <div class="item left-sidebar">Left Sidebar</div>
      <div class="item right-sidebar">Right Sidebar</div>
    </div>
    <div class="item footer">Footer</div>
  </div>
</body>
</html>
```

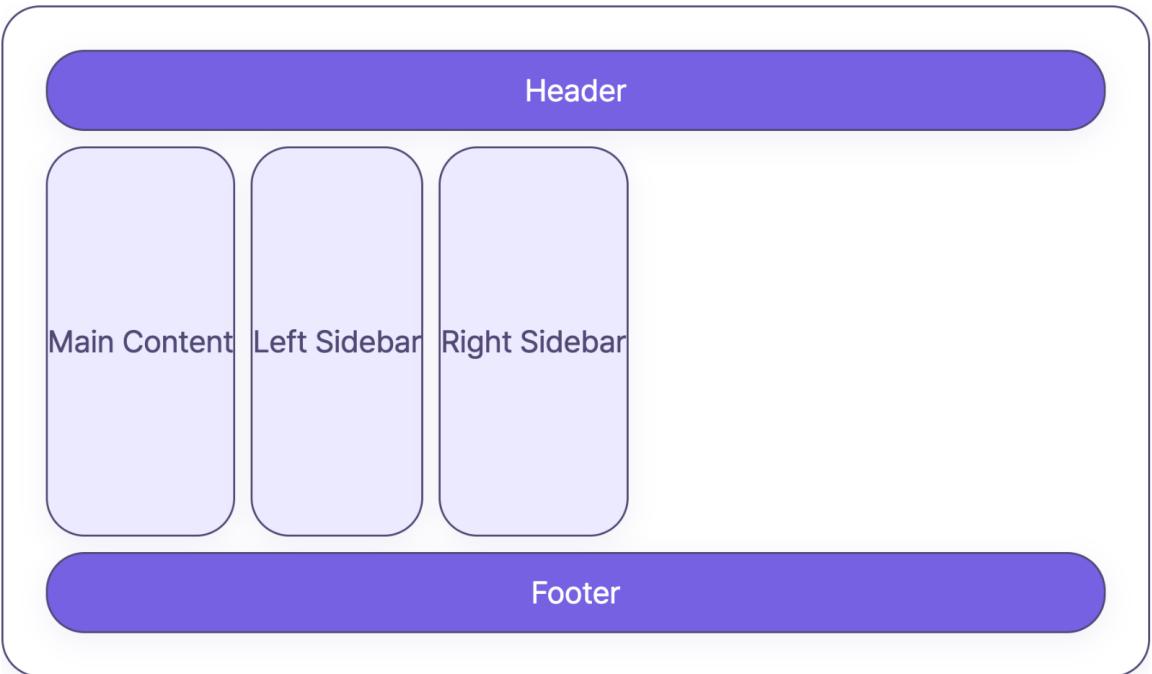


- 컨테이너 안에 요소들을 넣으면 마크업순서에 따라 자동적으로 위에서 아래로 쌓이게 된다. 우리는 `content`, `sidebar` 요소들을 나란히 배치하는 것이 목적이다.



```
<div class="container">
  <div class="header">Header</div>
  <div class="main-content">Main Content</div>
  <div class="left-sidebar">Left Sidebar</div>
  <div class="right-sidebar">Right Sidebar</div>
  <div class="footer">Footer</div>
</div>
```

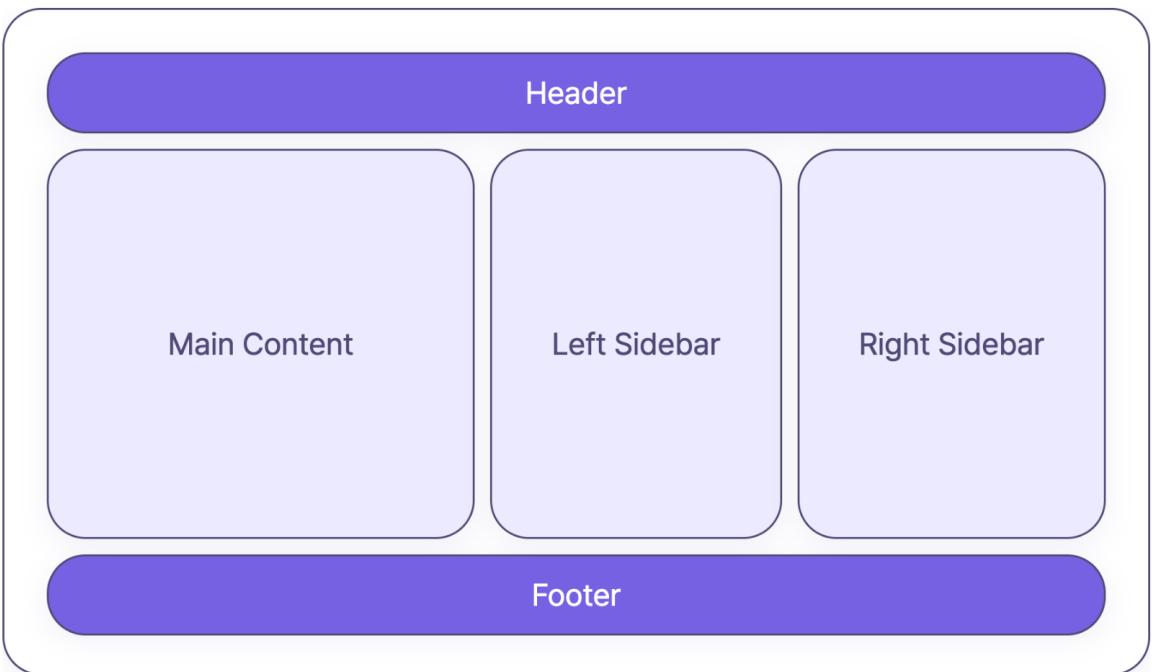
2. 나란히 배치하고 싶은 요소들을 `wrap` 으로 묶어준다. `wrap` 부분에 `display: flex;` 을 선언해 줌으로써 `flex-direction` 방향이 초기값인 `row` 로 정렬된다. 또한 `flex` 속성으로 인해 안의 요소들이 콘텐츠가 차지하는 만큼의 너비를 갖는다.



```
<div class="container">
  <div class="header">Header</div>
  <div class="wrap">
    <div class="main-content">Main Content</div>
    <div class="left-sidebar">Left Sidebar</div>
    <div class="right-sidebar">Right Sidebar</div>
  </div>
  <div class="footer">Footer</div>
</div>
```

```
.wrap{
    display: flex;
}
```

3. 아이템 요소는 윈도우 창 크기에 따라서 작아지기도 하고 커지기도 한다. 이때 비율을 설정해 주기 위해 남은 여백을 나눠 가질 수 있도록 숫자를 지정해준다. **Main Content** 가 가장 큰 비율을 차지해야 하므로 `flex-grow:2;` 를 지정해주고 나머지는 `flex-grow:1;` 로 지정한다.

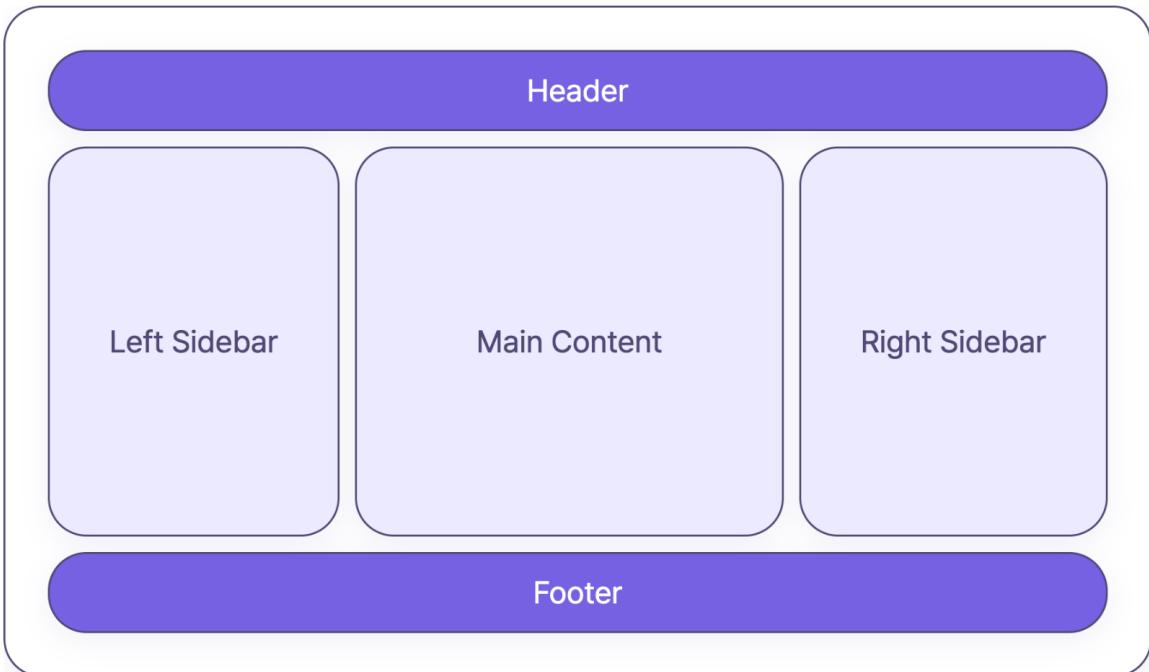


```
.main-content{
    flex-grow: 2;
}

.left-sidebar{
    flex-grow: 1;
}

.right-sidebar{
    flex-grow: 1;
}
```

4. 마크업 순서에 따라 `Left Sidebar`, `Main Content`, `Right Sidebar` 순으로 쌓이게 되는데, `Main Content` 를 가운데로 오게 배치해야 하므로 `order` 속성을 각각 정해주어 브라우저 상에 표시되는 모습을 지정하여 완성한다.



```
.main-content {  
    flex-grow: 2;  
    order: 2;  
}  
  
.left-sidebar {  
    flex-grow: 1;  
    order: 1;  
}  
  
.right-sidebar {  
    flex-grow: 1;  
    order: 3;  
}
```

11.2. Flex 속성을 이용하여 변형된 성배 레이아웃 구현하기

최종코드

```
<!DOCTYPE html>
<html lang="ko">

<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Flexbox 성배레이아웃2</title>
    <link rel="stylesheet" as="style" crossorigin
        href="https://cdn.jsdelivr.net/gh/orioncactus/pretendard/dist/web/static/pretendard.css" />
    <style>
        :root {
            --dark: #504975;
            --white: #ffffff;
            --background: #fbfbff;
            --primary: #7661e2;
            --light: #eceaef;
        }

        * {
            font-family: Pretendard, -apple-system, BlinkMacSystemFont, system-ui, Roboto, 'Helvetica Neue', 'Segoe UI', 'Apple SD Gothic Neo', 'Noto Sans KR', 'Malgun Gothic', sans-serif;
        }
        body {
            height: 100%;
            padding: 10%;
            box-sizing: border-box;
        }
        .container {
            width: 70%;
            padding: 22px;
            margin: 0 auto;
            border: 1px solid var(--dark);
            border-radius: 20px;
            background: #fff;
            box-shadow: 0px 4px 12px 0px #51459f14;
        }
    </style>

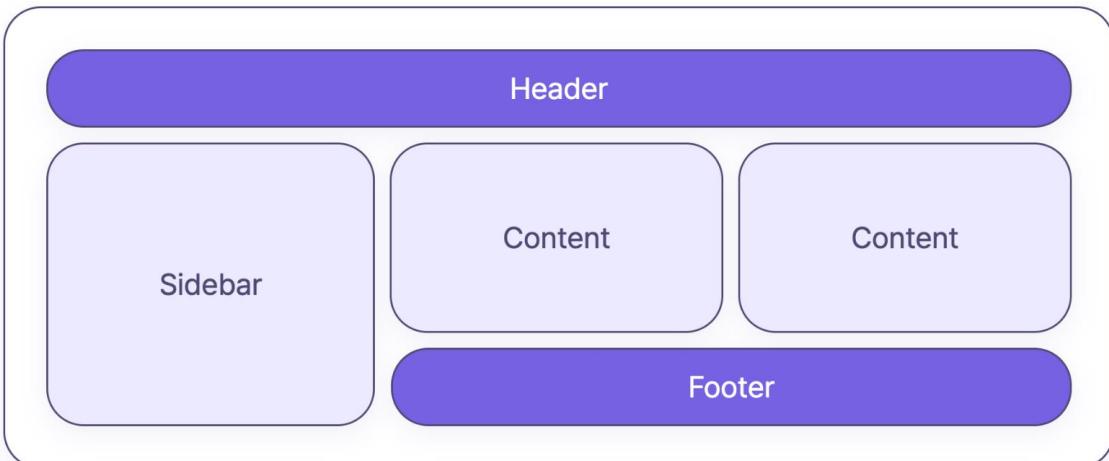
```

```
.item {  
    height: 40px;  
    /* padding: 0px ; 콘텐츠에 따라 조절해주세요 */  
    border: 1px solid var(--dark);  
    border-radius: 20px;  
    background-color: var(--primary);  
    color: var(--white);  
    box-shadow: 0px 4px 15px 0px #51459F14;  
    text-align: center;  
    vertical-align: middle;  
    line-height: 2.5;  
    word-break: keep-all;  
}  
.wrap{  
    display: flex;  
    flex-flow: row wrap;  
    gap: 0.5rem;  
    margin: 0.5rem 0;  
}  
.content, .sidebar{  
    background-color: var(--light);  
    color: var(--dark);  
}  
  
.content{  
    flex-grow: 1;  
    order: 2;  
    padding: 30px 0;  
}  
  
.sidebar{  
    flex-grow: 1;  
    order: 1;  
    padding: 55px 0;  
}  
  
.footer{  
    width: 66%;  
    margin-left: auto;  
    margin-top: -50px;  
}  
</style>  
</head>
```

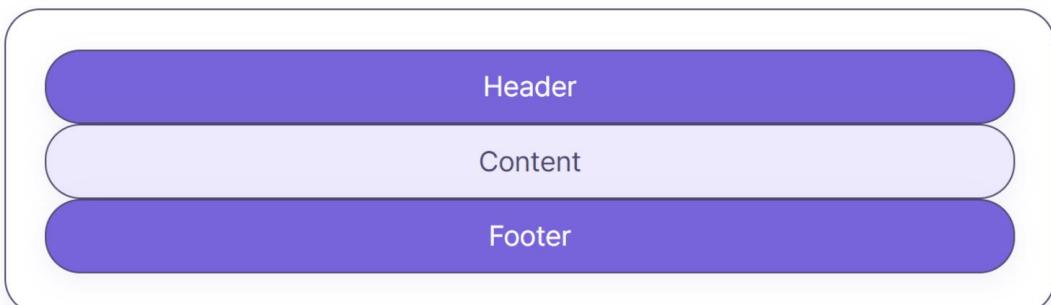
```

<body>
  <div class="container">
    <div class="item header">Header</div>
    <div class="wrap">
      <div class="item content">Content</div>
      <div class="item content">Content</div>
      <div class="item sidebar">Sidebar</div>
    </div>
    <div class="item footer">Footer</div>
  </div>
</body>
</html>

```



1. 기본 html 코드를 작성해본다. 큰 `container` 안에 `header` , `content` , `footer` 세가지로 나뉜 상태에서 제작을 한다.

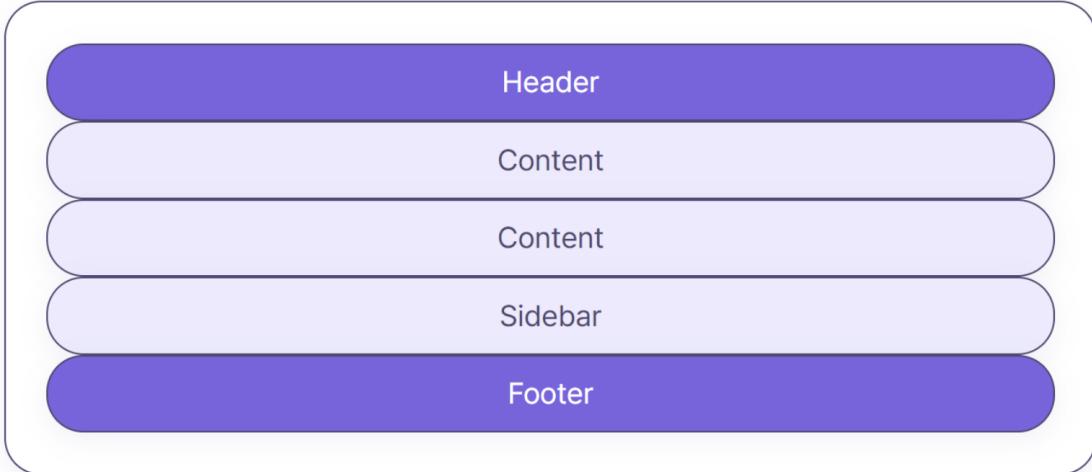


```

<div class="container">
  <div class="header">Header</div>
  <div class="content">Content</div>
  <div class="footer">Footer</div>
</div>

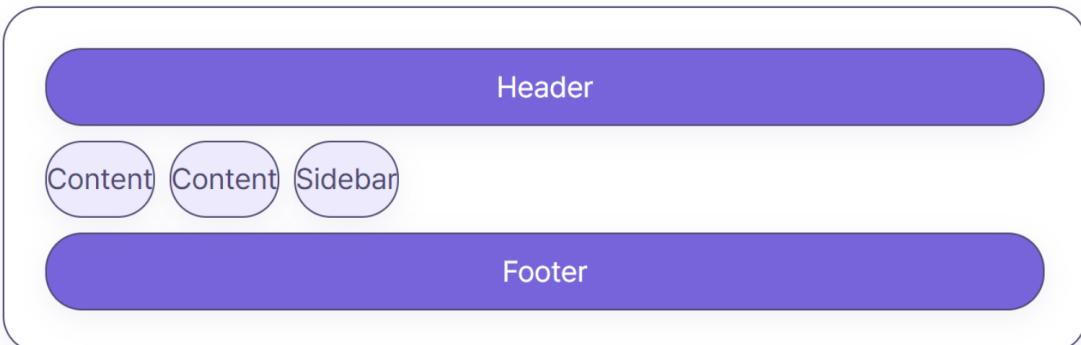
```

2. `content` 를 세분화하여 2개의 `content` 와 1개의 `sidebar` 가 새로운 요소로 나올 수 있게끔 분할하고 상위요소로 하나의 `wrap` 으로 묶어준다. 우리는 `wrap` 요소의 `flex` 를 이용하여 자유롭게 변형된 레이아웃을 배치하는 것이 목적이다.



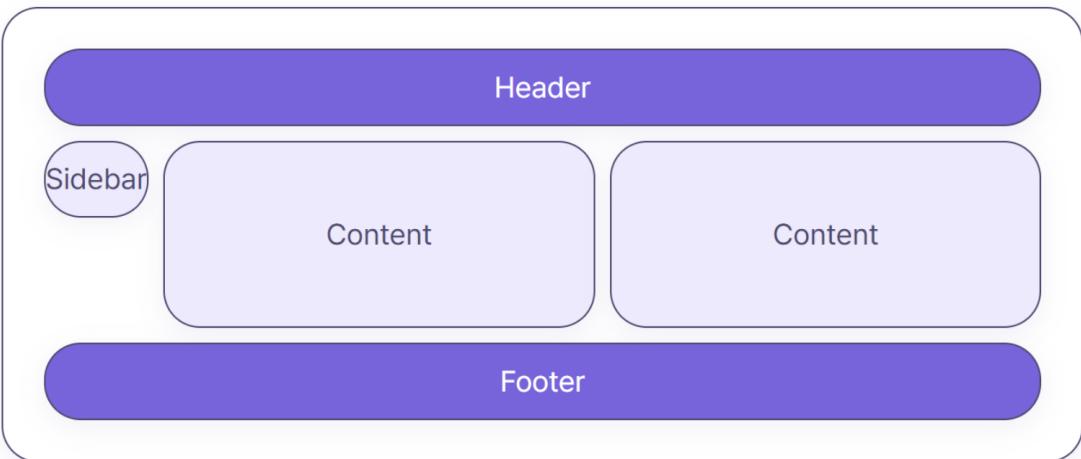
```
<div class="container">
  <div class="header">Header</div>
  <div class="wrap">
    <div class="content">Content</div>
    <div class="content">Content</div>
    <div class="sidebar">Sidebar</div>
  </div>
  <div class="footer">Footer</div>
</div>
```

3. `content` 와 `sidebar` 요소들의 상위 부모 요소인 `wrap` 에 `display:flex` 속성을 부여하고 각 요소 간 여백과 간격을 부여한다. `flex`의 하위 요소들의 간격을 조정할 때는 `gap` 속성을 이용하며 부모 요소인 `wrap` 자신의 여백을 조정할 때는 `margin` 을 이용한다.



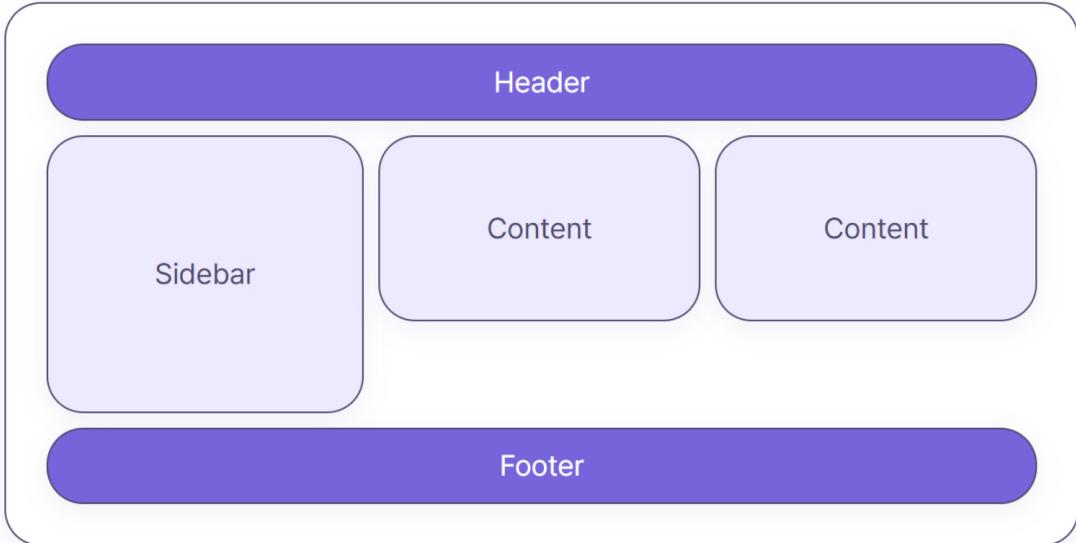
```
.wrap{
  display: flex;
  flex-flow: row wrap;
  gap: 0.5rem;
  margin: 0.5rem 0;
}
```

4. `contents` 의 스타일을 조정해본다. `flex-grow` 속성으로 여백을 동일하게 주도록 프로퍼티 값을 1로 통일해주고, 마크업 순서 상 `content` 가 가장 먼저 나와야 하지만 디자인 상으로는 `sidebar` 가 앞으로 나오도록 설정해주어야 함으로 `order` 값을 `sidebar` 보다 높은 양수의 숫자로 표현한다. (패딩 값은 임의의 높이를 설정하기 위하여 임의의 값을 지정하였다)



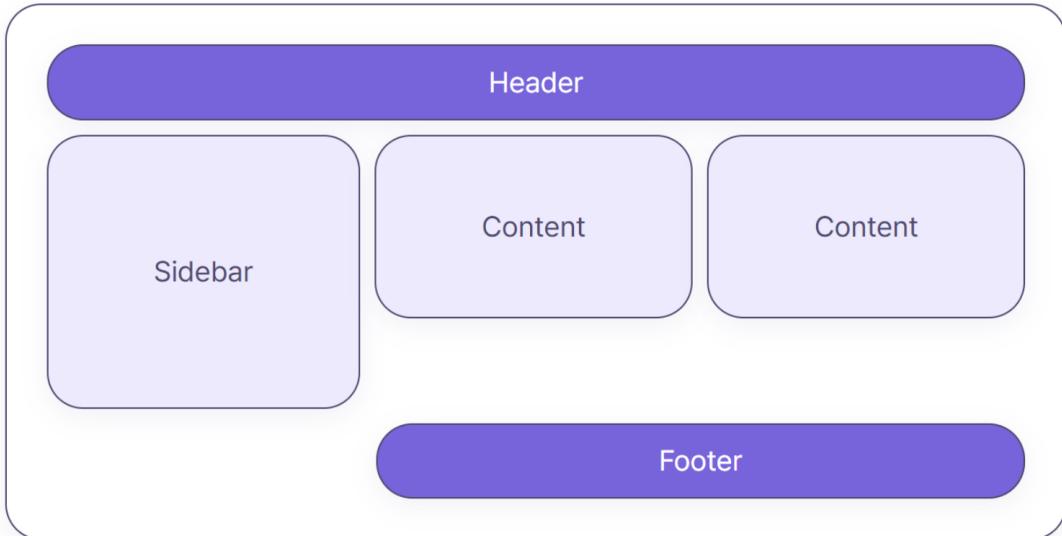
```
.content{
  flex-grow: 1;
  order: 2;
  padding: 30px 0;
}
```

5. `sidebar` 의 크기를 조정해준다. `content` 와 같은 여백을 가지기 위하여 `flex-grow` 값을 1로 통일하였고, `content` 보다 레이아웃 상 앞으로 배치하기 위하여 `order` 값을 `content` 의 `order` 값 2보다 작은 1로 작업해본다. `content` 와 마찬가지로 `content` 보단 높은 임의의 높이값을 설정하기 위하여 패딩값을 주었다.

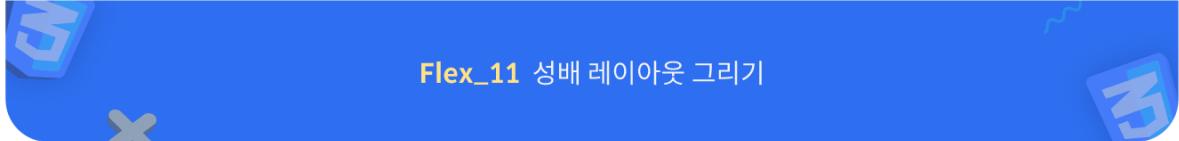


```
.sidebar{  
    flex-grow: 1;  
    order: 1;  
    padding: 55px 0;  
}
```

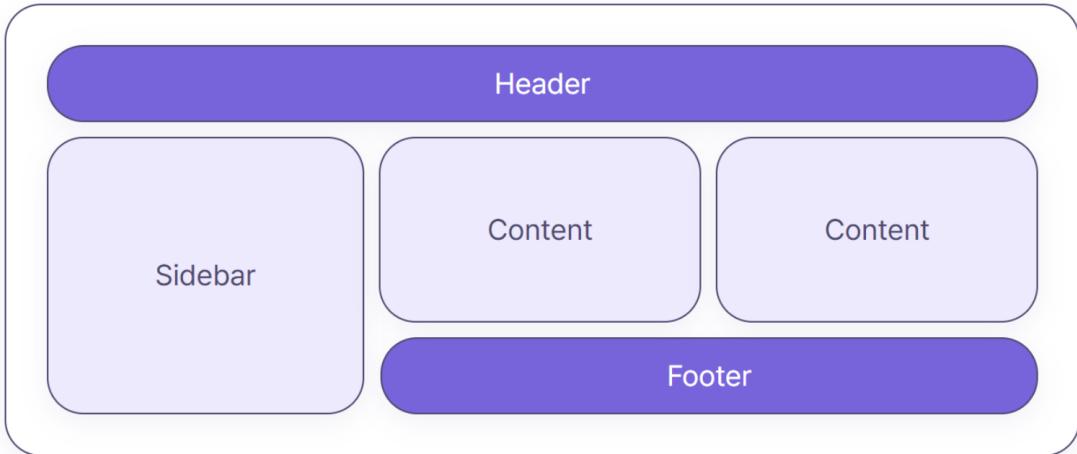
6. `footer` 의 스타일링 차례이다. `footer` 를 `content` 의 바로 아랫공간으로 이동시키기 위해서 전체 `content` 값을 유동적으로 적용할 수 있도록 `footer` 의 `width` 를 퍼센티지의 값으로 넣는다. 그리고 `footer` 는 블록 요소이므로 `footer` 의 해당 줄 전체를 차지하는 `margin` 의 왼쪽값을 `auto` 로 부여하여 `footer` 의 왼쪽 여백을 자동으로 가져갈 수 있도록 작성한다.



```
.footer{  
    width: 66%;  
    margin-left: auto;  
}
```



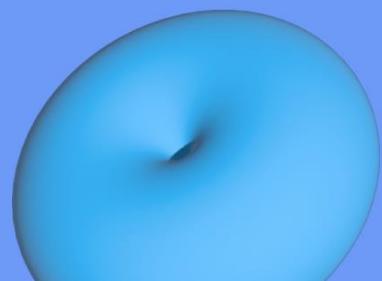
7. 마지막 단계이다. `footer` 여백에 여백의 반대라는 의미인 `nagative margin` 을 적용하여 `footer` 요소를 위쪽으로 끌어올려주면 `flex` 의 변형된 성배레이아웃 작성이 끝난다.



```
.footer{  
    width: 66%;  
    margin-left: auto;  
    margin-top: -50px;  
}
```

03 Grid 핵심개념

1. Grid
2. Grid row , Grid column
3. Gap
4. 각 셀의 영역 지정
5. IE지원을 위한 Grid



웹이 나오기 시작한 초기에는 종종 table(표)을 사용해 레이아웃을 구성하였다. 그러나 시멘틱한 마크업의 중요성이 대두되며 점차 이 방식은 지양하게 되었는데, 애초에 table이 표 형식의 데이터를 관리하고 제어하기 위해 만들어졌기 때문이다. 또한 table로 구성된 레이아웃은 반응형 디자인을 고려하기에는 적합하지 않았다.

이후 table의 대안으로 꽤 오랜 기간 사용해왔던 것이 float이다. 그러나 역시 float도 본래의 용도는 이미지와 텍스트의 레이아웃이기 때문에 우리가 원하는 복잡한 어플리케이션 레이아웃을 구현하기에는 적합하지 않았고, 여전히 반응형 디자인에서 제어가 어렵다는 단점이 있었다.

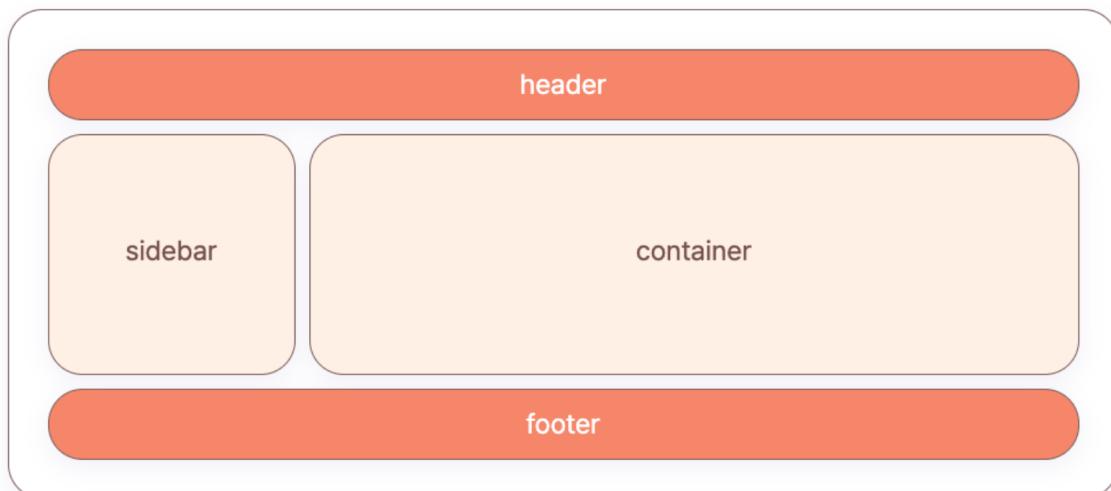
결국 CSS Working Group은 table과 float의 단점을 커버하면서, '레이아웃'의 용도에 적합한 Grid Layout Model을 만들게 되었다. 앞서 살펴보았던 Flex를 통해 손쉬운 반응형 레이아웃을 구현할 수 있었다.

Flex는 부모 요소 안에서 한 방향으로만 아이템들을 배치할 수 있지만, Grid는 행과 열, 두 가지 방향으로 배치가 가능하기 때문에 어렵고 복잡한 레이아웃 작성에 매우 적합하다.

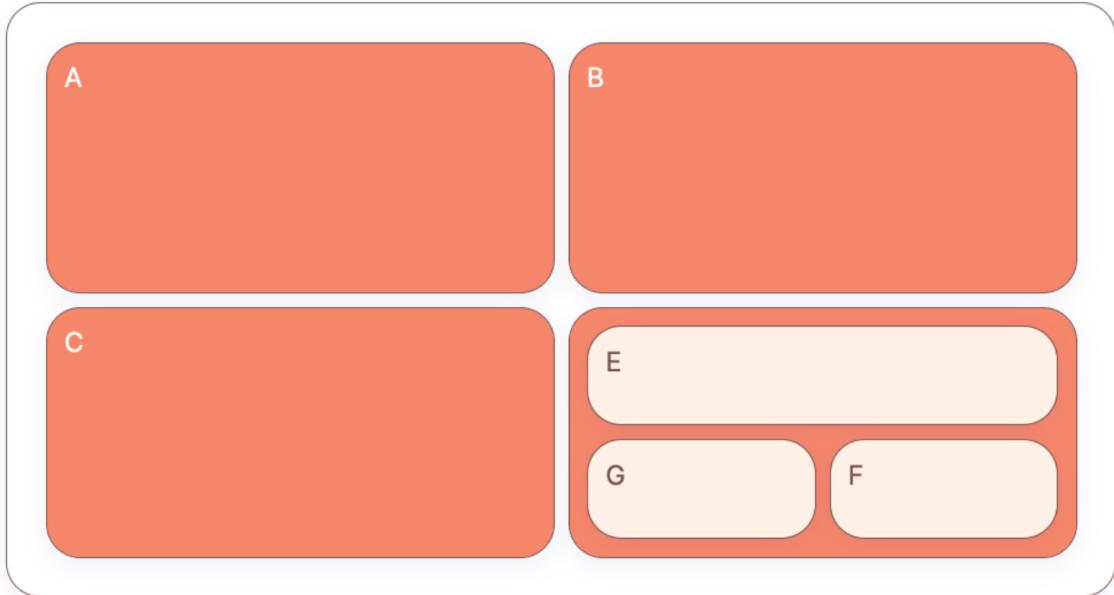
1. Grid

1.1 grid로 할 수 있는 것들

Grid는 1차원, 2차원 레이아웃을 구상할 수 있다. 아래와 같이 헤더-사이드바-컨테이너-푸터로 이루어진 기본적인 웹 페이지의 전체적인 레이아웃을 짜는 것이 가능하다.

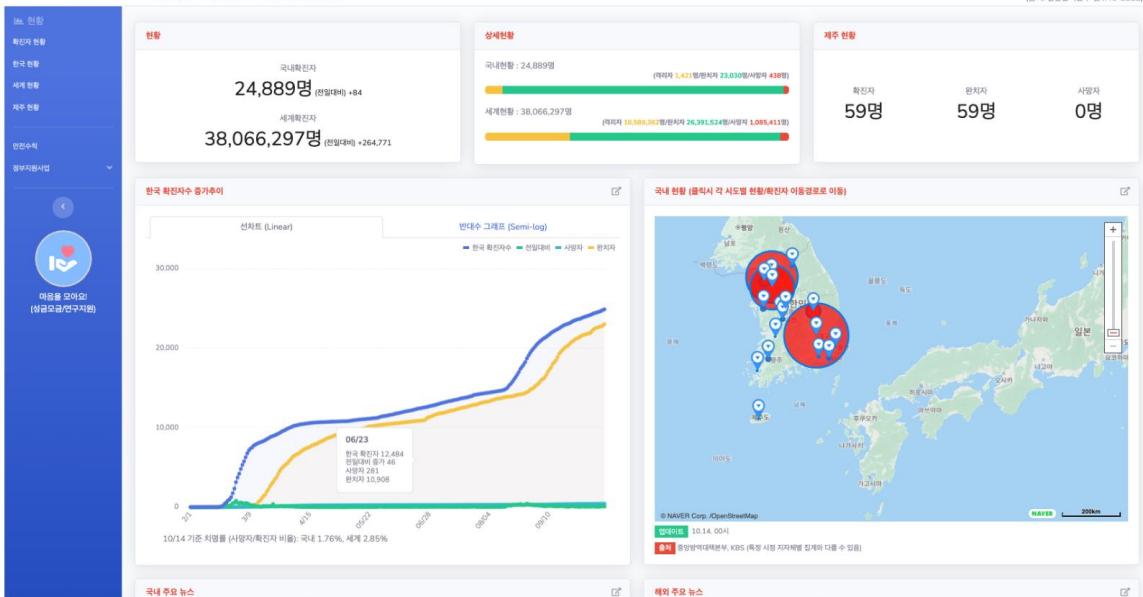


Grid 안의 아이템들도 또다시 grid 레이아웃으로 만드는게 가능하다. 아주 작은 텍스트 단위의 요소도 grid를 사용하여 아이템을 배치할 수 있다.



Grid 레이아웃으로 쉽게 만들 수 있는 홈페이지의 예이다. 헤더-사이드바-컨테이너로 구성되어 있고, 컨테이너 안의 아이템들도 grid를 사용한 것을 확인할 수 있다.

● 라이브 코로나 맵 (10월 14일 업데이트 종단!)



Grid_1 Grid

● 라이브 코로나 맵 (10월 14일 업데이트 중단!)

UPDATE: 10.14. 004
(출처: 질병관리본부 및 JHU CSSE)

The main content area displays a map of South Korea with various regions highlighted in different colors (blue, green, yellow, red) indicating different levels of COVID-19 risk or status. Overlaid on the map are several boxes of text and graphics:

- 코로나바이러스감염증-19 행동수칙**
- 유증상자**: 발열이나 호흡기 증상(기침이나 폐아픔 등)이 나타난 사람입니다. 동과나 출근을 하지 마시고 의료기관을 자제해 주십시오.
- 진단에서 중요한 흥미를 위하여 3-4일 겔과를 관찰하여 주십시오.**
- 38도 이상 고열이 지속된다거나 증상이 심해지면**: ① 플라스틱이나 종이로 만든 대형(4x10), 보건소로 문의 ② 병원으로 소화기 증상(증상)에 대해 문의하세요.
- 의료기관 방문 시** 기저귀를 이용하고 마스크를 착용하십시오.
- 진료 의료기관에 와서 아랫배 및 호흡기 증상처럼 접촉 여부를 알려주세요.**
- 방역, 기관 등 호흡기 증상이 있는 시**: 질병관리본부 1339, 경찰청 112, 119
- 기본방역수칙 가드**
- 코로나19 바이러스 관련 뉴스&이슈 체크**
- 소상공인 지원사업**

통계치

* 20년 10.14 기준				
병명	신증인원루연자 A(H1N1)	증증호흡기증후군(MERS)	증증급성호흡기증후군(SARS)	코로나19(COVID-19)
발진장소	멕시코	사우디아라비아		중국
감염예방체	사감	나타		백신(추정)
전파방식			비밀감염*	
주요증상			발열 및 호흡기 증상 등	
치료율	약 0.07%	약 35%	약 10%	세계 2.85%, 대한민국 1.76%
기초감염증생산수(R0)*	1.1~1.7	0.4~0.9	2~5	5~5.7
감염사망자	집계되지 않음(18,449)	2,482(854)	8,096(774)	2,711,414(192,126)

*기초감염증생산수(R0): 한 사람의 감염자가 몇 명을 전염시킬 수 있는지를 나타낸 값
*비밀감염: 감염자와 침, 물을 등 세제에 다른 사람의 입이나 코로 들어가 감염이 이루어지는 것

사회적 거리두기 사나리오별 상황예측

- 원자 0
- 연장 200
- 감염 0

TOP 1 발생국가(글로벌 일일 차트)

| 1. 미국(United States) : 8,090,253 (21.08%) | 2. 인도(India) : 7,239,389 (18.87%) |
| 3. 브라질(Brazil) : 6,480,000 (17.01%) | 4. 러시아(Russia) : 1,53% / 원자율 : 87.05 |

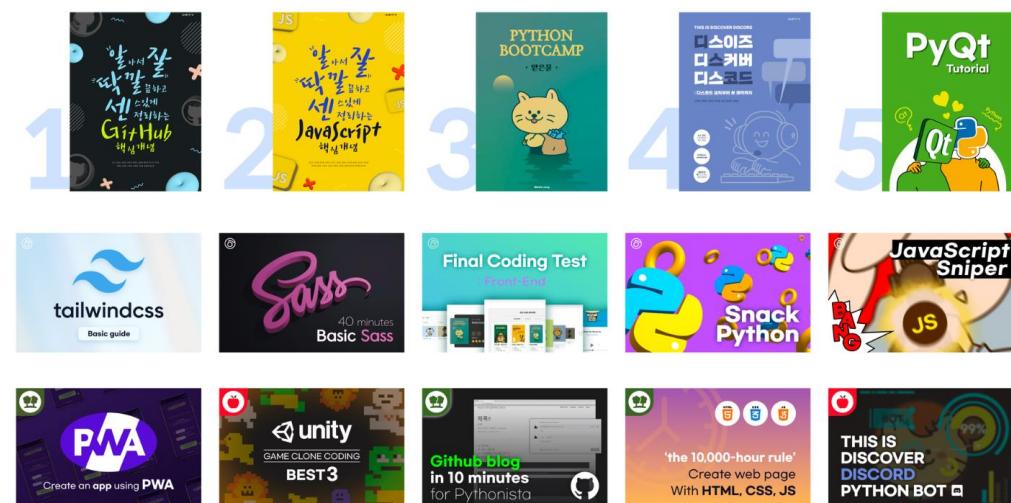
 The chart uses a color legend: light orange for active cases, dark orange for recovered cases, and red for deceased cases."/>

1.2. 그리드 용어정리

GRIDFLEX

HOME 경의 도서 Top10

이달의 TOP 5 IT 도서





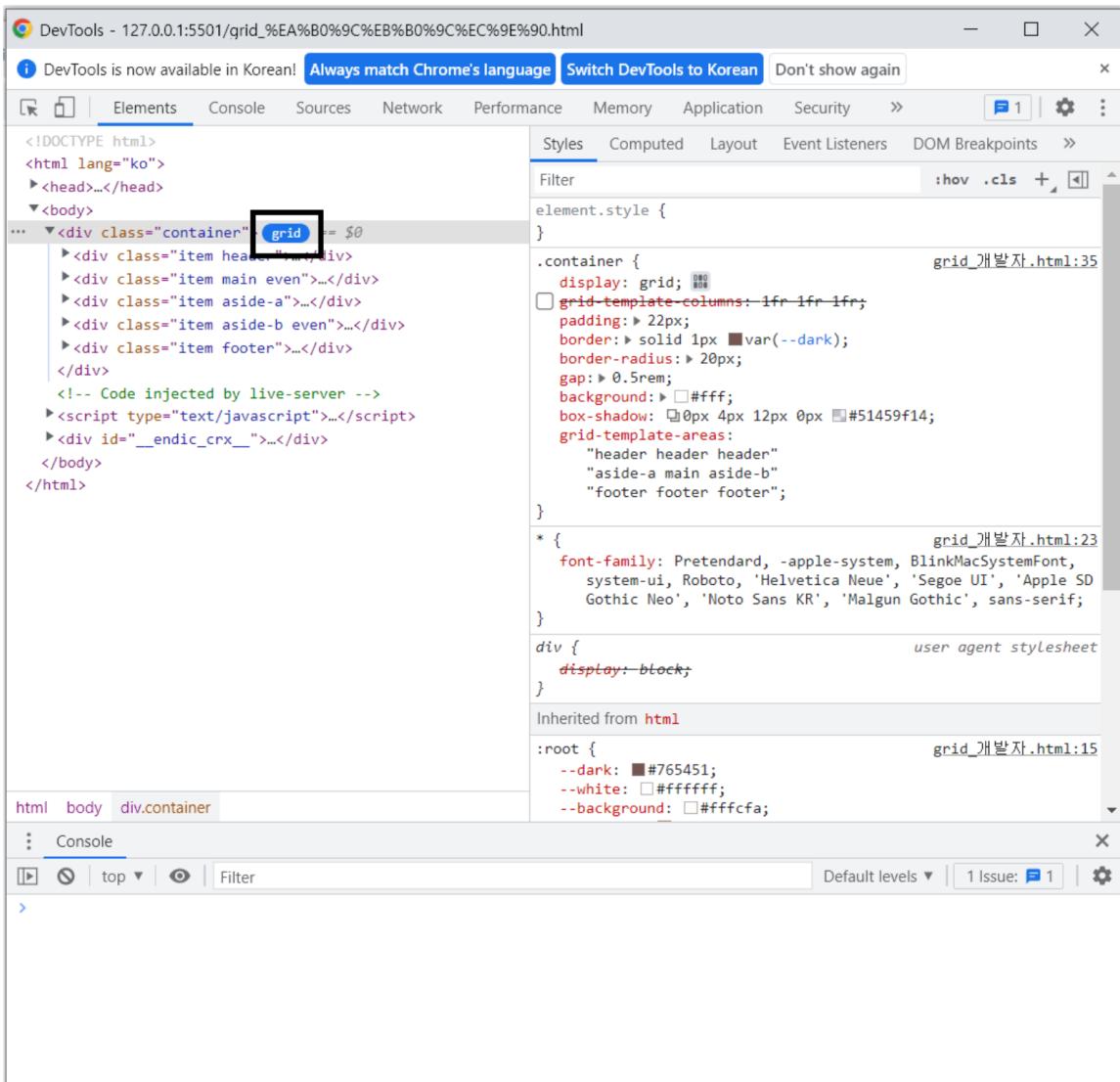
- **그리드 컨테이너 (Grid Container)**
 - grid를 적용하는 전체 영역을 의미. 열(COLUMNS)과 행(ROWS)을 가지며, 그리드 아이템(ITEMS)을 배치
- **그리드 아이템 (Grid Item)**
 - grid 컨테이너의 자식 요소들이 아이템들이 grid 규칙에 의해 배치
- **그리드 라인(Grid Line)**
 - grid를 그리는 행(ROW)/열(COLUMN)을 그리는 선
 - 각 선은 라인 넘버를 가지며 그리드 아이템을 배치하는 기준이 된다.
- **그리드 트랙 (Grid Track)**
 - grid 라인 사이의 행(ROW) 또는 열(COLUMN) 공간
- **그리드 셀 (grid Cell)**
 - grid의 한 칸을 가리키는 말 (유닛으로 부르기도 한다)
 - 4개의 그리드 라인이 모여 그려지는 한 칸의 공간
- **그리드 번호(Grid Number)**
 - grid 라인의 각 번호
- **그리드 영역(Grid Area)**
 - 4개의 그리드 라인으로 둘러싸인 공간으로 그리드 셀이 묶인 영역
 - 식별자를 통해 요소를 배치

- 그리드 갭(Grid Gap)
 - 그리드 거터(Grid Gutters)
 - 행(Row) 또는 열(Column) 사이의 간격

1.3. 개발자 도구로 그리드 체크하는 법

Chrome에서 개발자 도구를 이용하면 HTML 요소에 grid 속성이 적용된 요소는 옆에 버튼이 생기게 되는데 이 버튼을 누르고 닫음으로써 grid 요소의 배치, 속성, 크기를 파악할 수 있다.

맥은 <command + option + i>, 윈도우는 <ctrl + shift + i> 또는 F12를 이용하여 개발자 도구로 진입 할 수 있다.



The diagram illustrates a grid-based layout structure. A large orange rounded rectangle at the top contains text with four small red boxes numbered 1 through 4 pointing to specific parts. Below this are three smaller orange rounded rectangles, each with a red box numbered 1 through 4 pointing to its content. The text in all boxes is placeholder Latin text.

1. Lorem ipsum dolor sit, amet consectetur adipisicing elit. Amet mollitia harum voluptatem! Distinctio impedit fuga minus voluptatibus rerum dicta architecto porro qui aut. Repellendus magni possimus velit eaque dolorem accusantium?

2. **1.** Lorem ipsum dolor sit, amet consectetur adipisicing elit. Blanditiis, sed minima doloremque cum ab sapiente quibusdam dolore soluta mollitia, dolores, veritatis tempora odio voluptatum sequi iure illo temporibus fuga autem?

2. **2.** Lorem ipsum dolor sit amet consectetur adipisicing elit. Enim, est voluptatibus hic incident, at in, omnis corporis harum unde illum velit reiciendis accusantium veniam aliquam cum iure nihil dolor provident!

2. **3.** Lorem ipsum dolor sit amet consectetur adipisicing elit. Exercitationem vitae incident quasi sit blanditiis nulla neque quaerat omnis voluptatem rem id quibusdam mollitia, vel provident quam alias at. Nobis, corporis!

3. **1.** Lorem ipsum dolor sit amet consectetur adipisicing elit. Vero adipisci quam animi inventore provident facere nobis explicabo harum numquam iure quidem cum nostrum maiores veritatis, consequuntur saepe officia asperiores officiis?

4. **1.**

4. **2.**

4. **3.**

4. **4.**

Grid_1 Grid

또한, Styles 탭에서 `display : grid;` 속성 옆의 버튼을 이용해서 요소에 grid의 다양한 속성을 바로 적용하여 어떻게 변하는지 바로 확인할 수 있다.

The screenshot shows the Chrome DevTools interface with the URL `127.0.0.1:5501/grid_%EA%B0%9C%EB%B0%9C%EC%9E%90.html`. The **Elements** tab is selected. In the left sidebar, under the `<div class="container">` node, there is a blue box around the `grid` property in the `grid` style. The right panel shows the **Styles** tab with the `grid` style expanded. It displays various grid properties like `display: grid;`, `grid-template-columns: 1fr 1fr 1fr;`, and alignment properties (`align-content: normal`, `justify-content: normal`, `align-items: normal`, `justify-items: normal`). A large black box highlights the `grid-template-columns` property and its value. Below it, a smaller black box highlights the `align-items` property and its value. The bottom of the right panel shows the `user agent stylesheet` and other inherited styles.

Styles 탭 옆의 Layout 탭에 들어가게 되면 그리드가 적용된 요소를 더욱 상세하게 볼 수 있다.

The screenshot shows the Chrome DevTools interface with the URL `127.0.0.1:5501/grid_%EA%B0%9C%EB%B0%9C%EC%9E%90.html`. The **Layout** tab is selected. In the left sidebar, under the `<div class="container">` node, there is a blue box around the `grid` style. The right panel shows the **Layout** tab with the `Grid` section expanded. It includes an **Overlay display settings** section with checkboxes for `Show line numbers`, `Show track sizes`, `Show area names`, and `Extend grid lines`. Below it is a **Grid overlays** section where a checkbox for `div.container` is checked, with a small preview icon next to it. The bottom of the right panel shows the `Flexbox` section and a message stating `No flexbox layouts found on this page`.

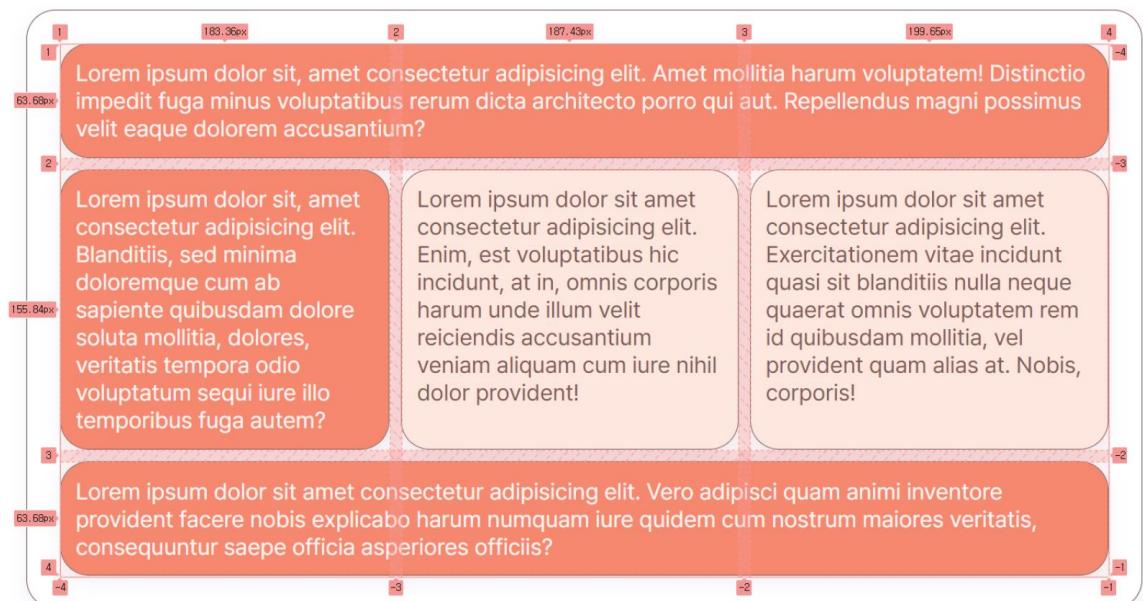
Grid_1 Grid

Show track sizes : 그리드 셀(Grid Cell)의 크기를 표시한다.

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. A container element with the class 'container' is expanded, revealing its child elements: 'item header', 'item main even', 'item aside-a', 'item aside-b even', and 'item footer'. In the bottom-left corner of the DevTools, there is a small preview of the page layout, which is a grid with four columns and some text content.

On the right side of the DevTools, under the 'Layout' tab, there is a section titled 'Overlay display settings'. Within this section, there is a checkbox labeled 'Show track sizes' which is checked. This setting allows the developer to see the individual track sizes for each grid cell.

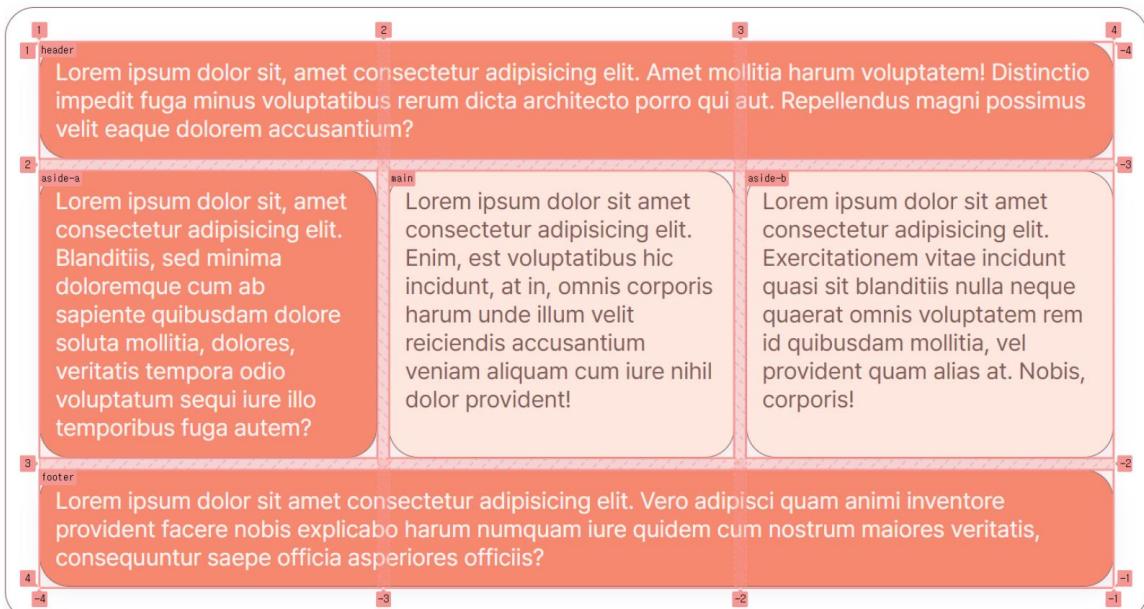
Below the 'Layout' tab, there are other sections like 'Grid overlays' and 'Flexbox', both of which are currently inactive.



Grid_1 Grid

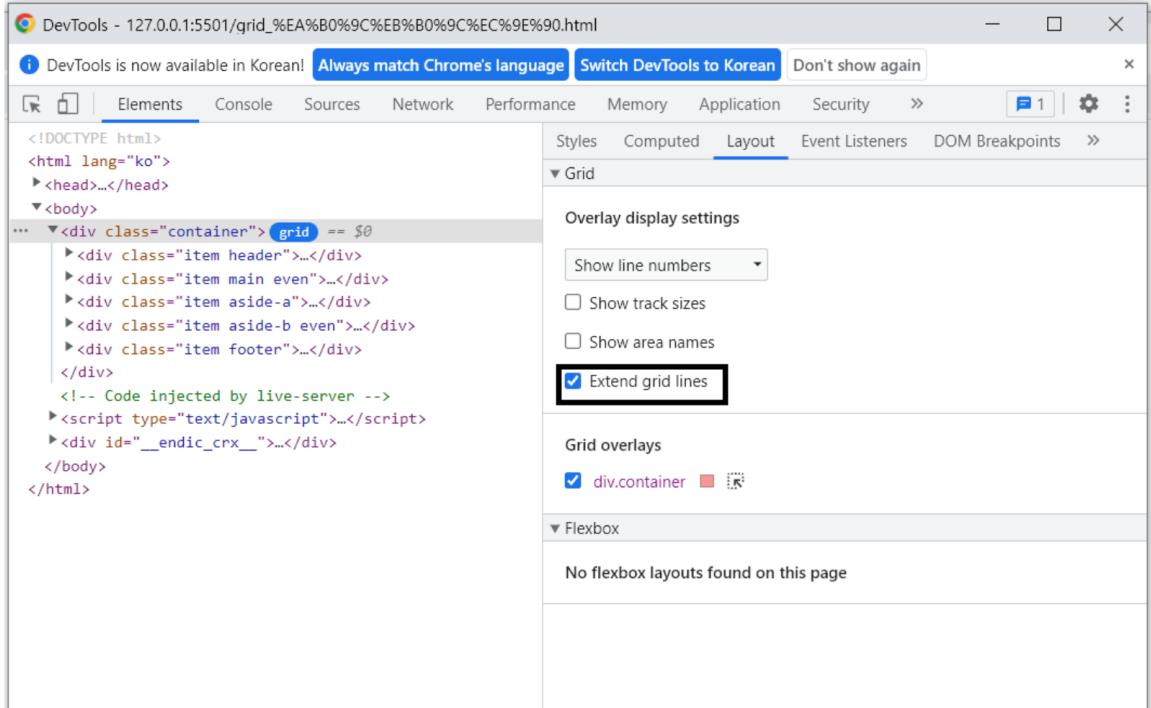
Show area names : `grid-template-areas` 속성이 적용되어 있다면, 그 영역의 이름을 표시한다.

The screenshot shows the Chrome DevTools interface with the 'Elements' tab selected. On the left, the DOM tree displays a structure with a 'container' div containing various child elements like 'header', 'main', 'aside-a', 'aside-b', and 'footer'. A tooltip 'grid' is shown over the 'container' element. On the right, the 'Layout' tab is active in the DevTools sidebar. Under the 'Grid' section, there is a 'Show area names' checkbox which is checked, highlighted with a red border. Other options like 'Show line numbers', 'Show track sizes', and 'Extend grid lines' are also present. Below this, the 'Grid overlays' section shows a preview of the grid layout with colored tracks (red, green, blue) and the 'div.container' selector.

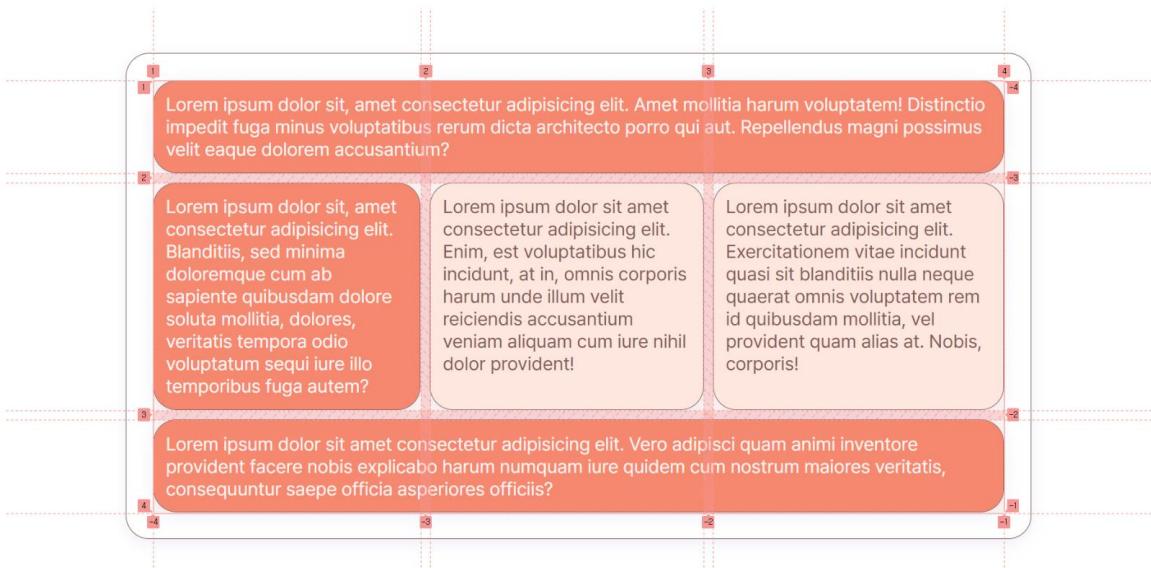


Grid_1 Grid

Extend grid lines : 그리드 라인(Grid Line)을 확장한다.



The screenshot shows the Chrome DevTools interface with the URL `127.0.0.1:5501/grid_1%EA%B0%9C%EB%B0%9C%EC%9E%90.html`. The 'Layout' tab is selected in the top navigation bar. In the 'Grid' section of the right sidebar, there is an 'Overlay display settings' panel with a checked checkbox labeled 'Extend grid lines'. Below this, under 'Grid overlays', there is a checked checkbox for 'div.container'. The main pane displays the HTML code for a grid layout, which includes a container with five items: header, main, aside-a, aside-b, and footer.



1.4. grid 사용법

Grid는 Flex와 마찬가지로 HTML 구조에 컨테이너와 아이템, 이렇게 두 가지 요소가 필요하다. 컨테이너는 부모 요소로, Grid 레이아웃의 영향을 받는 전체적인 공간이다. 반면 아이템은 자식 요소로, 컨테이너 내부에서 설정된 속성에 따라 배치된다. 이를 Grid에서는 각각 그리드 컨테이너(Grid Container)와 그리드 아이템(Grid Item)이라 한다.

```
<div class="container">
  <div class="item"></div>
  <div class="item"></div>
  <div class="item"></div>
</div>
```

grid를 사용하기 위해선 flex처럼 CSS에서 display 속성을 부여하는 과정이 필요하다. 컨테이너 `display:grid;`를 적용하면 이후 Grid가 제공하는 속성을 다양하게 사용할 수 있다.

선언 시 즉각적으로 아이템들이 가로로 배치되는 flex와 달리, grid는 `display:grid;` 선언을 해도 변화를 확인하기 어렵다. 왜냐하면 grid 레이아웃이 되었지만, 아직 컨테이너가 하나의 열(row)밖에 없어 아이템들이 기본정렬을 유지하고 있기 때문이다. 컨테이너에 `grid-template-columns`, `grid-template-rows` 속성을 추가하여 행과 열을 추가함으로써 그리드 형태를 명확히 확인할 수 있다.

```
.container{
  display:grid;
}
```

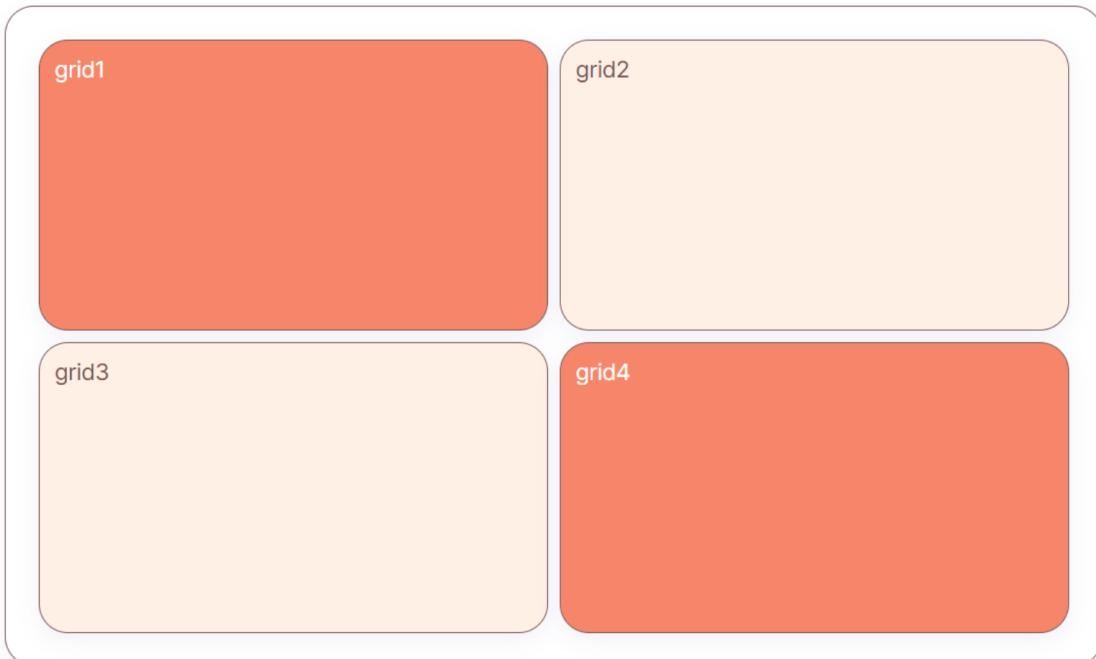
2. Grid row , Grid column

2.1. grid-template-rows, grid-template-columns

Grid는 행과 열로 이루어져 있다. 행은 row, '가로'이고 열은 column, '세로'이다. grid row와 column을 사용하기 위해서는 template 속성을 사용해 행과 열의 크기를 지정해 주어야 한다. 여기서 주의할 점은 행과 열의 **개수가 아닌 크기를** 지정하는 것이다.

- `grid-template-columns` : column(열)의 넓이(크기) 지정하기
- `grid-template-rows` : row(행)의 높이 지정하기

다음은 4개의 아이템의 column과 row에 각각 350px, 200px를 준 예제이다.



```
.container {  
    display: grid;  
    grid-template-columns: 350px 350px;  
    grid-template-rows: 200px 200px;  
}
```

이렇게 입력된 코드 값은 크기 단위로 입력되어 각 column과 row의 순서에 맞게 값이 적용 된 것을 볼 수 있다. 결론적으로 grid-template-columns와 grid-template-rows는 다음과 같이 사용할 수 있다.

```
grid-template-columns: column1넓이 column2넓이;
grid-template-rows: row1높이 row2높이;
```

2.2. repeat / 1fr이란

fr이란?

fr은 fraction의 줄임말로, fraction을 직역하면 '분수'라는 뜻을 가지고 있다. grid에서는 그리드 컨테이너에서 사용할 수 있는 공간의 일부를 나타내는 단위를 의미하는데, fr 단위를 사용하면 그리드 내부의 레이아웃 분할을 자동으로 계산해서 적용할 수 있다. 그리드 레이아웃에서 그리드 아이템의 크기를 지정할 때 px을 이용하면 항상 크기가 고정되기 때문에 반응형 웹 디자인에는 적합하지 않다. 그래서 상대적인 크기를 지정할 수 있도록 fr 단위를 사용하면, 더 편리하게 반응형 디자인을 적용할 수 있다. 예를 들어 `grid-template-columns: 1fr 1fr 1fr;` 이라는 코드를 적는다면, 컨테이너 안 그리드 아이템의 크기는 1:1:1의 비율로 각각 적용된다.

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

repeat() 함수

repeat()은 그리드 속성값으로 사용할 수 있는 함수로, 반복되는 값을 자동으로 처리할 수 있다. 만약, 스타일 중 반복되는 부분이 있다면 repeat() 함수를 이용해서 코드를 좀 더 간소화할 수 있다.

적용방법

```
repeat(반복횟수, 반복값);
```

아래 두 코드의 결과는 같다.

```
.container {  
    display: grid;  
    grid-template-columns: repeat(4, 140px);  
}
```

```
.container {  
    display: grid;  
    grid-template-columns: 140px 140px 140px 140px;  
}
```

grid1

grid2

grid3

grid4

반복을 줄일 수 있는 repeat() 함수 사용의 예:

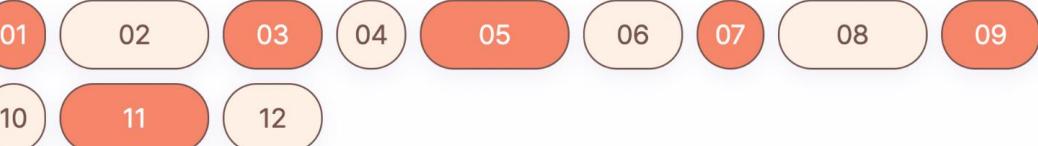
```
.container {
    display: grid;
    grid-template-columns: repeat(10, 1fr);
}
```

1fr를 10번 작성할 필요 없이, repeat() 함수를 이용해 간단하게 코드를 작성할 수 있다. 10개의 그리드 아이템이 같은 비율로 컨테이너에 들어가게 된다.



repeat() 값으로 같은 값/비율이 아닌 각기 다른 비율(fr)의 값을 줄 수도 있다. 아래와 같이 코드를 적으면 3개의 아이템이 1:3:2의 비율로 반복해서 들어간다.

```
.container {
    display: grid;
    grid-template-columns: repeat(3, 1fr 3fr 2fr);
}
```



2.3. min-max 함수

min-max 속성은 트랙의 크기를 최솟값과 최댓값으로 지정할 수 있는 함수를 의미한다. 그리드 열 또는 행의 값으로 minmax를 사용할 수 있다. minmax(트랙의 최솟값, 트랙의 최댓값) 형태로 사용한다.

자주 쓰이는 속성

- length: 음수의 아닌 길이값이다.
- percentage: 그리드 컨테이너의 블록 크기에 상대적인 음이 아닌 백분율이다.
- flex: 그리드 트랙을 숫자의 비율대로 나눈다.
- max-content: 그리드 트랙을 차지하는 최대 콘텐츠 범위이다.
- min-content: 그리드 트랙을 차지하는 최소 콘텐츠 범위이다.
- auto: 그리드 트랙을 차지하는 최대 콘텐츠 범위이다. (max-content 결과와 동일하다.)

```
minmax(100px, 1fr)
minmax(200px, 50%)
minmax(30%, 100px)
minmax(100px, max-content)
minmax(min-content, 50%)
minmax(max-content, auto)
```

grid-template-columns의 영역을 px 단위로 열의 너비를 설정할 수 있다. 화면 전체 크기를 변경하면 동시에 열의 너비가 늘어나거나 줄어들기도 한다. 아래 예시는 최소 30px에서 최대 300px 사이에서만 반응한다.

```
.container {
    display: grid;
    grid-template-columns: repeat(3, minmax(30px, 300px)); /* 최소30px~최대300px */
}
```



grid-template-columns의 영역을 % 단위와 fr 단위로 열의 너비를 설정할 수 있다. 아래 예시는 첫 번째 열의 너비가 최소 50%에서 최대 90% 비율로 늘어나며 두 번째 세 번째 열은 나머지 공간을 균등하게 나누어 가진다.

```
.container {  
    display: grid;  
    grid-template-columns: minmax(50%, 90%) 1fr 1fr;  
}
```

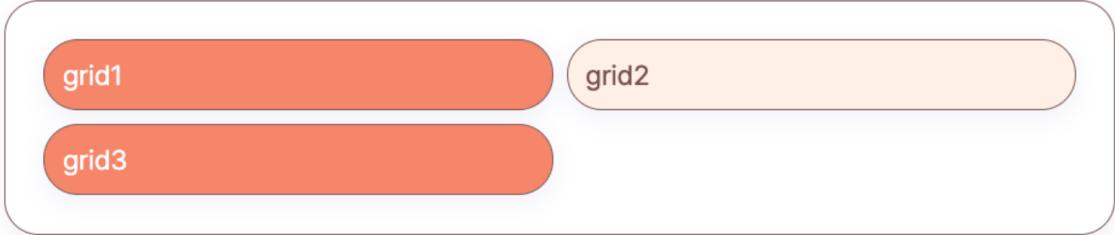
max 값을 auto로 입력하면 그리드 영역 범위를 최대 크기의 영역으로 유연하게 늘어나도록 할 수 있다.

```
.container {  
    display: grid;  
    grid-template-columns: repeat(3, minmax(30px,auto)); /* 최소 30px */  
}
```

minmax(auto, max-content)를 사용하면 그리드 아이템 콘텐츠 크기에 맞추어 영역이 설정된다.

```
.container {  
    display: grid;  
    grid-template-columns: repeat(3, minmax(auto,max-content));  
}
```

auto-fill과 사용하여 반응형 그리드 영역을 만들 수 있다. minmax를 사용하여 최소 200px의 영역을 유지하고 1fr의 길이는 전체 너비의 따라 반응한다. auto-fill은 설정된 너비에서 가능한 많은 영역을 만들어낸다.



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

2.4. grid-auto-rows , grid-auto-columns

grid-item이 사용자가 명시적으로 지정해준 grid-template-columns 또는 grid-template-rows에서 벗어난 위치에 존재하게 될 때, 트랙의 크기를 암시적으로 지정하는 속성이다. 쉽게 말하자면, grid 행(rows)/열(columns) 트랙 크기를 자동으로 설정한다.

grid-auto-rows

grid item이 grid-template-rows로 지정한 명시적 행 외부에 존재하는 경우 암시적 행의 크기가 적용된다. 앞으로 추가될 수 있는 모든 행의 높이를 정해주고 싶다면 grid-auto-rows를 적용해주면 된다.

grid-auto-columns

grid item이 grid-template-columns로 지정한 명시적 열 외부에 존재하는 경우 암시적 열의 크기가 적용된다. 암시적 크기가 적용된 행과 열은 오직 양수만 사용할 수 있다 (음수는 사용이 불가하다)

기본적으로 grid에서 각 행의 높이는 지정해주지 않는 이상 콘텐츠의 크기를 가지기 때문에 각각 다르다.



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

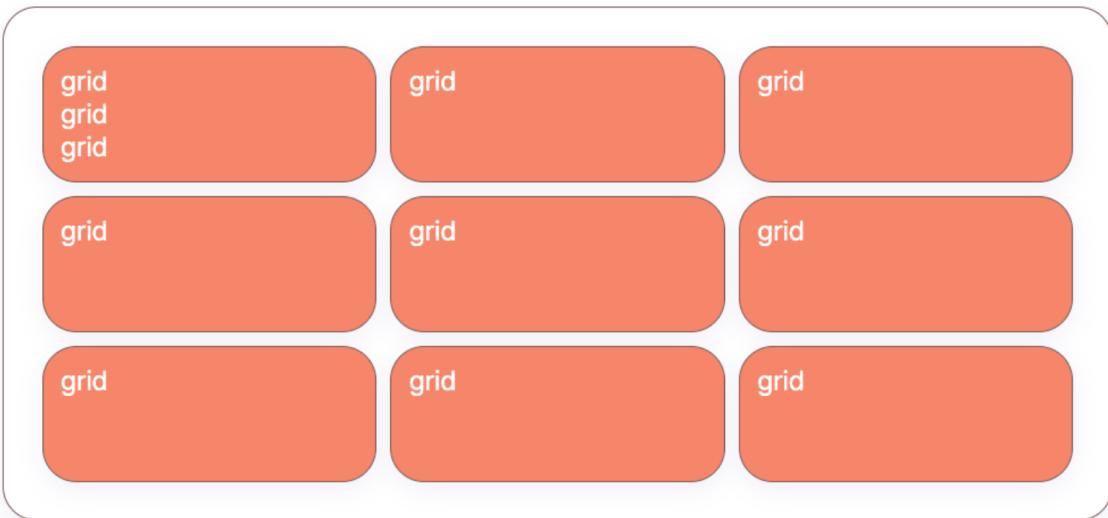
각 행의 높이를 같게 만들기 위해서 grid-template-row 속성을 이용해 여러 가지 여러 가지 방식으로 높이 값을 지정해줄 수 있지만, 이 높이 값은 '앞으로 추가될 아이템'에는 적용되지 않는다. grid-template-rows는 현재 정의되고 있는 행의 높이만을 설정할 뿐이기 때문이다. 아이템이 더 추가된다면 행의 높이가 달라지고, 그럼 grid-template-rows를 다시 한번 수정해야 한다는 번거로움이 있다.



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

행과 열이 하나씩 추가될 때마다 늘어나는 1fr, 1fr, 1fr, 1fr...

이 문제를 해결하기 위해 grid-auto-rows를 써주면 콘텐츠가 늘어날 때마다 CSS에 row의 개수를 수 정해 줄 필요 없이 자동으로 값이 적용되게 된다.



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

minmax 프로퍼티를 사용하여 값을 지정해줄 수도 있다.

```
.container {  
    display: grid;  
    grid-auto-rows: minmax(100px, auto);  
}
```

3. Gap

3.1. gap이란?

Gap은 그리드 컨테이너와 아이템 요소들의 간격을 설정하는 속성이다. gap 속성값에는 우리가 사용하는 각종 단위가 들어갈 수 있다. 길이를 나타내는 단위인 em, rem, px, vmin, vmax나 퍼센티지(%)를 사용 가능하며, calc() 함수를 이용하여 계산된 값도 사용할 수 있다. 하지만 fr 단위는 사용이 불가하다는 점에 주의해야 한다.

```
/*단일 길이값*/
gap: 10px;
gap: 1em;
gap: 3vmin;

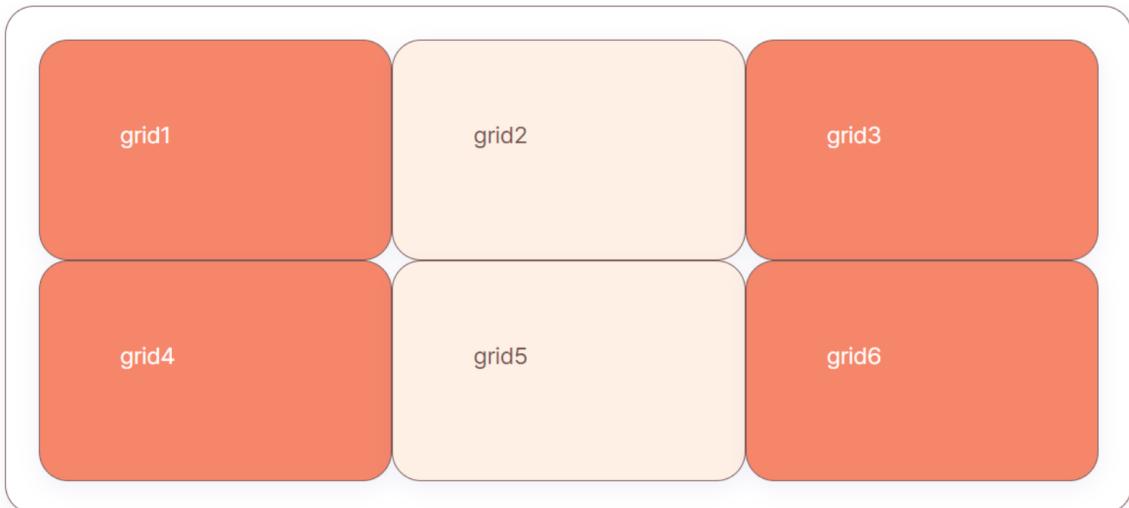
/*단일 퍼센티지값*/
gap: 10%;

/*이중 값*/
gap: 20px 10px;
gap: 10% 20%;

/*calc() 값*/
gap: calc(20px+10%);
```

fr 단위가 gap의 속성값이 될 수 없는 이유는, fr은 사용 가능한 공간을 분배하는 단위이기 때문이다. 아래 예시처럼 `grid-template-columns: 1fr 1fr 1fr;`라는 속성을 컨테이너에 적용하면, 컨테이너의 사용 가능한 넓이를 고려하여 1:1:1의 비율로 분배하는 것이 fr의 역할이다. 하지만, grid item 요소들 사이에는 사용 가능한 공간이 존재하지 않고, 서로 붙어있는 상태이다. 따라서 아이템 간의 간격을 벌리는 속성인 gap에 fr 단위가 적용되지 않는 것이다.

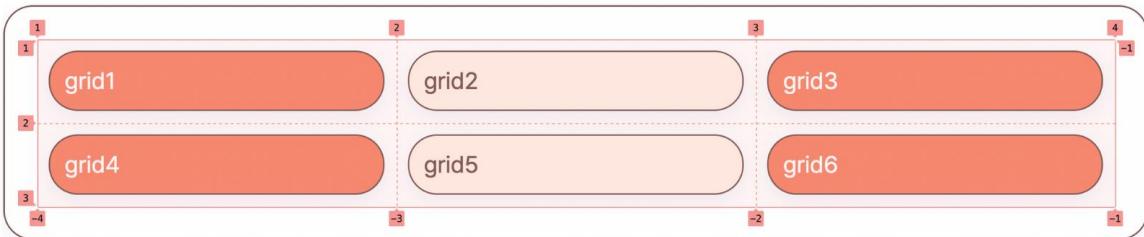
아래 예제는 `gap:2fr` 을 그리드 컨테이너에 부여하였지만 적용되지 않는 모습이며, 개발자 도구에서도 마찬가지로 fr이 잘못된 속성값이라 표시되는 것을 확인할 수 있다.



```
.container { grid-gap.html:46
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  padding: ▶ 22px;
  ▲ gap: 2fr;
}
```

3.2. margin과 gap의 차이점

margin의 경우 인접한 요소의 존재와 상관없이 스타일이 적용되 불필요한 공간을 만든다.



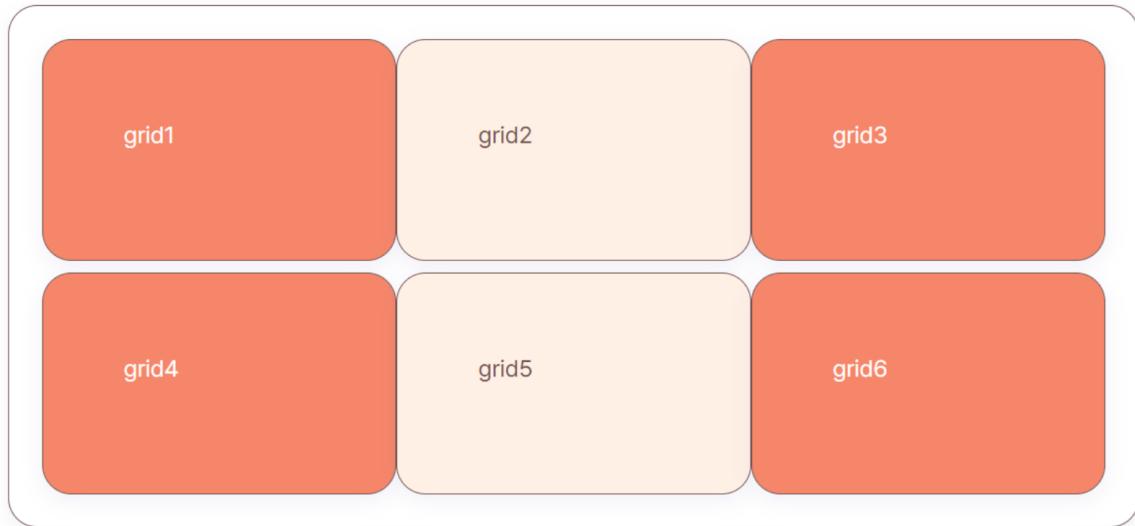
gap은 인접한 요소가 있을때만 사이에 공간을 만들어 불필요한 공간을 만들지 않는다.



3.3. gap의 종류

row-gap

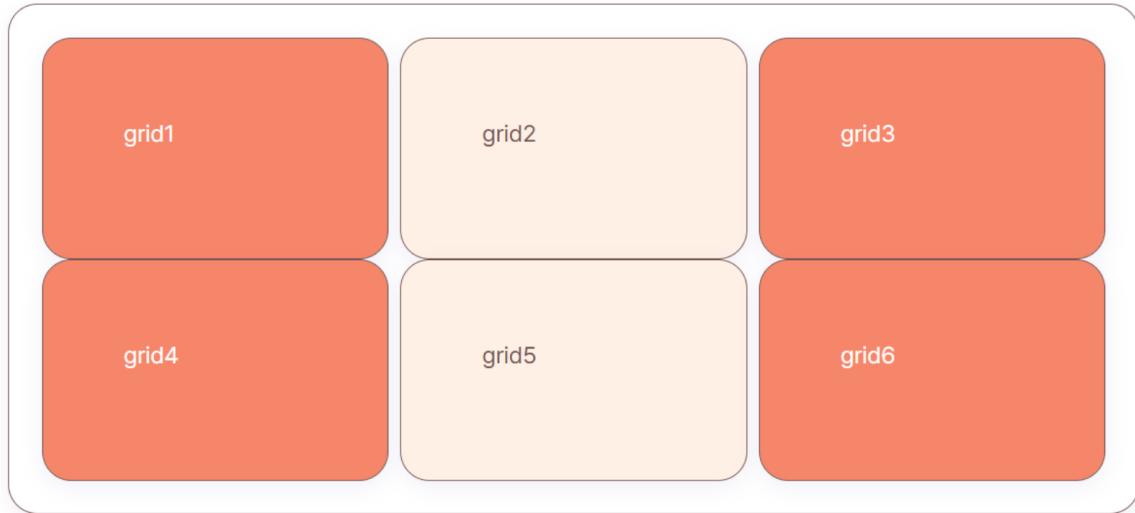
grid 셀의 행 사이의 간격을 설정하는 속성이다.



```
.container {  
  display: grid;  
  row-gap: 0.5rem;  
  grid-template-columns: 1fr 1fr 1fr;  
}
```

column-gap

grid 셀의 열 사이의 간격을 설정하는 속성이다.



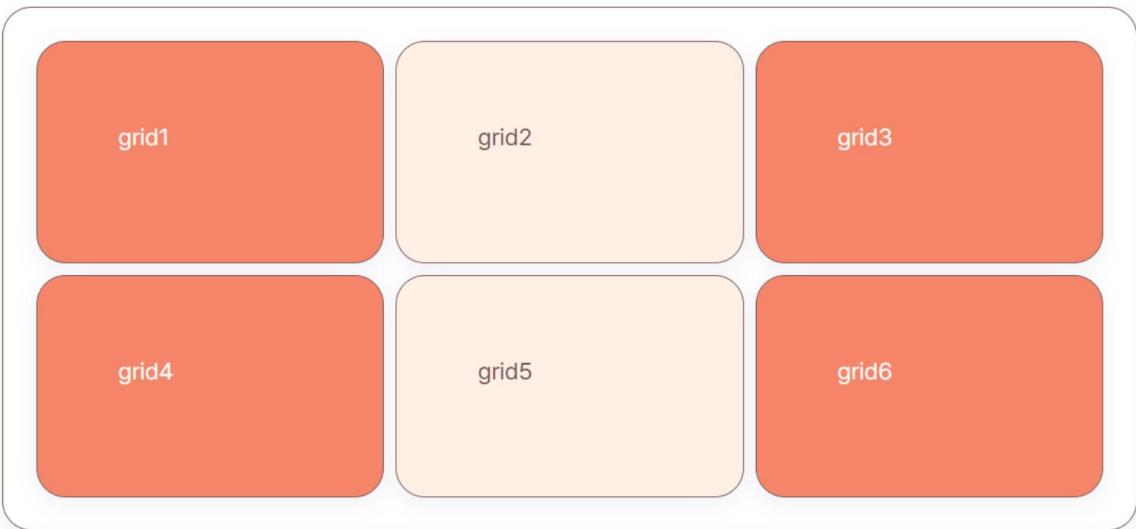
```
.container {  
  display: grid;  
  column-gap: 0.5rem;  
  grid-template-columns: 1fr 1fr 1fr;  
}
```

grid-gap

`grid-row-gap` 과 `grid-column-gap` 을 한 번에 작성할 수 있는 축약형이다.

1) grid-gap 단일 값

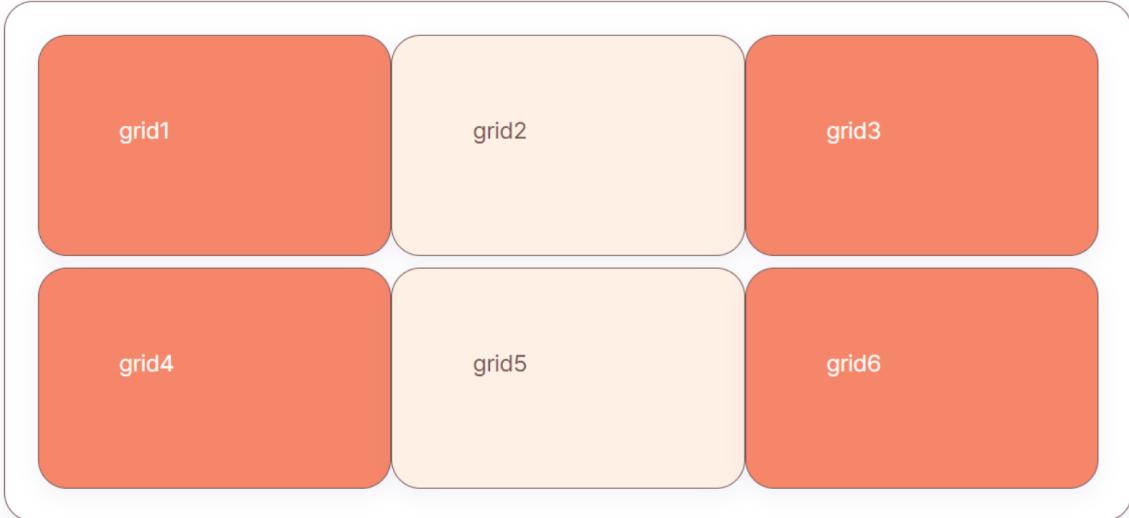
하나의 값으로 축약하여 row-gap과 column-gap을 동시에 같은 크기의 gap을 만들 수 있다. 사용이 편리하지만 row와 column에 각자 다른 크기나 단위를 지정할 수는 없다.



```
.container {  
    display: grid;  
    grid-template-columns: 1fr 1fr 1fr;  
    gap: 0.5rem;  
}
```

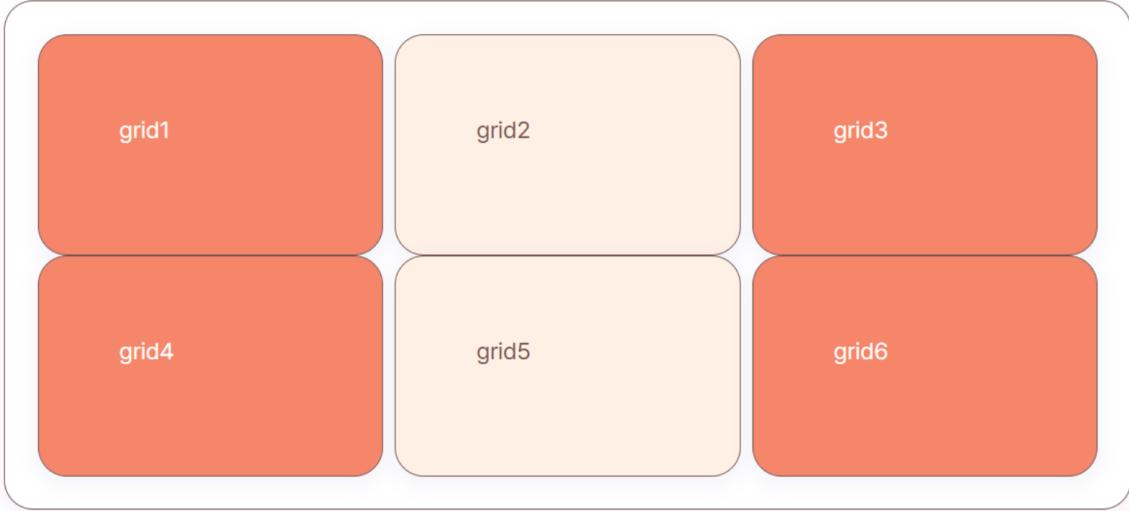
2) grid-gap 이중값

grid-gap에 부여하는 첫 번째 값은 row-gap, 두 번째 값은 column-gap이다. 각자 다른 크기와 단위를 섞어서 사용할 수 있다. 만약, row와 column에 같은 크기의 gap을 주고 싶다면 grid-gap 단일 값을 사용하는 동시에 적용하는 것이 더 효율적이다.



```
.container {  
    display: grid;  
    grid-template-columns: 1fr 1fr 1fr;  
    gap: 0.5 0rem;  
}
```

Grid_3 Gap



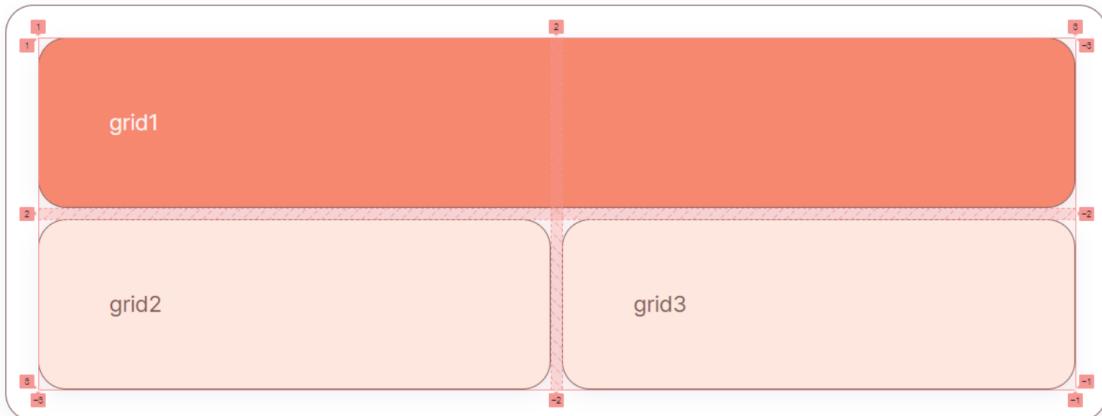
```
.container {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr;  
  gap: 0 0.5rem;  
}
```

4. 각 셀의 영역 지정

4.1. grid-column-start & grid-column-end

Grid 레이아웃에는 암묵적으로 grid 라인이 포함되어 있다. column 2개로 이루어진 grid 레이아웃을 상상해보도록 하겠다. column 라인은 앞에서부터 차례로 1,2,3번을 매길 수 있으므로 총 3개다.

- grid-column-start : column(열)시작의 라인 번호를 지정해준다.
- grid-column-end : column(열) 마지막의 라인 번호를 지정해준다.



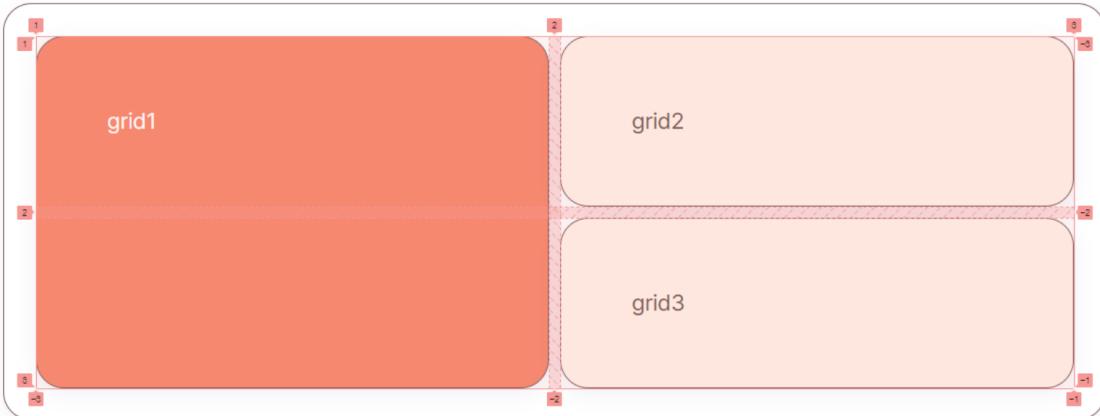
```
/*item*/
.grid1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
```

Grid 레이아웃에는 암묵적으로 grid 라인이 포함되어 있다. column 2개로 이루어진 grid 레이아웃을 상상해보도록 하겠다. column 라인은 앞에서부터 차례로 1,2,3번을 매길 수 있으므로 총 3개다.

4.2. grid-row-start & grid-row-end

다음은 grid-row-start & grid-row-end에 대해 알아보자. column과 사용 방법은 똑같으며 row의 영역을 지정할 수 있는 속성이다.

- grid-row-start : row의 시작 위치를 지정해준다.
- grid-row-end : row의 끝 위치를 지정해준다.

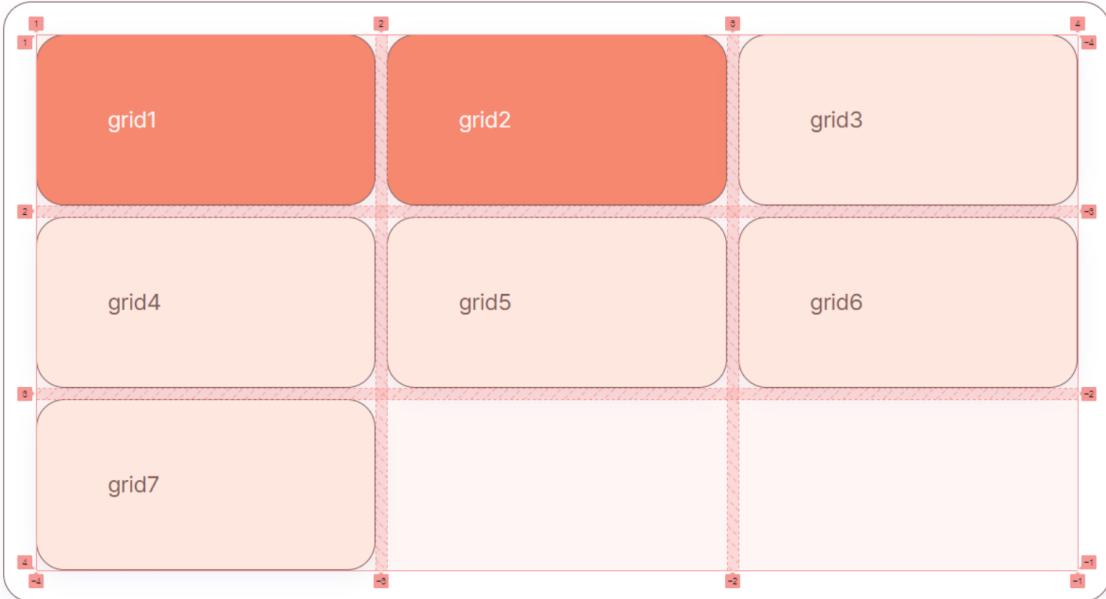


```
/* item */
.grid1 {
    grid-row-start: 1;
    grid-row-end: 3;
}
```

위 예제에서 grid1의 row 영역이 grid line의 1부터 3 까지 차지한 것을 볼 수 있다.

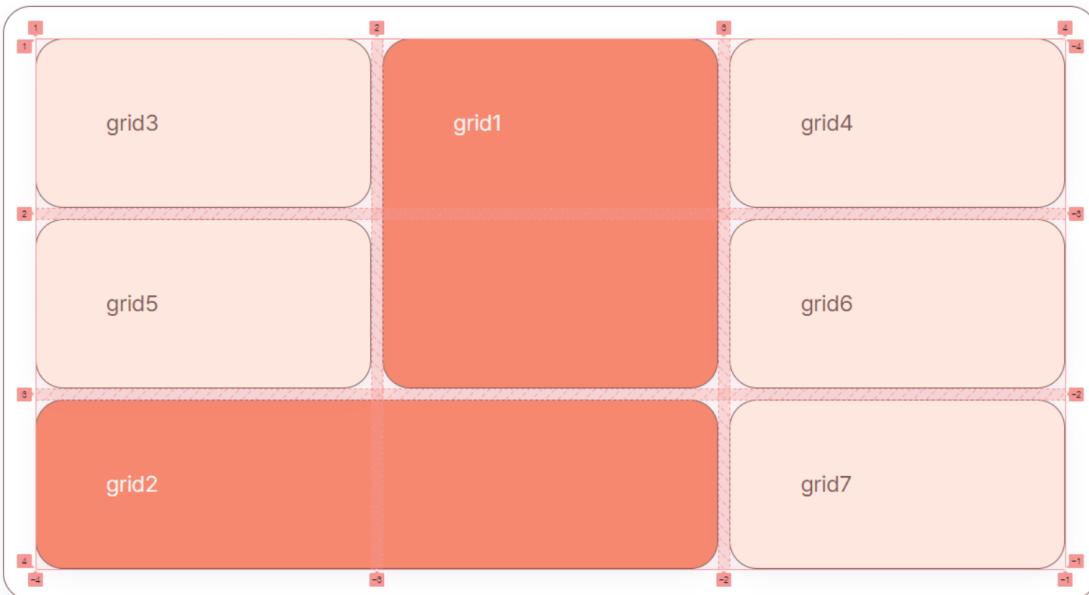
4.3. 셀의 영역 지정

이제 배운 속성들을 활용해서 아래 예제의 grid1 과 grid2의 영역을 지정해보겠다.



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

기본 컨테이너의 속성은 다음과 같이 주었다.



```
.grid1 {
    grid-column-start: 2;
    grid-column-end: 3;
    grid-row-start: 1;
    grid-row-end: 3;
}

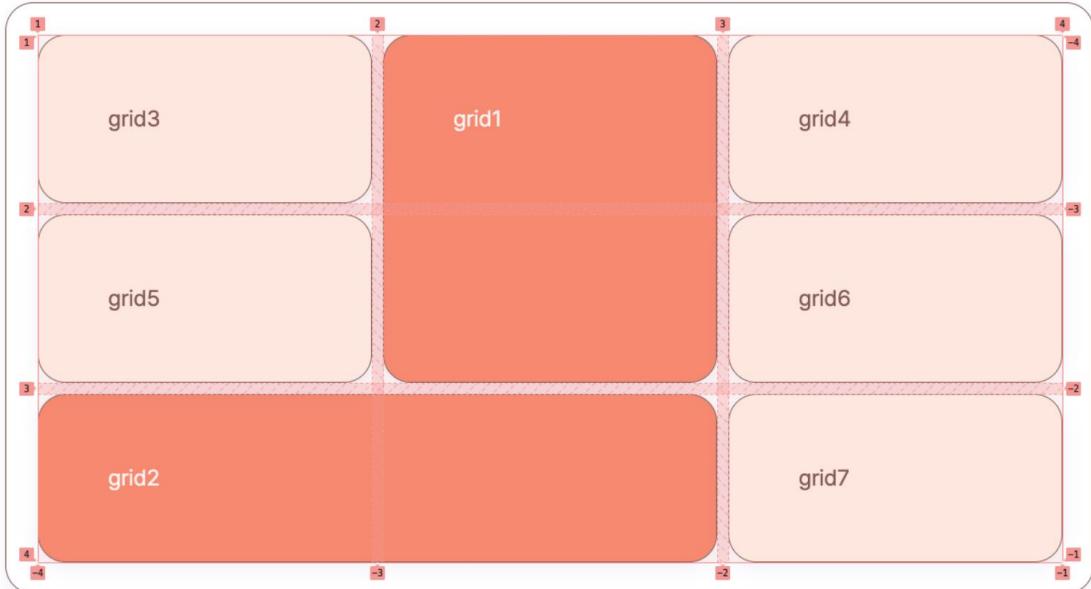
.grid2 {
    grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 3;
    grid-row-end: 4;
}
```

지금까지 배운 `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end` 속성을 사용하여 grid 셀의 영역을 지정할 수 있다. 하지만 이렇게 column과 row에 start와 end를 각각 지정해 주기에는 코드가 너무 길어지게 된다. start와 end를 한번에 지정 하려면 start와 end의 축약 속성인 `grid-column`, `grid-row`를 사용할 수 있다.

4.4. grid-column & grid-row

위에서 배운 `grid-column-start` 와 `grid-column-end` 는 `grid-column` 으로, `grid-row-start` 와 `grid-row-end` 는 `grid-row` 로 축약하여 start와 end를 한 번에 지정하여 사용할 수 있다.

- `grid-column` : 시작 / 끝 ;
- `grid-row` : 시작 / 끝 ;



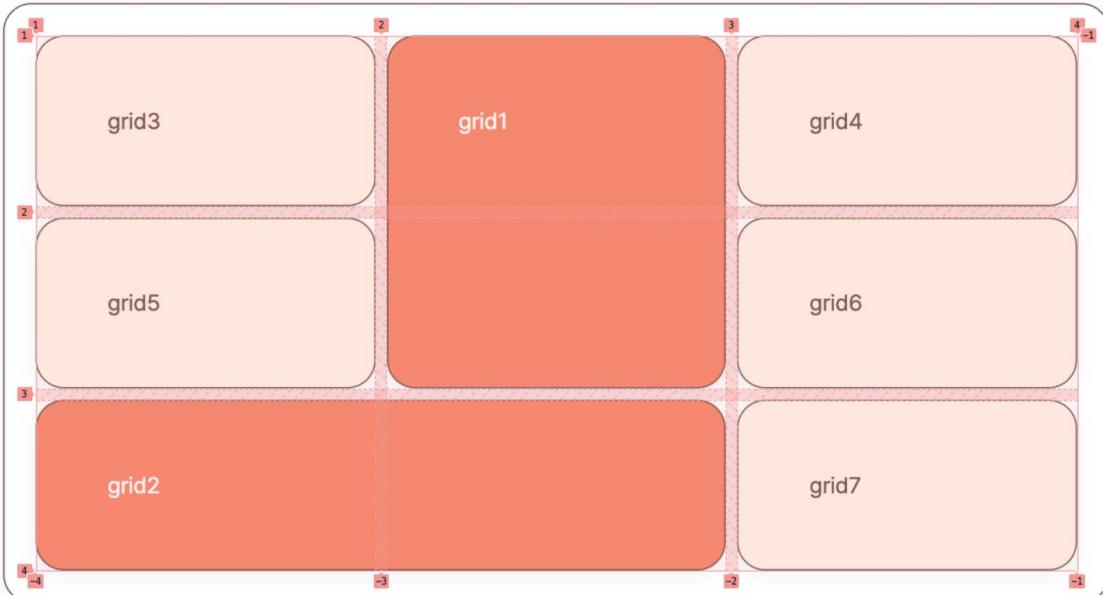
```
.grid1 {
    grid-column: 2 / 3;
    grid-row: 1 / 3;
}

.grid2 {
    grid-column: 1 / 3;
    grid-row: 3 / 4;
}
```

이렇게 `grid-column`, `grid-row` 속성을 사용하여 start와 end를 한 줄의 코드로 줄여 쓸 수 있게 되었다. 여기서 column과 row까지 한 번에 사용하고 싶다면 `grid-area` 속성을 사용하면 된다.

4.5. span 을 이용한 레이아웃 지정

Grid 에서 span은 아이템 요소에 쓰이는 속성이며, 몇개의 셀을 차지하게 할 것인지 지정해줄 수 있다. 숫자와 함께 쓰이며 이 숫자만큼 영역을 차지한다. 명시하지 않으면 span 1이 기본값이 된다.



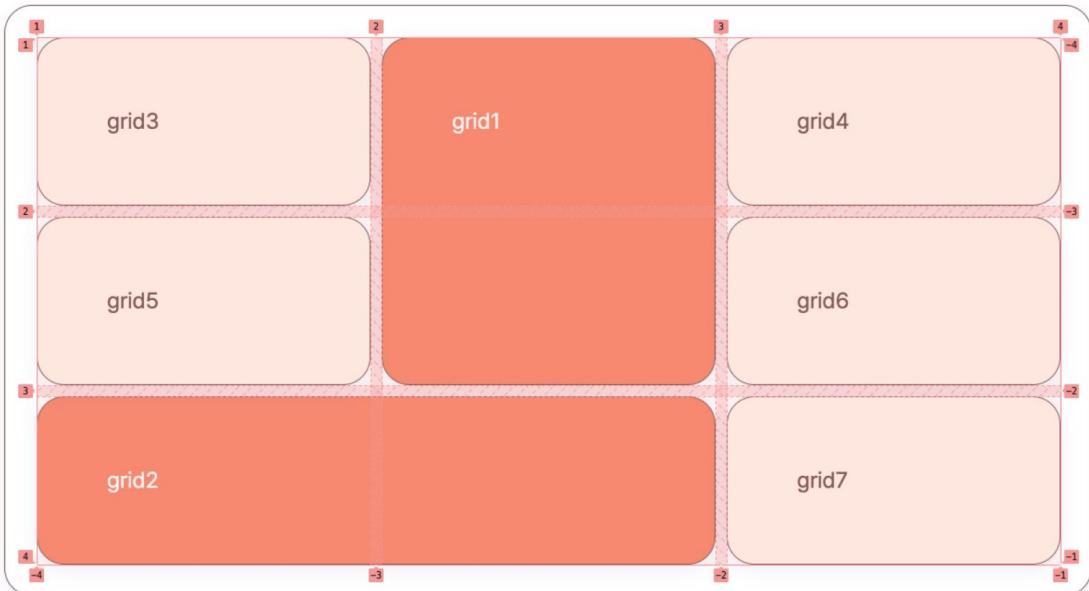
```
.grid1 {
    /* 2번 column 라인에서 시작해서 1칸 차지 */
    grid-column: 2 / 1 span;
    /* 1번 row 라인에서 시작해서 2칸 차지 */
    grid-row: 1 / 2 span;
}

.grid2 {
    /* 1번 column 라인에서 시작해서 2칸 차지 */
    grid-column: 1 / 2 span;
    /* 2번 row 라인에서 시작해서 1칸 차지 */
    grid-row: 3 / 1 span;
}
```

4.6. grid-area

grid 컨테이너 내의 item에 적용하는 속성이며, 아이템을 그리드에 배치할 때 사용할 수 있는 방법의 하나이다.

- `grid-row-start`, `grid-column-start`, `grid-row-end`, `grid-column-end`의 단축 속성이며, column과 row를 한 번에 지정할 수 있는 장점이 있다.
- 형태: `grid-area: grid-row-start / grid-column-start / grid-row-end / grid-column-end`
- 아래 예시의 grid1의 `grid-area`는 `grid-row: 1 / 3`, `grid-column: 2 / 3`과 동일하다.
- 아래 예시의 grid2의 `grid-area`는 `grid-row: 3 / 4`, `grid-column: 1 / 3`과 동일하다.



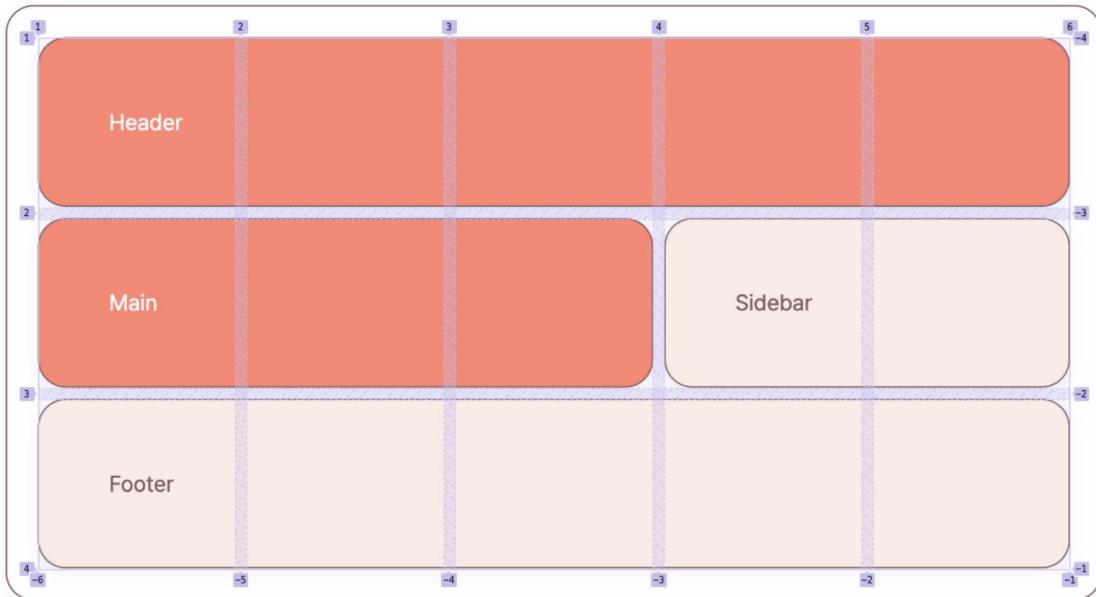
```
.grid1 {
    grid-area: 1 / 2 / 3 / 3;
}

.grid2 {
    grid-area: 3 / 1 / 4 / 3
}
```

4.7. grid-template-areas

`grid-template-areas` 는 그리드 내 각 영역(Grid Area)에 이름을 붙여 배치하는 매우 직관적인 방법이다.

각각의 아이템 요소가 차지하는 셀의 개수만큼 원하는 위치에 이름을 적어 영역을 지정해주면 지정한 자리만큼의 셀을 차지한다.



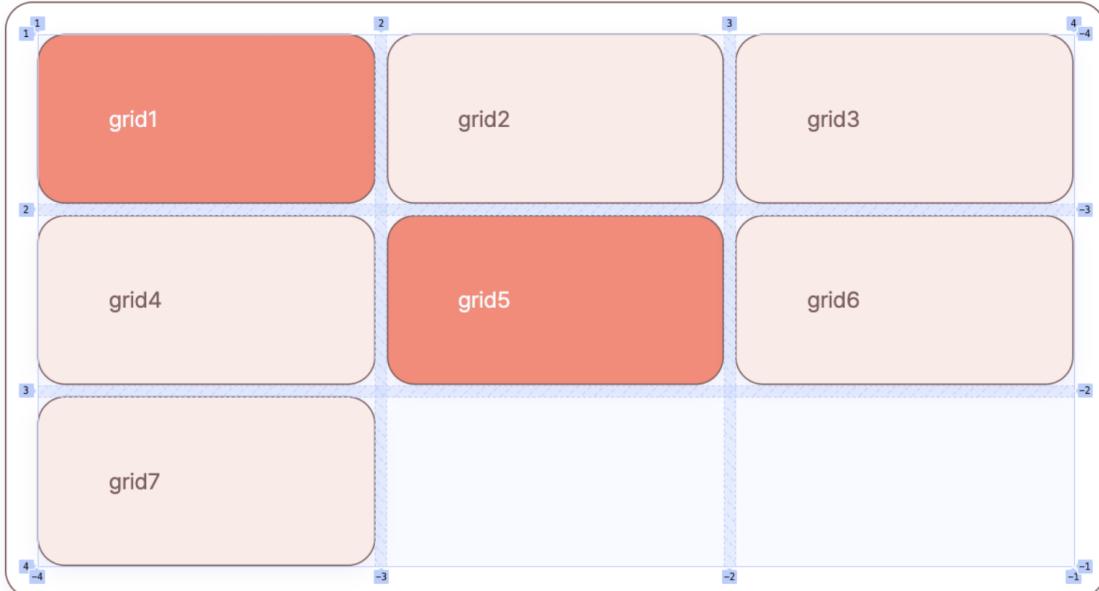
```
.container {  
    display: grid;  
    grid-template-columns: repeat(5, 1fr);  
    grid-template-rows: repeat(3, 1fr);  
    gap: 0.5rem;  
    grid-template-areas:  
        "header header header header header"  
        "main main main side side"  
        "footer footer footer footer footer";  
}  
  
.header {  
    grid-area: header;  
}  
  
.main {  
    grid-area: main;  
}  
  
.sidebar {  
    grid-area: side;  
}  
  
.footer {  
    grid-area: footer;  
}
```

- 각 아이템의 구분은 공백으로 하며 빈 영역을 만들고 싶다면 `.` (마침표)와 `none` 으로 빈 영역을 표시해줄 수 있다.

4.8. grid line 외의 추가적인 방법

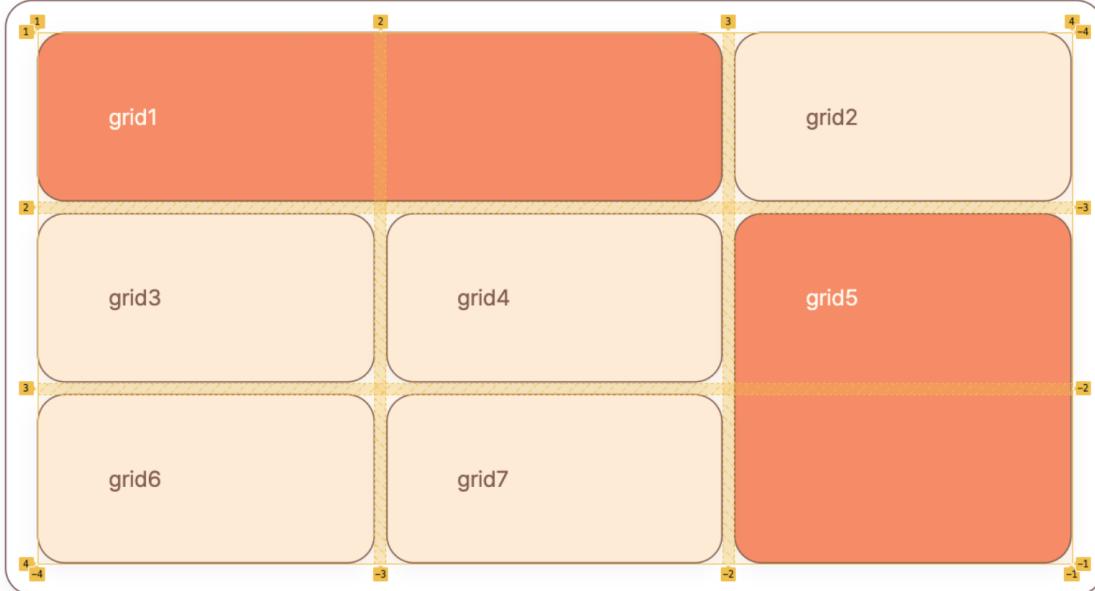
Grid name을 이용한 레이아웃 지정

그리드 라인은 암묵적으로 그리드 라인 숫자가 지정되어 있는데, 이 라인에 이름을 지정할 수 있다.



```
.container{  
    display: grid;  
    grid-template-columns: [col-1] 1fr [col-2] 1fr [col-3] 1fr [col-4];  
    /* grid-template-columns: 1fr 1fr 1fr 1fr 과 같다. */  
    grid-template-rows: [row-1] 1fr [row-2] 1fr [row-3] 1fr [row-4];  
    /* grid-template-rows: 1fr 1fr 1fr 1fr 과 같다. */  
}
```

각각의 라인에 이름을 정해주고 이 이름을 각 아이템 요소의 `grid-column-start`, `grid-column-end`, `grid-row-start`, `grid-row-end` 속성에 지정해주면 된다.



```
.grid1 {
    grid-row-start: row-1;
    grid-row-end: row-2;
    grid-column-start: col-1
    grid-column-end: col-3;
}

.grid5 {
    grid-row-start: row-2;
    grid-row-end: row-4;
    grid-column-start: col-3;
    grid-column-end: col-4;
}
```

쉽게 생각하면 숫자 대신 이름을 넣어준다고 보면 된다.

5. IE지원을 위한 Grid

아래와 같이 -ms-prefix를 붙여 IE지원을 해줄 수 있다.

- `display: grid` → `display: -ms-grid`
- `grid-template-rows` → `-ms-grid-rows`
- `grid-template-columns` → `-ms-grid-columns`

`repeat()` 속성은 다음과 같이 사용한다.

- `repeat(12, 1fr 20px)` → `(1fr 20px)[12]`

grid 속성 IE지원 대응표

속성	IE 지원
<code>display: grid;</code>	<code>display: -ms-grid;</code>
<code>grid-template-rows</code>	<code>-ms-grid-rows</code>
<code>grid-template-columns</code>	<code>-ms-grid-columns</code>
<code>grid-row-start</code>	<code>-ms-grid-row</code>
<code>grid-column-start</code>	<code>-ms-grid-column</code>
<code>align-self</code>	<code>-ms-grid-row-align</code>
<code>justify-self</code>	<code>-ms-grid-column-align</code>
<code>grid-row: 1 / span 2;에서 span 2 대신</code>	<code>-ms-grid-row-span</code>
<code>grid-column: 1 / span 2;에서 span 2 대신</code>	<code>-ms-grid-column-span</code>

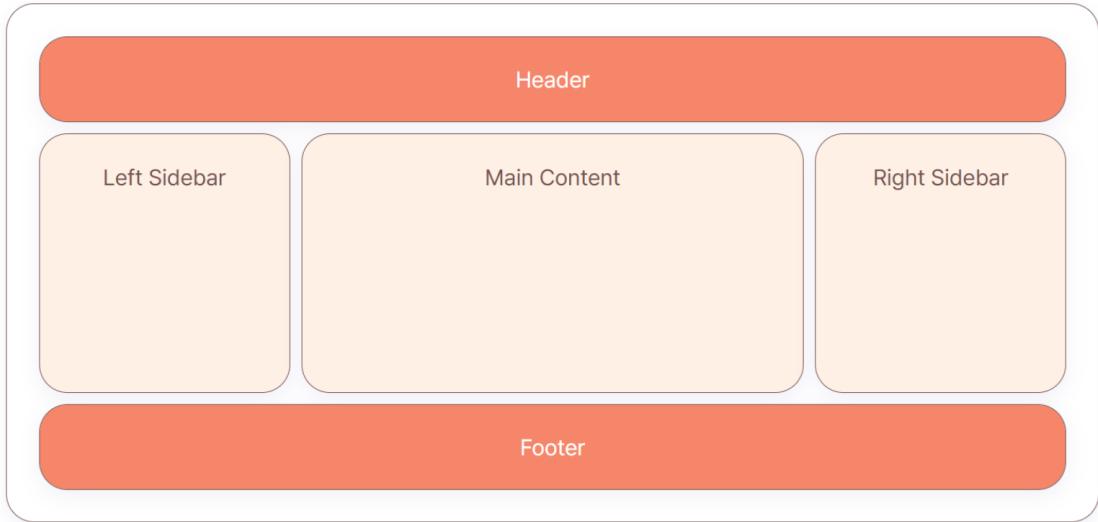
IE에서 주의해야 할 auto-placement

IE에서는 아래와 같은 grid의 auto-placement를 지원하지 않는다. IE에서는 grid의 자동 배치가 작동하지 않기 때문이다. prefix 사용도 불가능하다.

- `grid-auto-columns`
- `grid-auto-rows`
- `grid-auto-flow`

1) 기본 성배 레이아웃

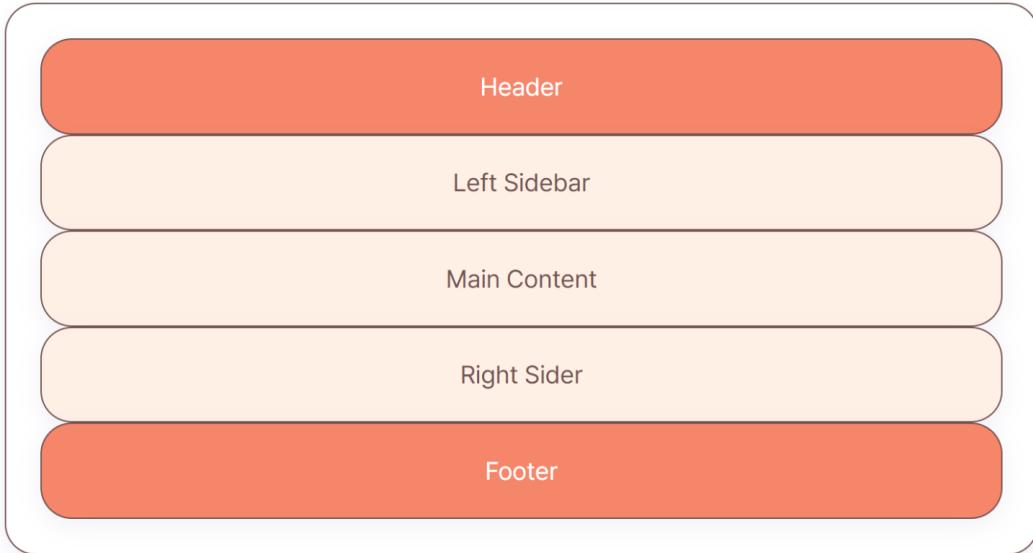
`grid-area` 속성을 사용하여 기본 성배 레이아웃을 만들어 보려고 한다.



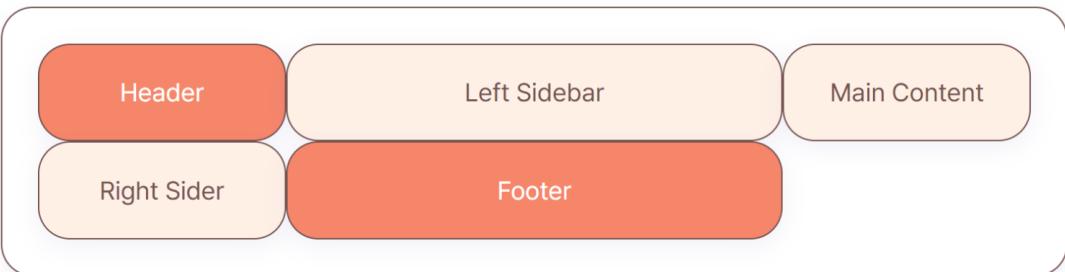
```
<div class="container">
  <div class="header">Header</div>
  <div class="left-sidebar">Left Sidebar</div>
  <div class="main-content">Main Content</div>
  <div class="right-sider">Right Sidebar</div>
  <div class="footer">Footer</div>
</div>
```

```
.container {  
    display: grid;  
    grid-template-columns: 1fr 2fr 1fr;  
    grid-template-rows: 1fr 3fr 1fr;  
    padding: 22px;  
    gap: 0.5rem;  
}  
  
/* layout */  
  
.header {  
    grid-area: 1/1/2/4;  
}  
  
.left-sidebar {  
    grid-area: 2/1/3/2;  
}  
  
.main-content {  
    grid-area: 2/2/3/3;  
}  
  
.right-sider {  
    grid-area: 2/3/3/4;  
}  
  
.footer {  
    grid-area: 3/1/4/4;  
}
```

grid 속성값들을 사용하지 않았을 때 아이템들은 HTML 마크업 순서대로 배치가 된다.



그리드가 적용되는 상위 부모요소 container에 `display:grid` 속성을 추가한다. 또한 `grid-template-columns: 1fr 2fr 1fr;` 을 주어 1:2:1 비율로 3개의 column을 만든다.



```
.container {
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  padding: 22px;
}
```

`grid-template-rows: 1fr 3fr 1fr;` 를 주는데, 위의 `grid-template-columns` 와 같이 1:3:1 비율로 3개의 row를 만든다.



```
.container {  
    display: grid;  
    grid-template-columns: 1fr 2fr 1fr;  
    grid-template-rows: 1fr 3fr 1fr;  
    padding: 22px;  
}
```

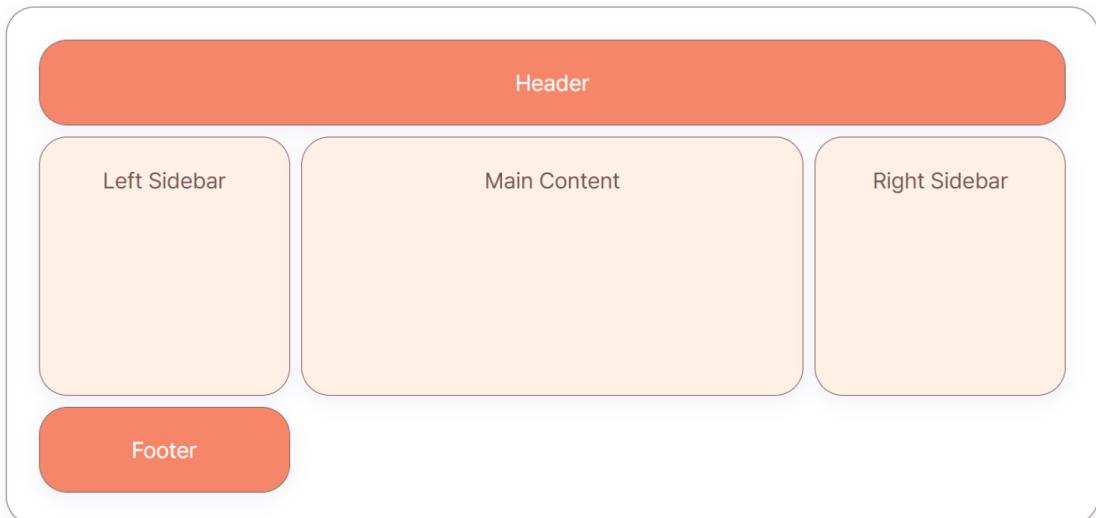
`gap: 0.5rem;` 을 주어 아이템 사이의 간격을 설정해준다.



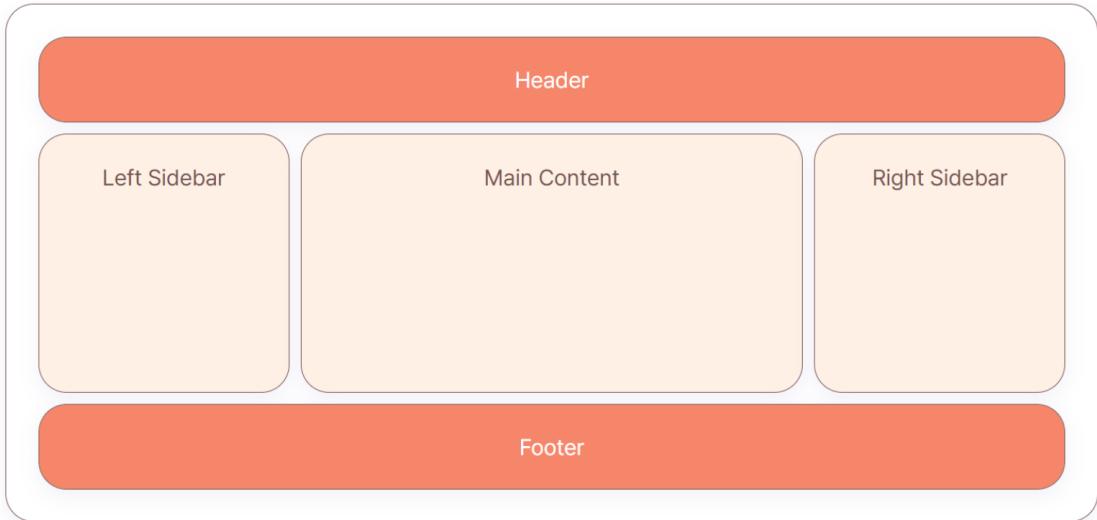
```
.container {  
    display: grid;  
    grid-template-columns: 1fr 2fr 1fr;  
    grid-template-rows: 1fr 3fr 1fr;  
    gap: 0.5rem;  
    padding: 22px;  
}
```

각 아이템마다 `grid-area` 속성을 주어 영역을 정한다

- header의 row 영역은 1번째 라인부터 2번째 라인까지, column 영역은 1번째 라인에서 4번째 라인까지의 범위를 차지하게 한다.
- left-sidebar의 row 영역은 1번째 라인부터 2번째 라인까지, column 영역은 1번째 라인에서 4번째 라인까지 범위를 차지하게 한다.
- main-content의 row 영역은 2번째 라인부터 3번째 라인까지, column 영역도 2번째 라인에서 3번째 라인까지 범위를 차지하게 한다.
- right-sidebar의 row 영역은 2번째 라인부터 3번째 라인까지, column 영역은 3번째 라인에서 4번째 라인까지 범위를 차지하게 한다.



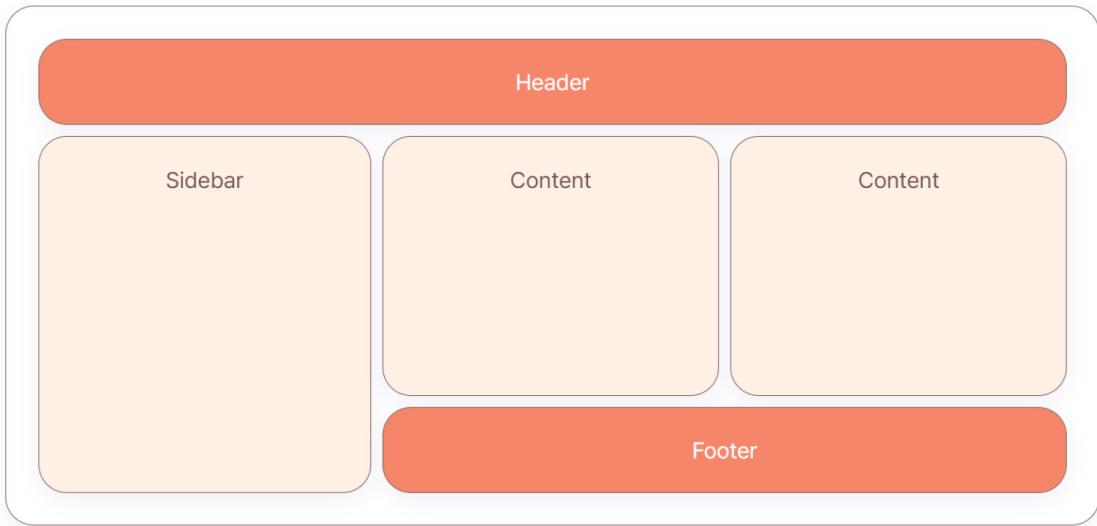
- footer의 row 영역은 3번째 라인부터 4번째 라인까지, column 영역은 1번째 라인에서 4번째 라인까지 범위를 차지하게 하여 레이아웃을 완성한다.



```
.header {  
    grid-area: 1/1/2/4;  
}  
  
.left-sidebar {  
    grid-area: 2/1/3/2;  
}  
  
.main-content {  
    grid-area: 2/2/3/3;  
}  
  
.right-sidebar {  
    grid-area: 2/3/3/4;  
}  
  
.footer {  
    grid-area: 3/1/4/4;  
}
```

2) 변형 성배 레이아웃

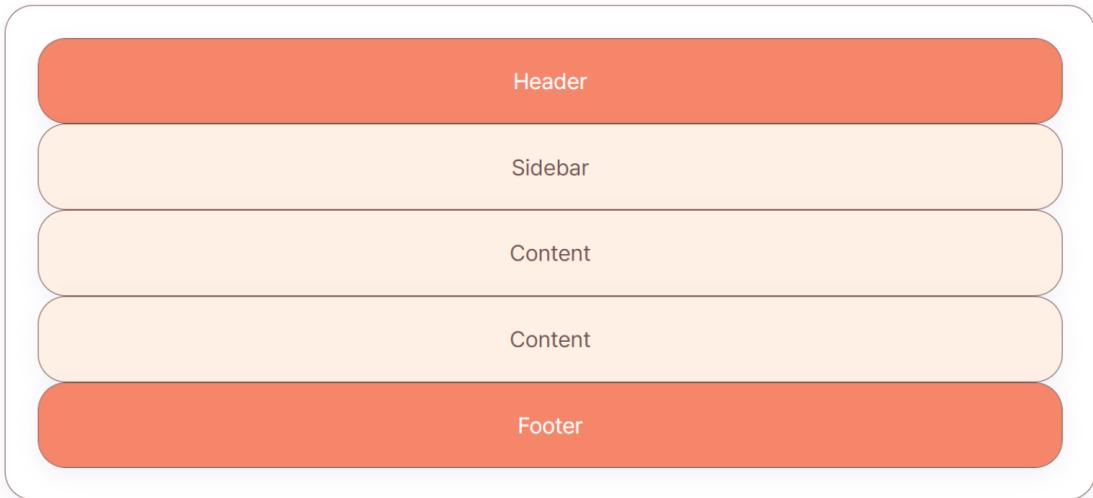
`grid-template-area` 속성을 사용하여 변형된 성배 레이아웃을 만들어 보려고 한다.



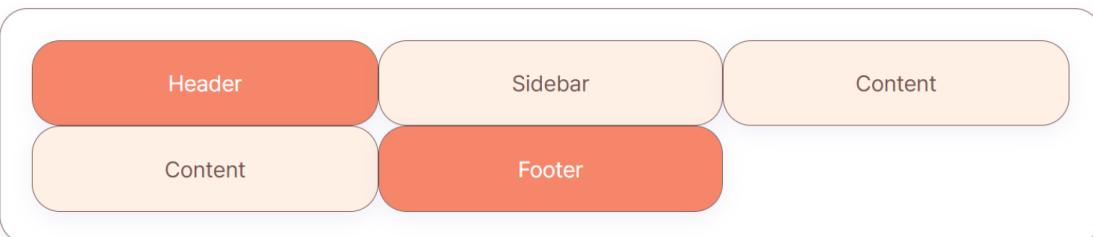
```
<div class="container">
    <div class="header">Header</div>
    <div class="sidebar">Sidebar</div>
    <div class="content1">Content</div>
    <div class="content2">Content</div>
    <div class="footer">Footer</div>
</div>
```

```
.container {  
    display: grid;  
    grid-template-areas:  
        "header header header"  
        "sidebar content1 content2"  
        "sidebar footer footer";  
    grid-template-columns: 1fr 1fr 1fr;  
    grid-template-rows: 1fr 3fr 1fr;  
    padding: 22px;  
    gap: 0.5rem;  
}  
  
/* layout */  
  
.header {  
    grid-area: header;  
}  
  
.sidebar {  
    grid-area: sidebar;  
}  
  
.content1 {  
    grid-area: content1;  
}  
  
.content2 {  
    grid-area: content2;  
}  
  
.footer {  
    grid-area: footer;  
}
```

grid 속성값들을 사용하지 않았을 때 아이템들은 HTML 마크업 순서대로 배치가 된다.

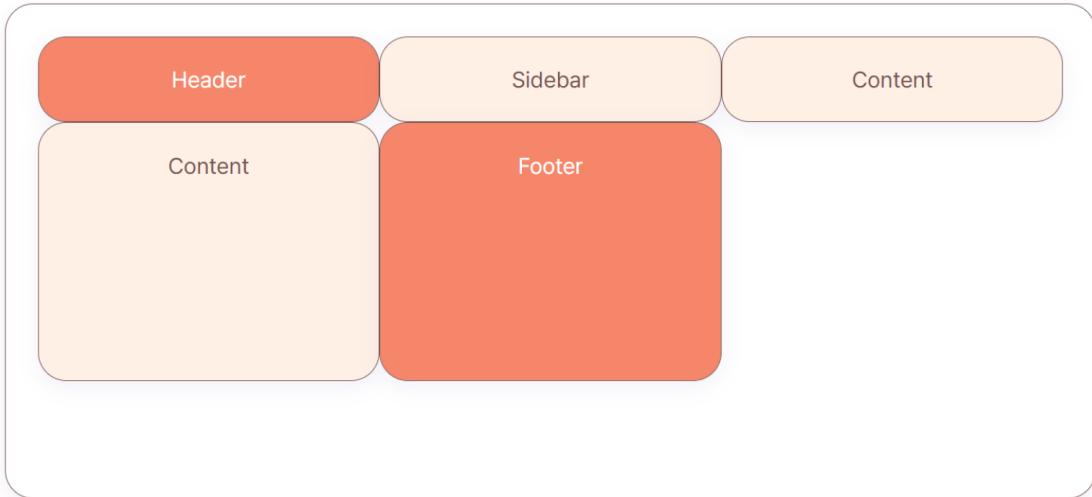


grid가 적용되는 컨테이너에 `display:grid` 속성을 추가한다. 또한 `grid-template-areas` 속성을 이용하여 변형된 성배레이아웃을 만들어볼 예정이므로 `grid-template-areas` 속성에 우리가 구성하고자 하는 레이아웃을 단어들로서 표현해준다.



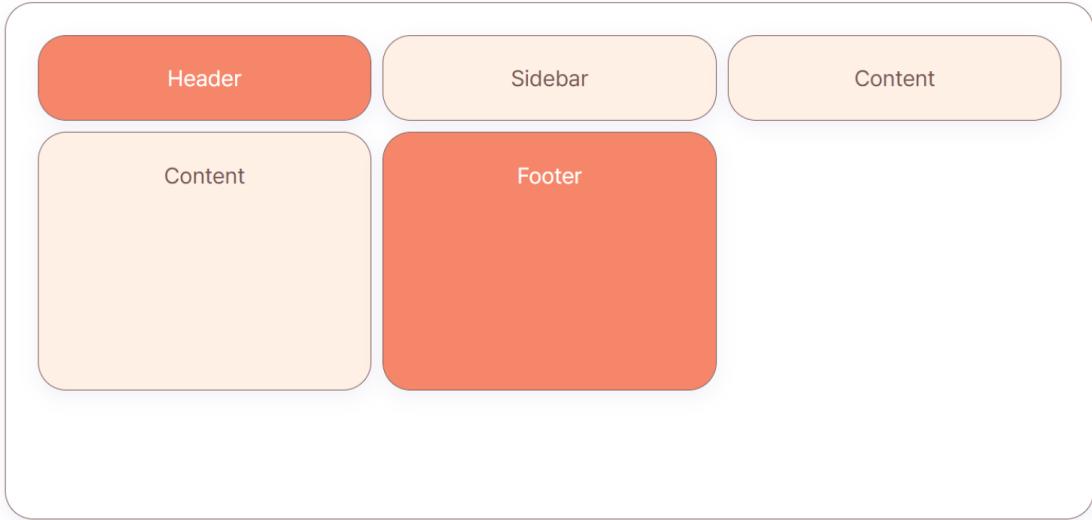
```
.container {  
    display: grid;  
    grid-template-areas:  
        "header header header"  
        "sidebar content1 content2"  
        "sidebar footer footer";  
    padding: 22px;  
}
```

현재 레이아웃은 임의의 3x3 공간을 가지고 있다고 생각하자. 이에 따라 컨테이너의 자식요소들의 너비를 설정해 준다. `grid-template-columns` 의 비율은 1:1:1, `grid-template-rows` 의 비율은 1:3:1로 설정하여 2번째 컬럼이자 컨텐츠 요소들의 비율을 키워보도록 작성한다.



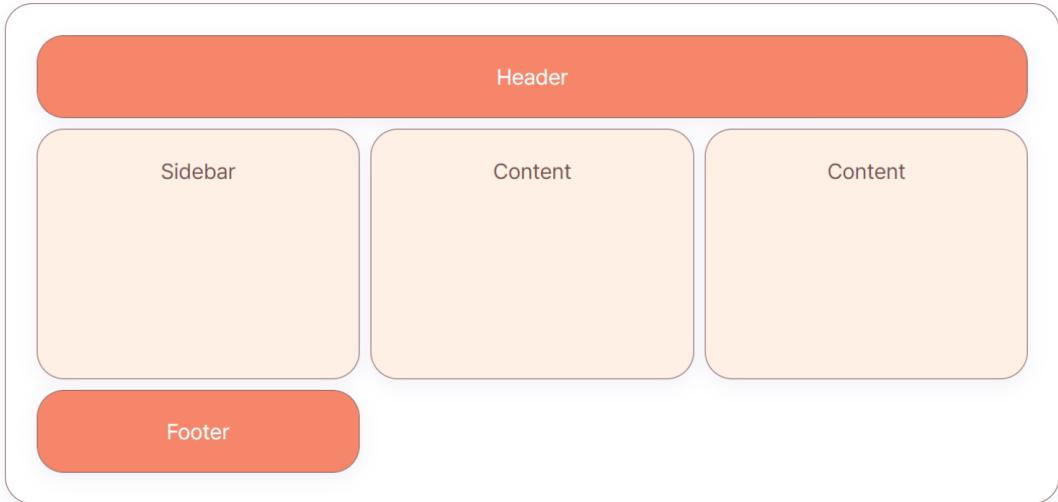
```
.container {  
    display: grid;  
    grid-template-areas:  
        "header header header"  
        "sidebar content1 content2"  
        "sidebar footer footer";  
    grid-template-columns: 1fr 1fr 1fr;  
    grid-template-rows: 1fr 3fr 1fr;  
    padding: 22px;  
}
```

gap: 0.5rem; 을 주어 아이템 사이의 간격을 설정해준다.



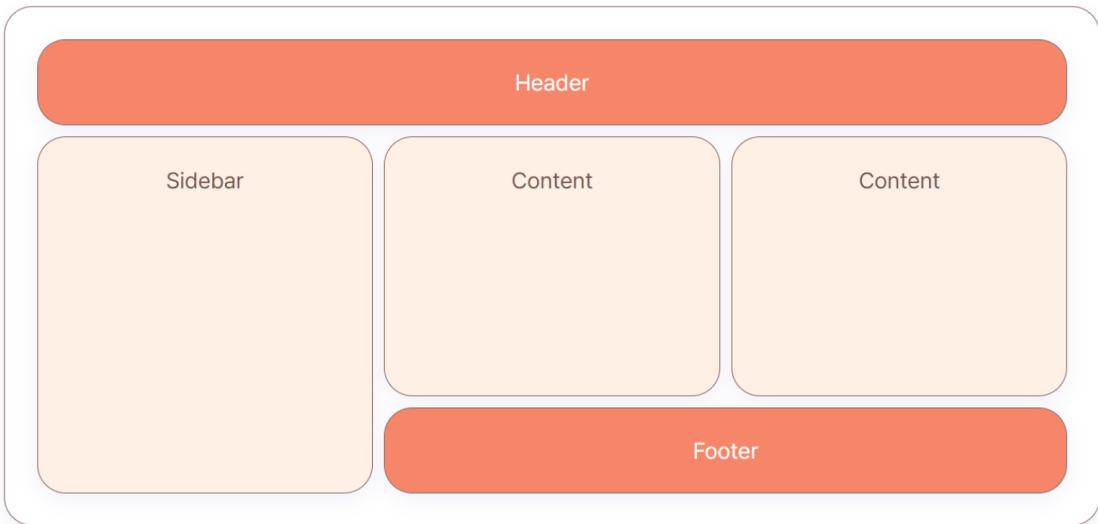
```
.container {  
    display: grid;  
    grid-template-areas:  
        "header header header"  
        "sidebar content1 content2"  
        "sidebar footer footer";  
    grid-template-columns: 1fr 1fr 1fr;  
    grid-template-rows: 1fr 3fr 1fr;  
    gap: 0.5rem;  
    padding: 22px;  
}
```

header 공간의 row는 header 만 차지하므로 `grid-area` 속성을 이용하여 `grid-template-area` 의 할당된 부분을 적용한다.



```
.header {grid-area: header;}
```

위와 마찬가지로 sidebar, content * 2, footer 부분을 나머지 `grid-template-area` 에 할당된 부분에 맞추어 구역을 지정한다.



```
.header {grid-area: header;}
.sidebar {grid-area: sidebar;}
.content1 {grid-area: content1;}
.content2 {grid-area: content2;}
.footer {grid-area: footer;}
```

제주코딩베이스캠프 로드맵



제주코딩베이스캠프 온라인 강의

<https://bit.ly/3Jf8NIH>

초판 1쇄 발행 | 2022. 06. 03

지은이 | 김민찬, 강혜진, 김상돈, 김 진, 김혜원, 김도희, 김태희, 김세훈, 김보람, 김유진, 김예지, 김희진, 류재준, 박누리, 박소현, 박유진, 신현수, 여소희, 윤수영, 이현섭, 이호준, 이수빈, 임홍렬, 임희래, 임다현, 장효순, 조미진, 조윤희, 전유진, 지다혜, 최수빈, 최원범, 차경림, 한재현, 허지현, 홍제섭

편 집 | 김 진, 차경림

총 팔 | 이호준

펴낸곳 | 사도출판

주 소 | 제주특별자치도 제주시 동광로 137 대동빌딩 4층

표지디자인 | 차경림

홈페이지 | <http://www.paullab.co.kr>

E - mail | paul-lab@naver.com

ISBN | 979-11-88786-59-6 (PDF)

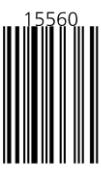
Copy right © 2022 by. 사도출판

이 책의 저작권은 사도출판에 있습니다. 저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

이 책에 대한 의견을 주시거나 오탈자 및 잘못된 내용의 수정 정보는 사도출판의 이메일로 연락을 주시기 바랍니다.

“**알아서 잘**
“**可以更好**”
하는 꿈하고
센스있게
정리하는
flex&grid
: flex가 grid 어렵다?

비매품/무료



9 791188 786596 ISBN 979-11-88786-59-6 (PDF)

