Responsive Web 문법

Media Queries

<Product by 안재욱>

Media Queries

Responsive Web 이란,

- 반응형웹 디자인을 기반으로 다양한 디바이스(PC, 태블릿PC, 스마트폰, 스마트TV 등)를 대응하는 웹
- 반응형 웹 (Responsive Web)은 기기의 해상도에 따라 레이아웃이 반응하여 웹페이지를 표현
- 하나의 소스로 다양한 디바이스의 해상도에 따라 다양한 레이아웃으로 표현 가능
- 반응형 웹은 특별한 프로그램을 적용하는 것이 아님.
- 반응형 웹은 자바스크립트로 통제하는 것이 아님.

주요 특징

- 모든 스마트 기기에서 접속 가능!
- 가로모드에 맞추어 레이아웃 변형 가능
- 사이트 유지/관리 용이

반응형 웹은 HTML과 CSS를 사용하여 기능 부여 가능

<Product by 안재욱>

Responsive Web 이란,

각각의 디바이스 서로 다른 크기를 갖고 있다.

그럼, 어떻게 맞출 것인가?

<Product by 안재욱>

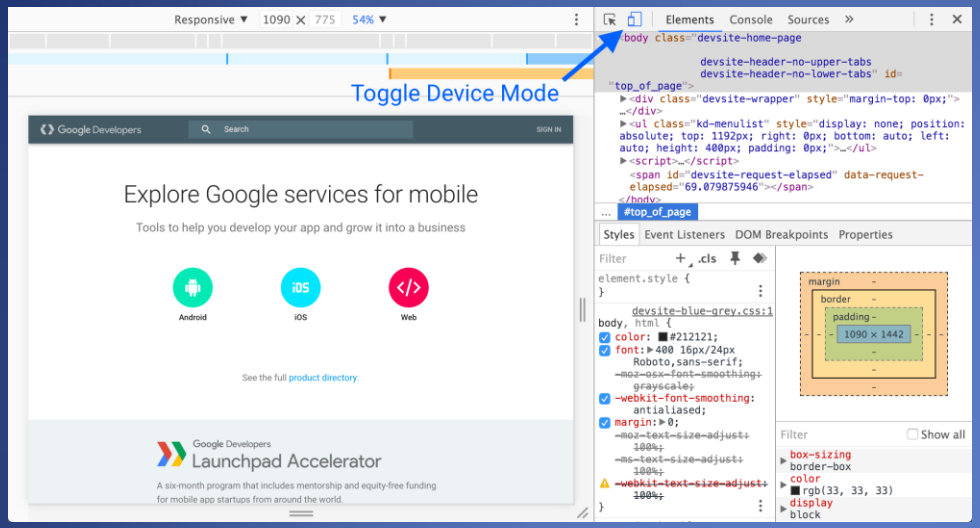
Responsive Web 을 들어가기 앞서

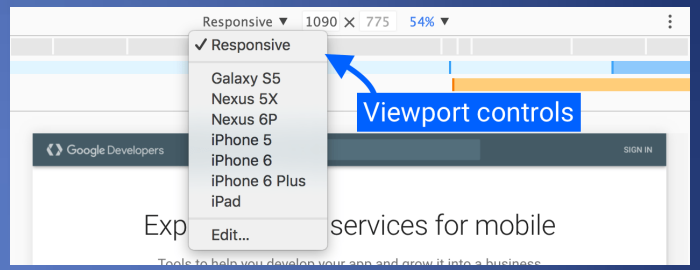
CSS3에 Media Query를 적용 + HTML5문법에서 코딩 추가 및 변형

<Product by 안재욱>

Responsive Web 보기 – 브라우저에서 확인

크롬 브라우저에서 확인





- 장점 : 최신 출시된 다양한 기기에 대한 해상도 및 적용 여부를 크롬 브라우저를 이용한 확인 가능. (국내 기기에 대한 부분은 업데이트가 매우 늦은 편)

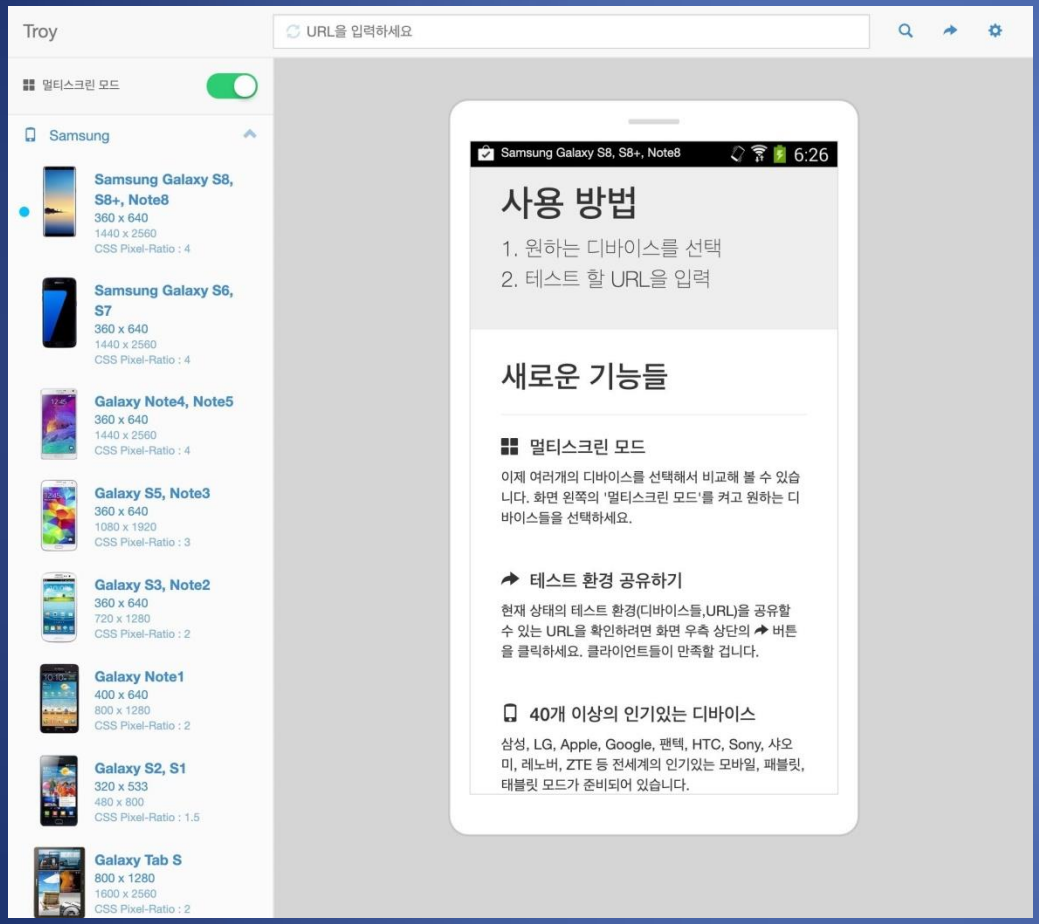
- 단점 : 일부 기능 작동 또는 css 설정간 반응형에 대한 버그(레이아웃 깨짐 또는 오작동)이 발생. 또한 멀티 디바이스 뷰에 대한 지원이 없기에 한가지 디바이스에 대한 화면 확인 가능

- 이를 보완하기 위해서는 반드시 서버 환경에서 모바일 브라우저 확인이 반드시 필요

<Product by 안재욱>

Responsive Web 보기 – 브라우저에서 확인

다음 트로이(http://troy.labs.daum.net/)에서 확인

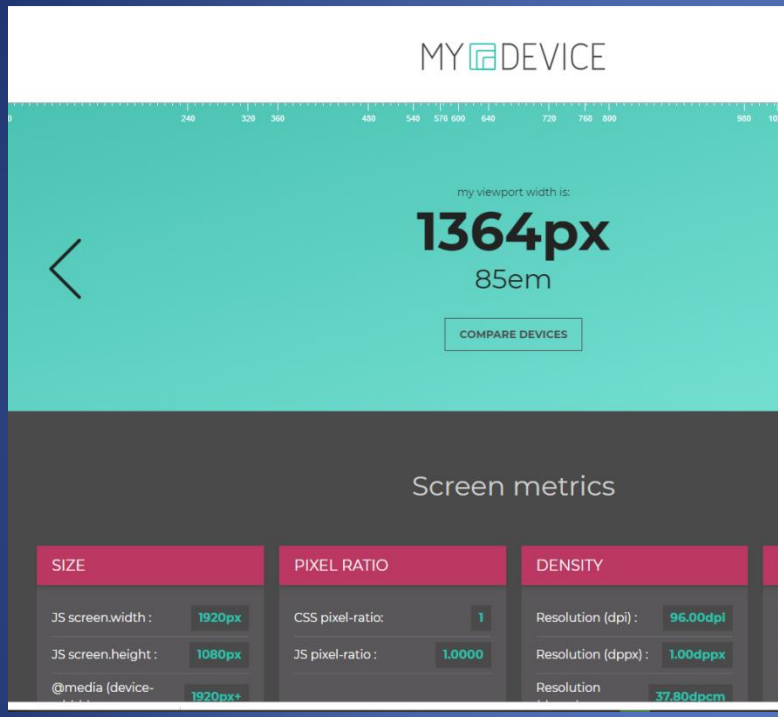


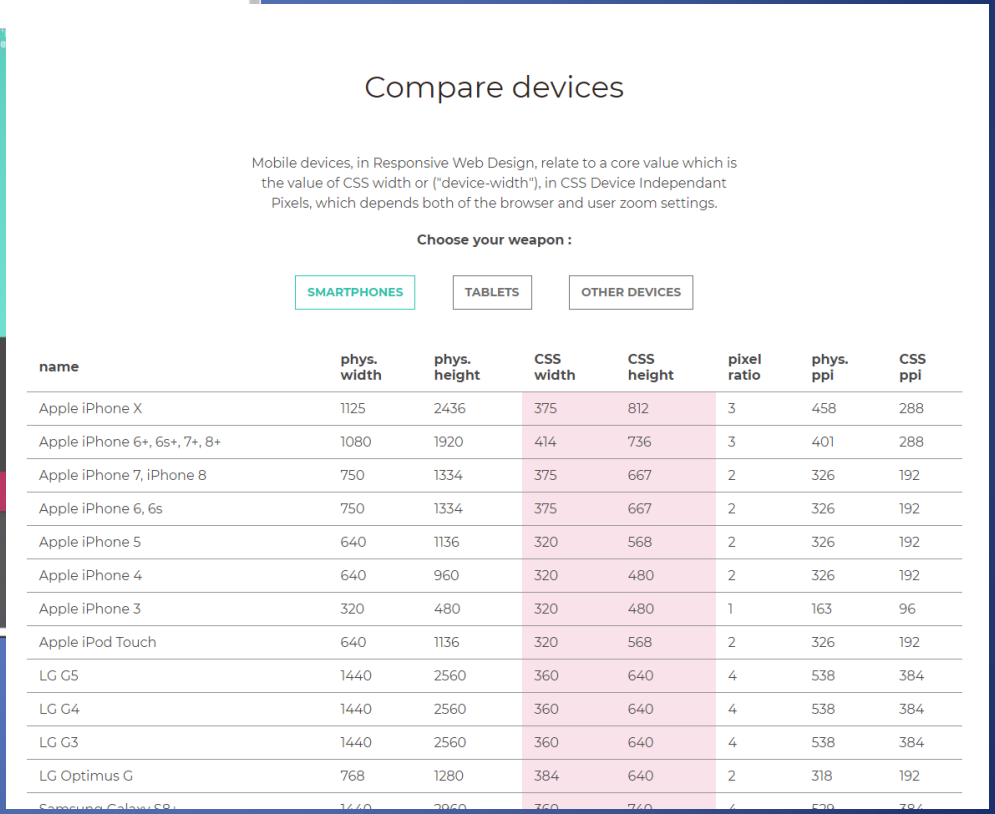
- 장점 : URL 입력만으로 다양한 디바이스의 적용 여부를 한 번에 확인 가능
- 단점 : 카카오의 인수로 인한 각 디바이스의 업데이트가 제 때 이뤄지지 않았다는 단점이 있음. 국산 디바이스에 대한 대응은 아직까지 유효한편임

<Product by 안재욱>

Responsive Web 해상도 확인

해상도 확인 싸이트 (https://www.mydevice.io **)**





Compare devices

Mobile devices, in Responsive Web Design, relate to a core value which is the value of CSS width or ("device-width"), in CSS Device Independant Pixels, which depends both of the browser and user zoom settings.

Choose your weapon :

SMARTPHONES | TABLETS | OTHER DEVICES

name	phys. width	phys. height	CSS width	CSS height	pixel ratio	phys. ppi	CSS ppi
Apple iPhone X	1125	2436	375	812	3	458	288
Apple iPhone 6+, 6s+, 7+, 8+	1080	1920	414	736	3	401	288
Apple iPhone 7, iPhone 8	750	1334	375	667	2	326	192
Apple iPhone 6, 6s	750	1334	375	667	2	326	192
Apple iPhone 5	640	1136	320	568	2	326	192
Apple iPhone 4	640	960	320	480	2	326	192
Apple iPhone 3	320	480	320	480	1	163	96
Apple iPod Touch	640	1136	320	568	2	326	192
LG G5	1440	2560	360	640	4	538	384
LG G4	1440	2560	360	640	4	538	384
LG G3	1440	2560	360	640	4	538	384
LG Optimus G	768	1280	384	640	2	318	192

- 각 디바이스 별 **CSS width** 또는 CSS height 를 확인 가능

Responsive Web 설정

디바이스 사이즈 설정하기(viewport 기준)

- viewport의 기본 형식

```
<meta name="viewport" content="<속성1=값>, <속성2=값>", ...>
```

- viewport의 속성

속성	설명	사용 가능한 값	기본값
width	뷰포트 너비	device-width 또는 크기(양수)	브라우저 기본값
height	뷰포트 높이	device-height 또는 크기(양수)	브라우저 기본값
user-scalable	확대/축소 가능여부	yes 또는 no	yes
initial-scale	초기 확대/축소 값	1~10	1
minimun-scale	최소 확대/축소 값	0~10	0.25
maximun-scale	최대 확대/축소 값	0~10	1.6

Responsive Web 설정

디바이스 사이즈 설정하기(viewport 기준)

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-scalable=no">
```

- viewport 는 각 화면의 실제 크기에 맞춰서 영역을 설정하는 것에 대한 정의 구문

- width=device-width 는 각각의 디바이스의 가로 길이에 맞춰 전체 가로 길이를 설정
- initial-scale=1.0 은 최초의 화면에서 더 이상 줌(zoom) 기능이 일어나지 않도록 설정
- user-scalable=no 는 사용자가 확대(화면에 더블 클릭 또는 터치 기반 확대)할 수 없게 통제

Responsive Web 설정

- **뷰포트 내용을 맞출 때 유의사항**

1. 가로 스크롤이 나오지 않도록 유의한다.

- 모바일 환경에서 가로 스크롤을 통한 이동에 익숙하지 않음.

2. 큰 사이즈의 고정 가로 폭(width)에서 px 단위 사용을 非(비) 추천

- 가로 폭을 %가 아닌 px로 고정시킬 경우 가장 작은 모바일 단위에서 css를 재조정 하거나 가로 스크롤 바가 생성됨.

- 단, 테이블의 경우 필요 요소가 디바이스의 가로 폭(width)보다 클 경우 테이블 내부 가로 스크롤을 생성해서 사용할 수 있음.

3. 특정 뷰포트를 기준으로 이미지를 렌더링 금지

- 각각의 디바이스 가로가 다르기 때문에 특정한 기기를 대상으로 웹사이트의 뷰포트를 설정하는 것은 바람직하지 않음. (다변화하는 해상도에 대응하고자 함)

Responsive Web 제작

- **Media Query**

  -Media Query는 CSS3에서 도입된 CSS의 기능

  CSS 코드 내부에서 분기하는 방법

```
@media only all and (조건문) {실행문}
```

Responsive Web 제작

- **Media Query**

@media only all and (조건문) {실행문}

- **@media**: 미디어 쿼리가 시작됨을 선언
- **only**: only 키워드는 미디어 쿼리를 지원하는 사용자 에이전트만 미디어 쿼리 구문을 해석하라는 명령. 생략 가능.
- **all**: all 키워드는 미디어 쿼리를 해석해야 할 대상 미디어를 선언한 것. 생략 가능. (all 키워드 대신 screen 또는 print와 같은 특정 미디어를 구체적으로 언급 가능)
- **and**: and 키워드는 논리적으로 'AND' 연산을 수행하여 앞과 뒤의 조건을 모두 만족해야 한다는 것을 의미.
- **(조건문)**: 브라우저는 조건문이 참일 때{실행문}을 처리하고 거짓일 때 무시.
- **{실행문}**: 일반적인 CSS 코드를 이 괄호 안에 작성. (조건문)이 참일 경우에만 실행.

Responsive Web 제작

- **미디어 쿼리 유형**

기기명	설명
all	모든 장치 (특정한 유형이 없이 모든 형태에서 적용, 생략 가능, 기본값)
print	인쇄 장치
screen	컴퓨터 화면 또는 스마트 기기 화면
tv	영상과 음성이 동시에 출력되는 장치
projection	프로젝터 장치
handheld	손에 들고 다니는 소형 장치
speech	음성 출력 장치
aural	음성 합성 장치(화면을 읽어 소리로 출력해 주는 장치)
embossed	점자 인쇄 장치(화면을 읽어 종이에 점자로 찍어내는 장치)
tty	디스플레이 기능이 제한된 장치
braille	점자 표시 징ㅊ;

Responsive Web 제작

- Media Query

```
@media only all and (조건문) {실행문}
```

※ 미디어 쿼리 및 실행문 작성간 주의 사항

① 상단에서 작성한 선언에 대해서 변경 사항에 대해서만 작성

- 상단과 일치하는데, 중복 사용할 경우 로딩 속도에 영향을 줌

② 반드시 중괄호 { } 내부에 CSS 작성할 것

- 중괄호 미디어 쿼리 문을 적용하겠다는 의미(중괄호가 이중으로 들어가기 때문에 반드시 닫힘 중괄호 부분을 확인할 것)

③ 반드시 and 와 (조건문)은 사이를 띄울 것

- 미디어 쿼리 사용시 문법적으로 오류 발생 방지

④ 반드시 (조건문) 내부에는 이상(min-width) 또는 이하(max-width)의 규칙을 순서대로 나열하여 적용할 것 – css가 위에서부터 로딩 되는 것을 고려하여 적용

- min-width 는 작은 사이즈 순으로 작성할 것
- max-width 는 큰 사이즈 순으로 작성할 것

Responsive Web 제작

- **Media Query**

CSS 코드 외부에서 분기하는 방법

```
<link rel="stylesheet" type="text/css" media="all and (조건A)" href="A.css">
<link rel="stylesheet" type="text/css" media="all and (조건B)" href="B.css">
```

✓ 외부에서 분기할 경우, 많은 CSS 파일을 로딩해야 하기 때문에 그만큼의 로딩 시간이 걸림. – **(비추천)**

Responsive Web 제작

- **Media Query Code 구성**

```
@charset "utf-8";
/* All Device */
모든 해상도를 위한 공통 코드 작성. 모든 해상도 코드실행.

/* Mobile Device */
768px 미만 해상도의 모바일 기기를 위한 코드.

/* Tablet & Desktop Device */
@media all and (min-width:768px) {
사용자 해상도가 768px 이상일 때 실행. 태블릿 및 데스크톱의 공통코드를 작성.
}

/* Tablet Device */
@media all and (min-width:768px) and (max-width:1024px) {
사용자 해상도가 768px 이상이고 1024px 이하일 때 실행. 아이패드 또는 비교적 작은 해상도의 Lap-Top(노트북)이나 데스크톱에 대응하는 코드를 작성.
}

/* Desktop Device */
@media all and (min-width:1025px) {
사용자 해상도가 1025px 이상일 때 실행. 1025px 이상의 랩탑 또는 데스크톱에 대응하는 코드를 작성
}
```

Responsive Web 제작

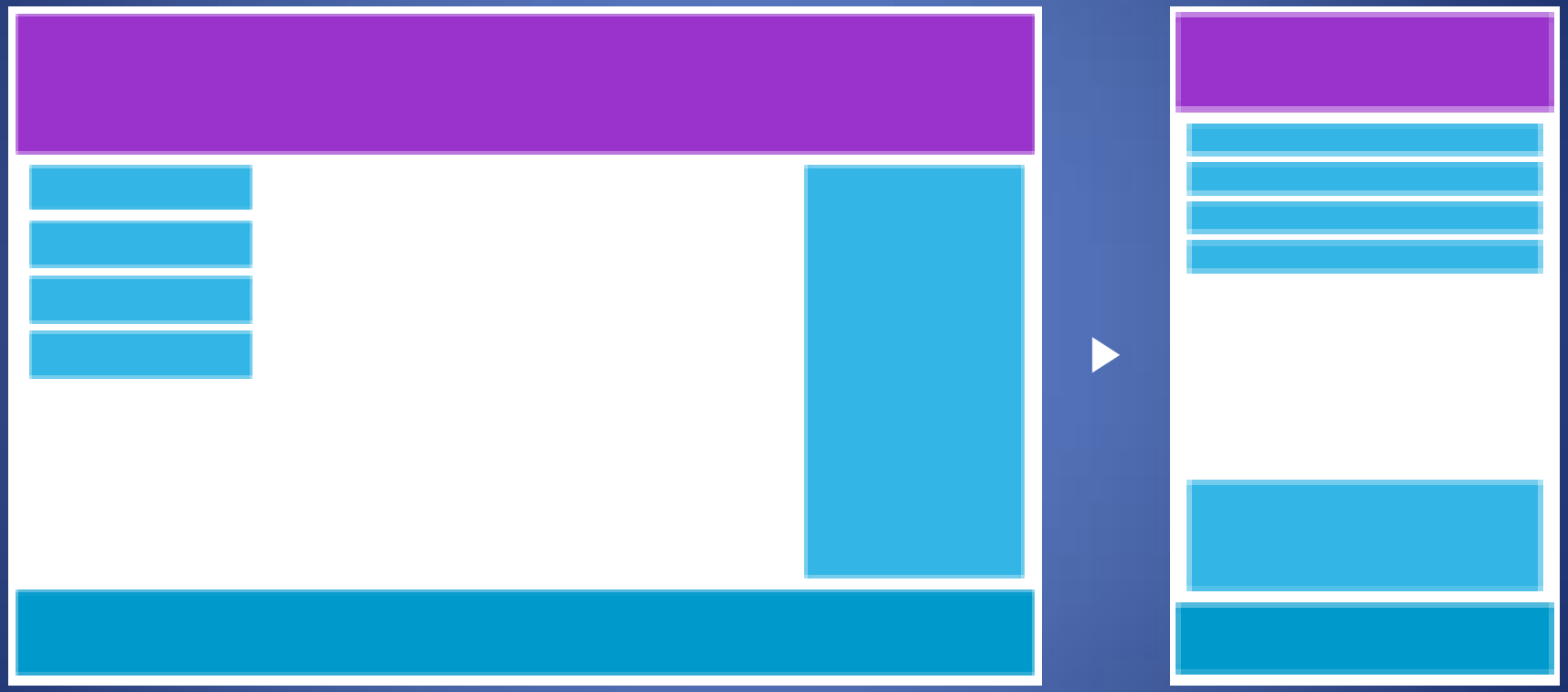
- **Media Query Code 적용 예시**

```
@media only screen and (max-width: 500px) {
    body {
        background-color: blue;
    }
}
```

✓ '가로 500px 이하의 디바이스에서 배경색을 변경한다.'는 의미

Responsive Web 제작

- 미디어 쿼리 코드 실행 중단 조건 추가 (PC → Mobile)



<PC>

<Mobile>

Responsive Web 제작

- 미디어 쿼리 코드 실행 중단 조건 추가 (PC → Mobile)

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

/* For mobile phones: */
@media only screen and (max-width: 768px) {
    [class*="col-"] {
        width: 100%;
    }
}
```

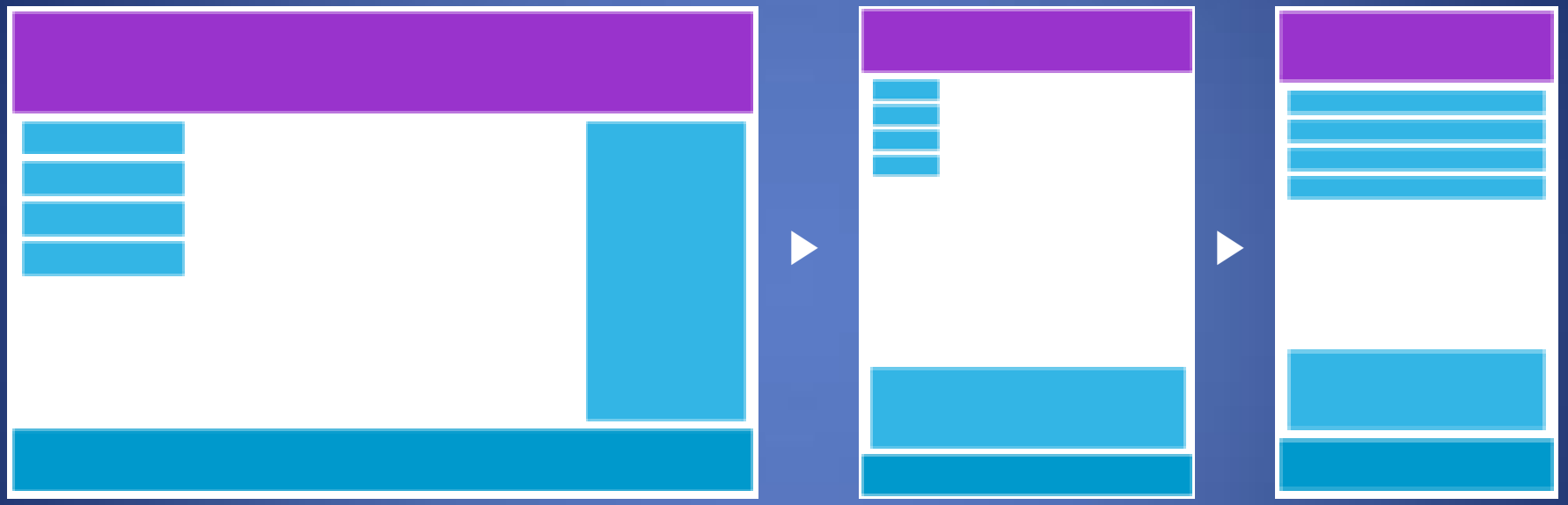
-반응형 웹 사이트 구축을 위한 디자인 순서 :

모바일→태블릿→ PC 순으로 진행하는 것 추천

※ 데이터 통신의 경우, 모바일이 유선 네트워크 환경보다 먼저 로딩이 되도록 구성

Responsive Web 제작

- 미디어 쿼리 코드 실행 중단 조건 추가 (PC → Tablet → Mobile)



<PC> <Tablet> <Mobile>

Responsive Web 제작

- 미디어 쿼리 코드 실행 중단 조건 추가 (PC → Tablet → Mobile)

```
/* For mobile phones: */
[class*="col-"] {
    width: 100%;
}
@media only screen and (min-width: 768px) {
    /* For tablets: */
    .col-m-1 {width: 8.33%;}
    .col-m-2 {width: 16.66%;}
    .col-m-3 {width: 25%;}
    .col-m-4 {width: 33.33%;}
    .col-m-5 {width: 41.66%;}
    .col-m-6 {width: 50%;}
    .col-m-7 {width: 58.33%;}
    .col-m-8 {width: 66.66%;}
    .col-m-9 {width: 75%;}
    .col-m-10 {width: 83.33%;}
    .col-m-11 {width: 91.66%;}
    .col-m-12 {width: 100%;}
}
```

```
@media only screen and (min-width: 1024px) {
    /* For desktop: */
    .col-1 {width: 8.33%;}
    .col-2 {width: 16.66%;}
    .col-3 {width: 25%;}
    .col-4 {width: 33.33%;}
    .col-5 {width: 41.66%;}
    .col-6 {width: 50%;}
    .col-7 {width: 58.33%;}
    .col-8 {width: 66.66%;}
    .col-9 {width: 75%;}
    .col-10 {width: 83.33%;}
    .col-11 {width: 91.66%;}
    .col-12 {width: 100%;}
}
```

Responsive Web 제작

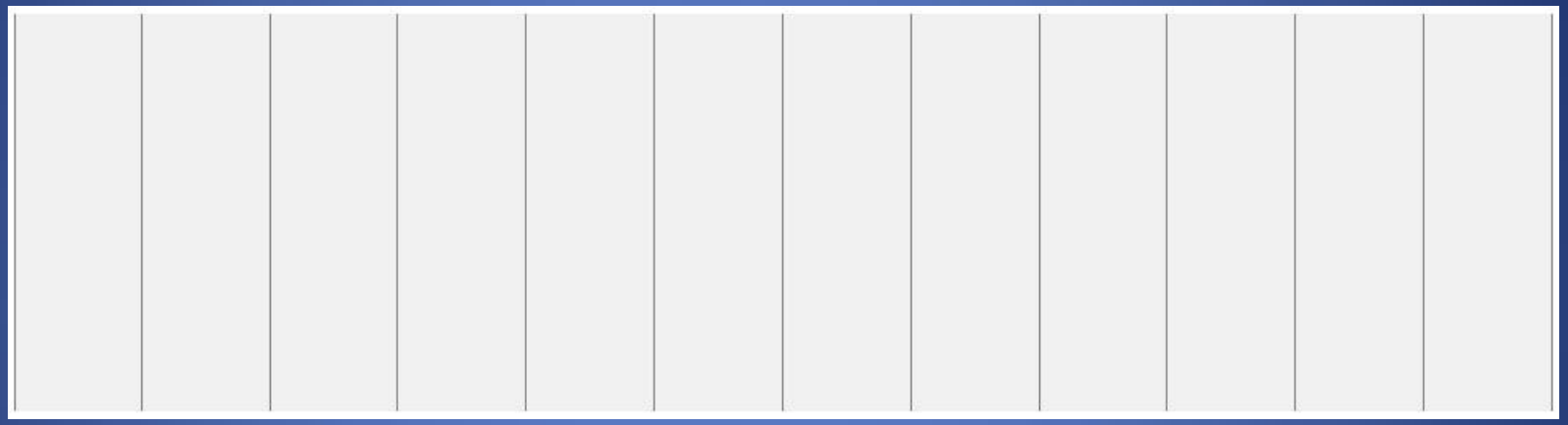
- 미디어 쿼리 중복 적용

```
<div class="row">
  <div class="col-3 col-m-3">...</div>
  <div class="col-6 col-m-9">...</div>
  <div class="col-3 col-m-12">...</div>
</div>
```

Responsive Web 설정

- **반응형 웹 가변 Grid 설정**

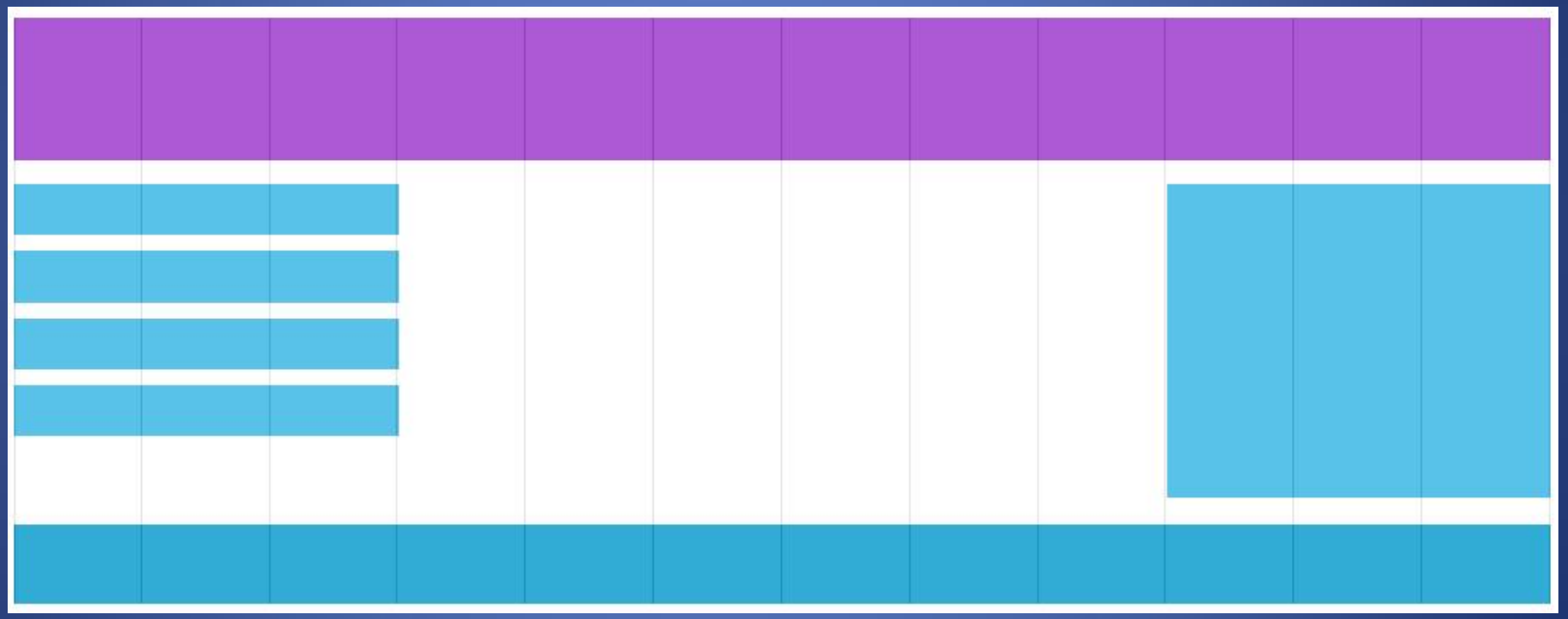
1. 웹 페이지를 grid를 통해 열로 나눔



Responsive Web 설정

- **반응형 웹 Grid 설정**

2. 반응형에서는 총 12개의 열로 나누게 되며, 일정 이상을 축소했을 경우, width 값을 100%로 설정



Responsive Web 설정

- **반응형 웹 가변 Grid View 구성**

CSS 파일에서 *{box-sizing: border-box;}으로 속성값 추가 혹은 확인

※ 크로스 브라우징을 위한 선언

```
*{-webkit-box-sizing: border-box; -moz-box-sizing: border-box; box-sizing: border-box;}
```

위에서 "box-sizing: border-box;" 의 의미

width + padding + border = 요소

높이의 실제 폭 + 채우기 + 테두리 = 요소의 실제 높이

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성**

글자의 사이즈를 px단위로 조정을 하게 되면 고정값을 갖기 때문에 사이즈별로 개별 조정의 불편함이 있음.
가변 레이아웃을 잡을 때, 폰트사이즈가 함께 유동적으로 변경하려면 em, rem 단위를 사용해야 함

글자크기(em) = 글자크기(px) / 16px

기본적으로 1em은 16px을 지정하게 되는데, 이는 부모의 요소에서 폰트사이즈를 지정하면 이에 맞춰서 변경

만약, font-size:1.5em의 css를 적용한 <p> 태그의 부모 요소( )가 font-size를 24px로 css를 설정했다면, <p> 태그에 적용되는 font-size는 36px로 적용됨

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성**

<문제> "가변폰트"의 실제 폰트 사이즈는 얼마인가?

```
<style>
        #wrap{font-size:12px;}
        .headet-text{font-size:2em;}
        .fluid-text{font-size:1.5em;}
</style>
<div id="wrap">
        <header class="headet-text">
                <p>가변 그리드 레이아웃</p>
                <p class="fluid-text">가변폰트</p>
        </header>
</div>
```

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성**

<문제> "가변폰트"의 실제 폰트 사이즈는 얼마인가?

```
<style>
        #wrap{font-size:12px;}
        .headet-text{font-size:2em;}
        .fluid-text{font-size:1.5em;}
</style>
<div id="wrap">
        <header class="headet-text">
                <p>가변 그리드 레이아웃</p>
                <p class="fluid-text">가변폰트</p>
        </header>
</div>
```

#wrap의 폰트 사이즈 12px
.header-text의 폰트 사이즈 12px × 2(em) = 24px
.fluid-text의 폰트 사이즈 24px × 1.5(em) = 36px

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성**

앞의 em 단위는 부모의 영향을 받기 때문에 폰트 사이즈가 계속 달라진다는 단점이 있음. 이런 단점을 없애기 위해 만든 단위가 rem. (직속 부모의 상속 문제를 해결할 수 있음)

글자크기(px) = 최상 단의 글자크기(px) × 폰트 사이즈(rem)

유동 변화를 최상 단의 폰트 사이즈로부터 가져올 수 있는 단위인 **rem(Root em)**.

최상 단의 지정된 폰트사이즈로부터 영향을 받기 때문에 부모의 폰트 사이즈와는 무관하게 적용 가능

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성**

<문제> "가변폰트"의 실제 폰트 사이즈는 얼마인가?

```
<style>
        body{font-size:16px;}
        .headet-text{font-size:2rem;}
        .fluid-text{font-size:1.5rem;}
</style>
<body>
        <div id="wrap">
                <header class="headet-text">
                        <p>가변 그리드 레이아웃</p>
                        <p class="fluid-text">가변폰트</p>
                </header>
        </div>
</body>
```

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성**

<문제> "가변폰트"의 실제 폰트 사이즈는 얼마인가?

```
<style>
        html{font-size:24px;}
        .headet-text{font-size:2rem;}
        .fluid-text{font-size:1.5rem;}
</style>
<body>
        <div id="wrap">
                <header class="headet-text">
                        <p>가변 그리드 레이아웃</p>
                        <p class="fluid-text">가변폰트</p>
                </header>
        </div>
</body>
```

최상단인 <html>의 폰트 사이즈 24px
.header-text의 폰트 사이즈 24px × 2(rem) = 48px
.fluid-text의 폰트 사이즈 24px × 1.5(em) = 36px

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성 – view의 크기 기준**

vw(Viewport Width) 단위는 웹 브라우저의 너비를 100을 기준으로 하여 크기를 결정하는 단위

(vw 단위를 적용할 글자 크기값 × 브라우저 너비값) ÷ 100 = 크기값

만약 글자의 크기를 10vw로 설정하면 웹 브라우저의 너비는 100을 기준으로 하기 때문에 10%가 적용

정확한 값을 얻기 위해서 화면의 크기가 1280px이라면 **(10 × 1280) ÷ 100** 공식을 이용해서 얻은 값이 128px로 표현됨

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성 – view의 크기 기준**

vh(Viewport Height) 단위는 웹 브라우저의 높이를 100을 기준으로 하여 크기를 결정하는 단위

(vh 단위를 적용할 글자 크기값 × 브라우저 높이값) ÷ 100 = 크기값

만약 글자의 크기를 10vh로 설정하면 웹 브라우저의 높이는 100을 기준으로 하기 때문에 10%가 적용

정확한 값을 얻기 위해서 화면의 높이가 1024px이라면 **(10 × 1024) ÷ 100** 공식을 이용해서 얻은 값이 102.4px로 표현됨(px 단위로는 표현이 되지 않는 소수점 영역도 글자에 포함이 됨)

Responsive Web 설정

- **반응형 웹 가변 글꼴(font) 구성 – view의 크기 기준**

vmin(Viewport Minimum) 단위는 웹 브라우저의 가로 또는 세로 중 짧은 쪽을 기준으로 100으로 적용하는 단위

만약 글자의 크기를 10vmin로 설정하고 가로가 1280(가로) × 1024(세로)면 작은 사이즈를 갖고 있는 세로에 대한 부분이 적용, 결과값은 **102.4px로 출력**.

vmax(Viewport Maximum) 단위는 웹 브라우저의 가로 또는 세로 중 넓은 쪽을 기준으로 100으로 적용하는 단위

만약 글자의 크기를 10vmax로 설정하고 가로가 1280(가로) × 1024(세로)면 큰 사이즈를 갖고 있는 가로에 대한 부분이 적용, 결과값은 **128px로 출력**.

Responsive Web 설정

- **반응형 웹 가변 이미지 구성**

가변 이미지란?
화면 너비에 따라 이미지의 너비와 높이가 바뀌는 이미지

<img> 태그의 경우, 이미지를 넣더라도 크기 미지정시, 원본 수치로 적용
웹사이트에서 수치개념을 적용 → 각 디바이스별로 적용하는데 한계 有
반응형 웹에서 가변 이미지로 구성을 하는 것이 **max-width**

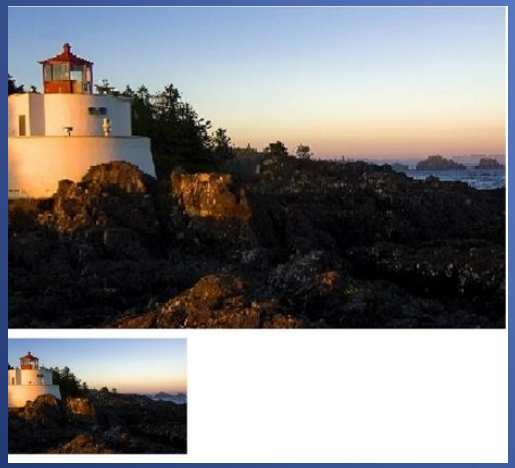
```
img {
  max-width: 100%;
}
```

Responsive Web 설정

- **반응형 웹 가변 이미지 구성**

가변 이미지 실습
아래와 같이 작성하고 가로가 400px인 이미지를 넣어서 기존 pc와 반응형 웹에서 확인하세요.

```
<style>
  .pic1 { width:100%; }
  .pic2 { max-width: 300px; }
</style>
<p><img src="images/photo1.jpg" class="pic1"></p>
<p><img src="images/photo1.jpg" class="pic2"></p>
```



Responsive Web 설정

- **반응형 웹 가변 이미지 구성**

<picture>태그와 <source>태그 – 상황별 다른 이미지로 표기하기

HTML5.1에는 <picture> 태그가 표준으로 되면서 <source> 태그와 함께 사용하면 화면의 해상도 뿐만 아니라 화면 너비에 따른 다른 이미지 파일을 포함할 수 있음.

조건을 지정하는 방법은 다음과 같음.

속성	설명
srcset	이미지 파일 경로
media	srcset에 지정한 이미지를 표시하기 위한 조건(max-width 또는 min-width)
type	파일유형
sizes	파일의 크기

Responsive Web 설정

- **반응형 웹 가변 이미지 구성**

<picture>태그와 <source>태그 – 실습

<picture> 태그는 IE에서는 지원하지 않기 때문에 대체 이미지가 필요(<img>태그 넣어주어야 함)

```
<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Fluid Image</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
</head>
<body>
        <picture>
                <source srcset="img/shop-large.jpg" media="(min-width:1024px)">
                <source srcset="img/shop-medium.jpg" media="(min-width:768px)">
                <source srcset="img/shop-small.jpg" media="(min-width:320px)">

                <img src="img/shop.jpg" alt="fill with coffee" style="width:100%;">
        </picture>
</body>
</html>
```

display : flex

<Product by 안재욱>

Flex Box Layout

- **플렉서블 박스 레이아웃**

그리드 레이아웃을 기본으로 플렉스 박스를 원하는 위치에 배치하는 것
플렉스 박스를 이용하면 여유 공간에 따라 너비나 높이를 자유롭게 변형이 가능

기본형식

```
display: flex | inline-flex
```

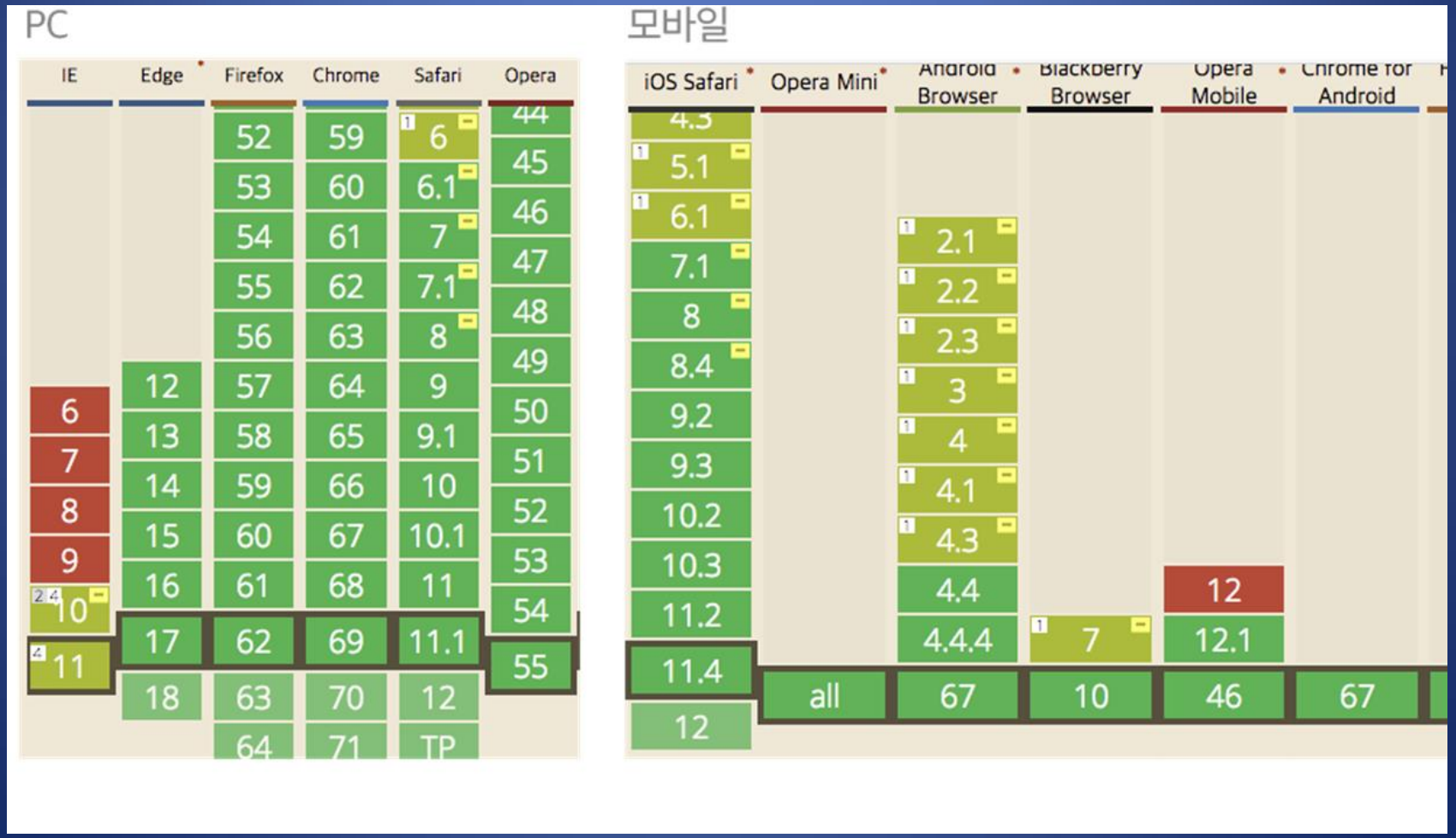
속성	설명
flex	플렉스 박스를 박스 레벨 요소로 정의
inline-flex	플렉스 박스를 인라인 레벨 요소로 정의

※ float 은 "띄우다"라는 의미로 배치와는 무관하지만 결과적으로 방향성을 가질 수 있기 때문에 종종 사용하였으나, 실질적인 공간 내부에서 배치와는 관계 없음. 반응형 웹에서는 내부 공간에서 실질적인 배치를 위한 조정이 가능하도록 구성하는 역할로 도입(CSS3부터 적용)

Flex Box Layout

- **플렉서블 박스 적용 브라우저**

flexbox는 Internet Explorer 10 이상에서 지원
Internet Explorer 10과 Internet Explorer 11에서 버그 존재



Responsive Web 제작

- **플렉서블 박스 레이아웃**

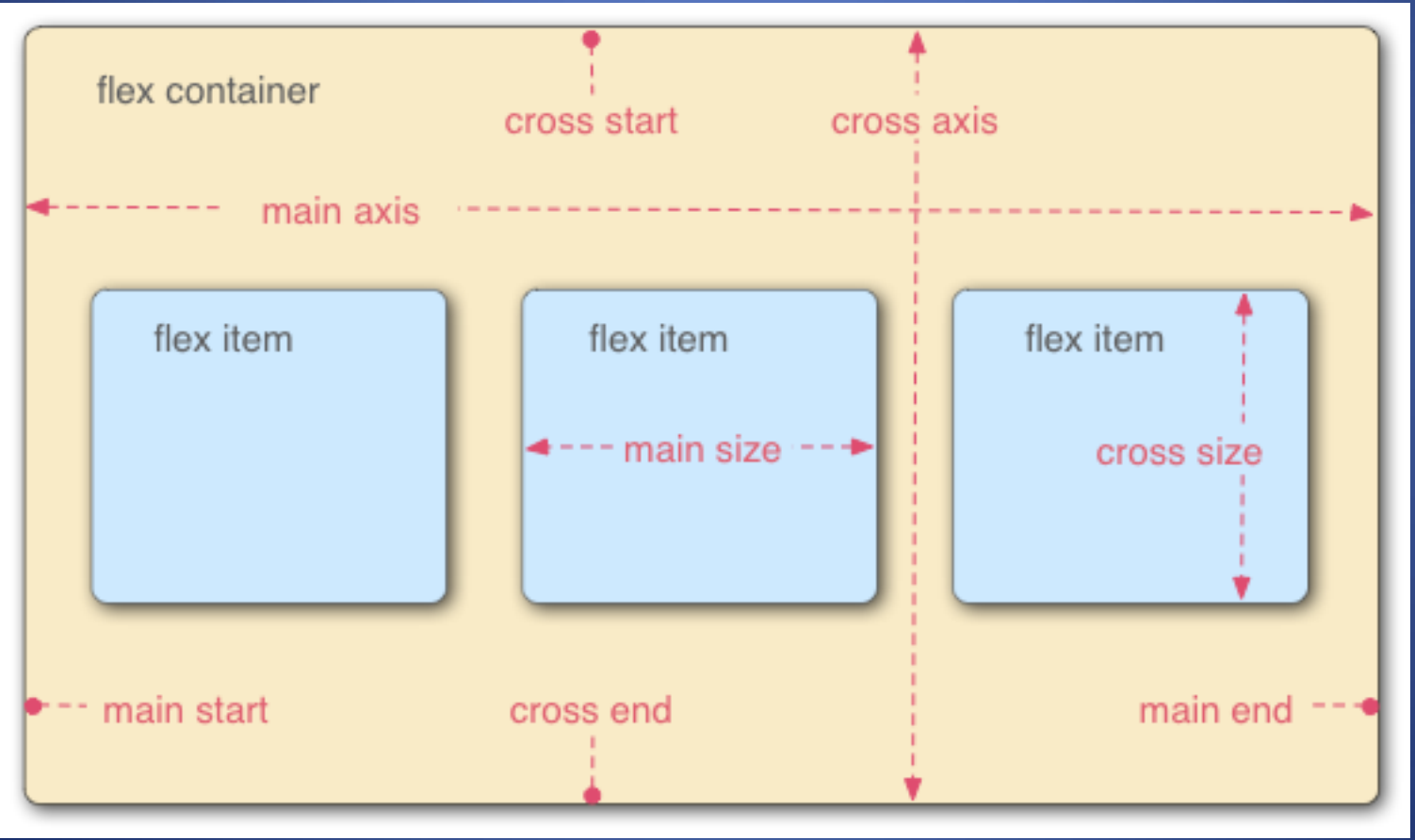
크로스 브라우저를 위한 접두사 적용 방법
– 아직 W3C에서 확정되지 않았기에 접두어를 붙여서 접근할 필요가 있음

```
display: -webkit-box;  /*iOS 6이하, 사파리 3.1*/
display: -moz-box;  /*파이어폭스 19 이하*/
display: -ms-flexbox;  /*IE 10*/
display: -webkit-flex;  /*웹킷 구형 버전*/
display: flex;  /*표준 스펙*/
```

Flex Box Layout

- **flexbox 두 가지 축 - 수평방향(Main Axis), 수직방향(Cross Axis)**

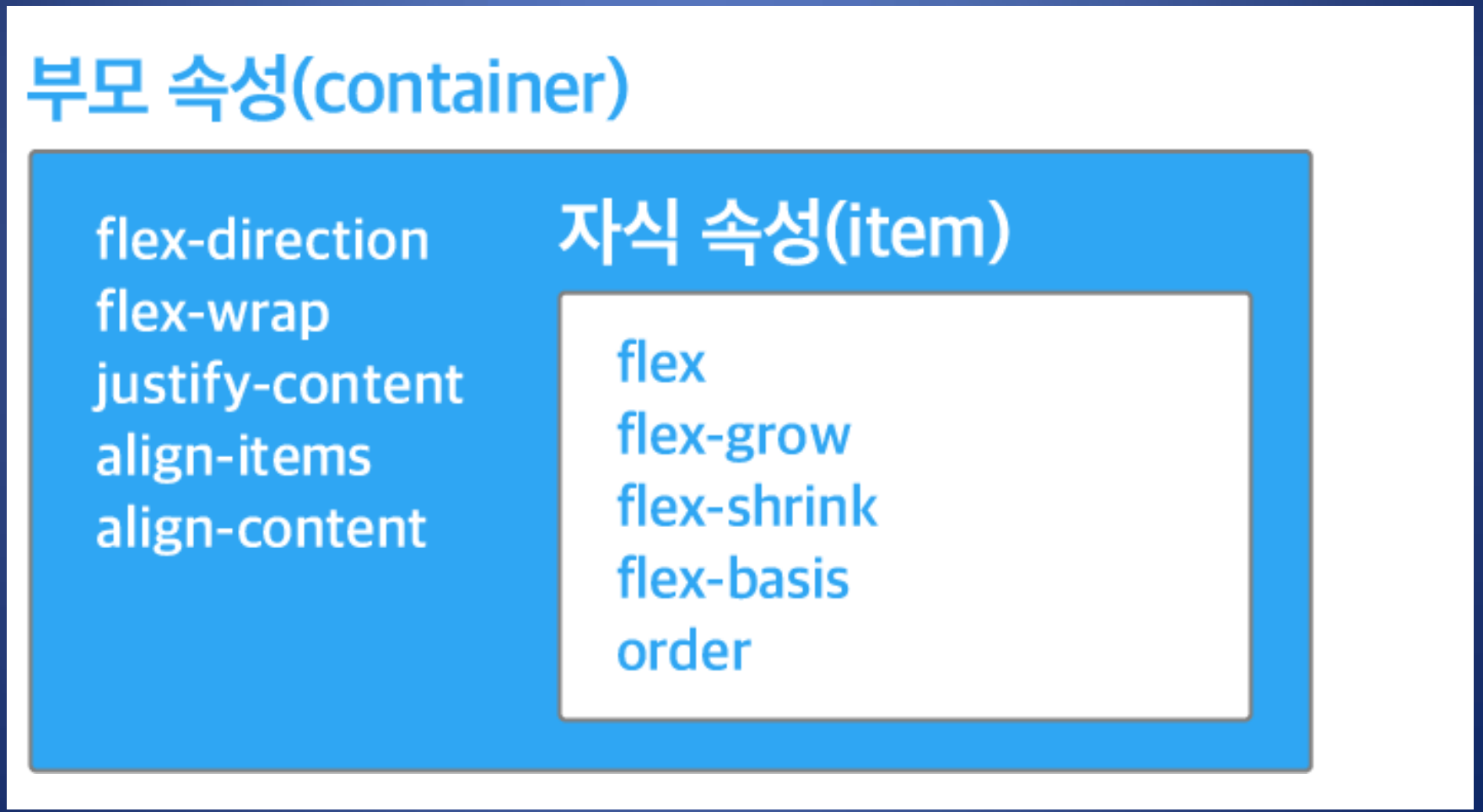
수평방향으로 결정하는 선언과 수직방향으로 결정하는 선언인 두 가지 축이 있음



Flex Box Layout

- **flexbox 영역별 구조 – 부모 속성(Container), 자식 속성(item)**

부모 속성에서 자식의 방향 및 분배를 작성 / 자식 속성에서 각 요소의 배치 순서 및 영역의 너비값을 지정



Flex Box Layout

- **flexbox 두 가지 축 - 방향 결정 선언** (flex-direction)

주축은 flex-direction의 속성을 사용
주로 내부 자식 요소의 **방향을 결정하는 선언**으로 작성

flex-direction : row | row-reverse | column | column-reverse

속성값	설명
row	주축을 가로로 교차축을 세로로 지정. 플렉스 항목은 주축 시작점에서 끝점으로(왼쪽부터 오른쪽 방향) 배치
row-reverse	주축을 가로로 교차축을 세로로 지정. 플렉스 항목은 주축 끝점에서 시작점으로(오른쪽부터 왼쪽 방향) 배치
column	주축을 세로로 교차축을 가로로 지정. 플렉스 항목은 주축 시작점에서 끝점으로(위쪽부터 아래쪽 방향) 배치
column-reverse	주축을 세로로 교차축을 가로로 지정. 플렉스 항목은 주축 끝점에서 시작점으로(아래쪽부터 위쪽 방향) 배치

Flex Box Layout

- **flexbox 두 가지 축 - 방향 결정 선언** (flex-direction)

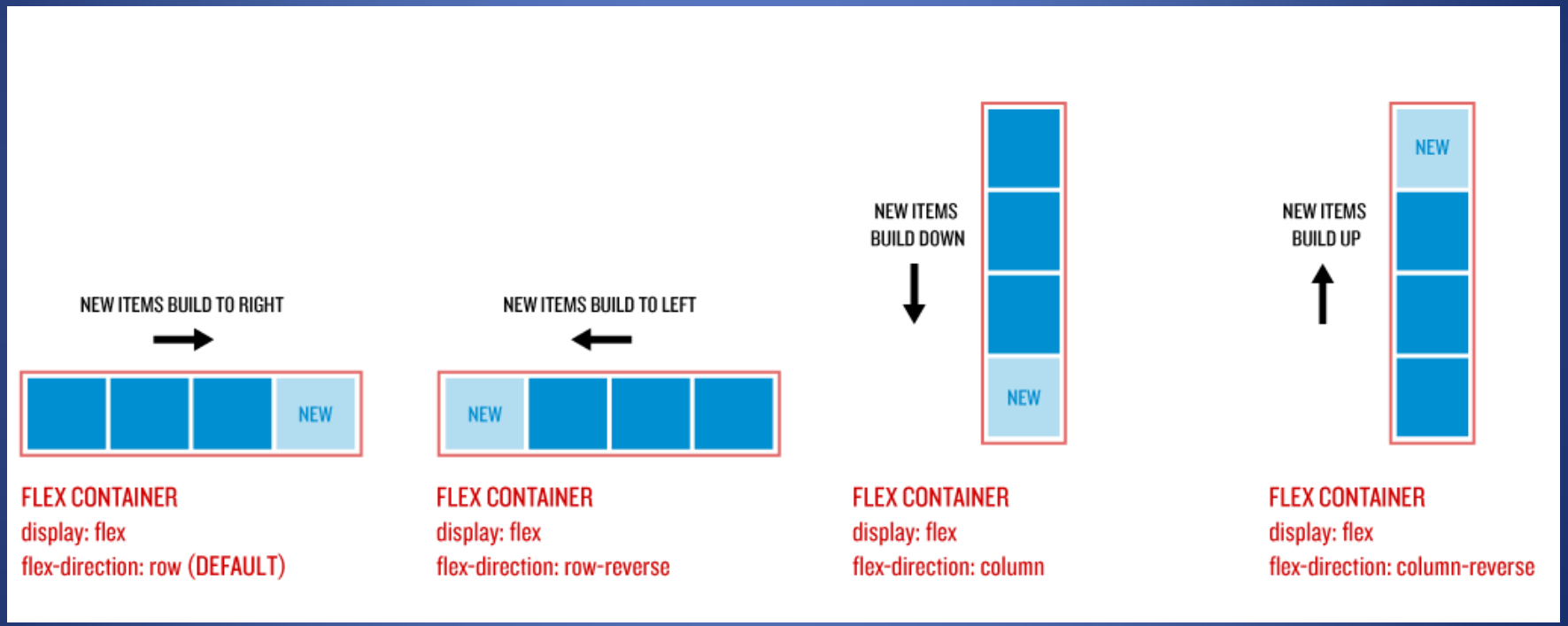
주축은 flex-direction의 속성을 사용
주로 내부 자식 요소의 **방향을 결정하는 선언**으로 작성

flex-direction:row; ▶ 가로 방향으로 배치한다.(기본값)
flex-direction:column; ▶ 세로방향으로 배치한다.
flex-direction:row-reverse; ▶ 가로 방향 역순으로 배치한다. (가로 공간에 여유가 있을 경우 오른쪽으로 붙는다.)
flex-direction:column-reverse; ▶ 세로 방향 역순으로 배치한다. (세로 공간에 여유가 있을 경우 하단으로 붙는다.)

Flex Box Layout

- **flexbox 두 가지 축 - 방향 결정 선언** (flex-direction)

주축은 flex-direction의 속성을 사용
주로 내부 자식 요소의 **방향을 결정하는 선언**으로 작성



Flex Box Layout

- **flexbox 두 가지 축 - 공간 정렬 선언** (flex-wrap)

주축 내부의 공간을 결정하는 flex-wrap의 속성을 사용
주로 내부 자식 요소의 **공간을 결정하는 선언**으로 작성

flex-wrap : nowrap | wrap | wrap-reverse |

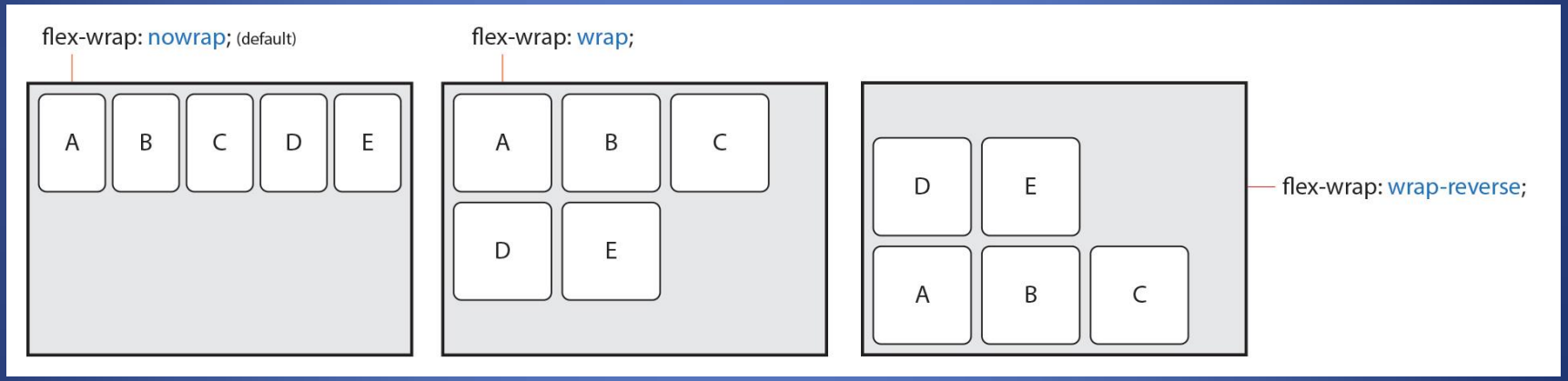
flex-wrap:nowrap; ▶ 가로폭이 줄어들더라도 공간 부족으로 떨어지는 것을 막을 수 있음. (공간 부족시 동일하게 줄어드는 경향이 있음)
flex-wrap:wrap; ▶ 가로폭이 줄어들면 공간 부족시 자동적으로 떨어짐. (전체 세로공간 대비하여 여유 공간을 균등하게 나누면서 떨어짐)
flex-wrap:wrap-reverse; ▶ 가로폭이 줄어들면 공간 부족시 역순으로 떨어짐

Flex Box Layout

- **flexbox 두 가지 축 - 공간 정렬 선언** (flex-wrap)

주축 내부의 공간을 결정하는 flex-wrap의 속성을 사용
주로 내부 자식 요소의 **공간을 결정하는 선언**으로 작성

flex-wrap : nowrap | wrap | wrap-reverse |



Flex Box Layout

- **flexbox 두 가지 축 - 수평 방향 정렬 선언** (justify-content)

주축 내부의 수평방향 정렬을 결정하는 justify-content의 속성을 사용
주로 내부 자식 요소의 **가로방향 정렬을 결정하는 선언**으로 작성

flex-wrap : flex-start | flex-end | center | space-between | space-around

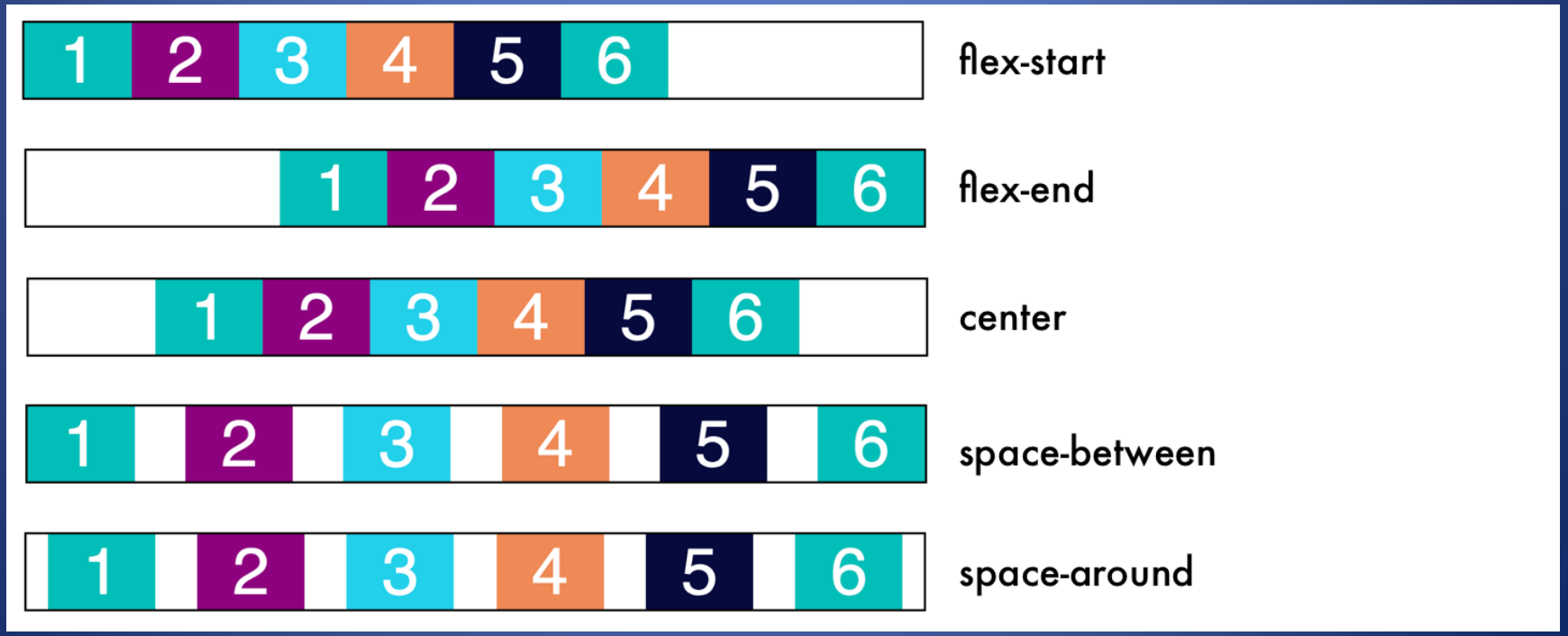
justify-content:flex-start; ▶ 내부 박스를 좌측으로 정렬 (기본값)
justify-content:flex-end; ▶ 내부 박스를 우측으로 정렬
justify-content:center; ▶ 내부 박스를 중앙으로 정렬
justify-content:space-between; ▶ 내부 박스를 모든 공간을 활용하여 펼침 정렬
justify-content:space-around; ▶ 각 내부 박스에 여유 공간을 내부 박스 기준으로 좌우측으로 나누어 정렬

Flex Box Layout

- **flexbox 두 가지 축 - 수평 방향 정렬 선언** (justify-content)

주축 내부의 수평방향 정렬을 결정하는 flex-wrap의 속성을 사용
주로 내부 자식 요소의 **가로방향 정렬을 결정하는 선언**으로 작성

justify-content : flex-start | flex-end | center | space-between | space-around



Flex Box Layout

- **flexbox 두 가지 축 - 수직 방향 정렬 선언** (align-items)

주축 내부의 수직 방향 정렬을 결정하는 align-items의 속성을 사용
주로 내부 자식 요소의 **세로방향 정렬을 결정하는 선언**으로 작성

align-items : stretch | flex-start | flex-end | center

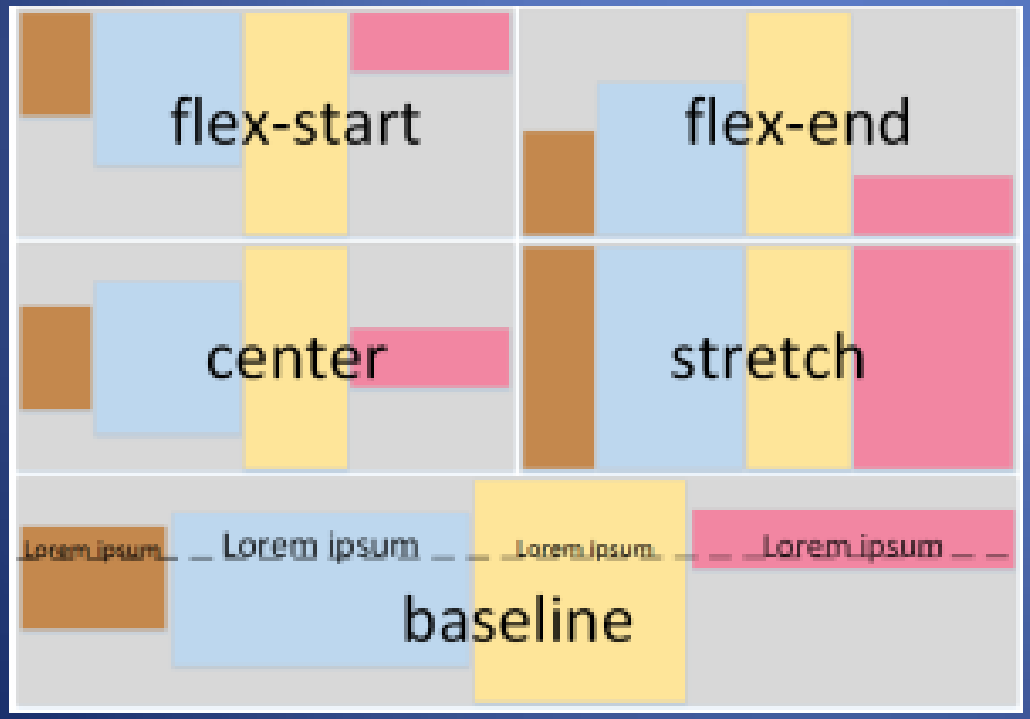
align-items:stretch; ▶ 외부 박스 세로 기준으로 내부 박스에 높이 값이 설정되어 있지 않을 경우 위아래의 공간을 모두 사용한다.(기본값)
align-items:flex-start; ▶ 외부 박스 세로 기준으로 내부 박스를 상단(top)으로 배치한다.
align-items:flex-end; ▶ 외부 박스 세로 기준으로 내부 박스를 하단(bottom)으로 배치한다.
align-items:center; ▶ 외부 박스 세로 기준으로 내부 박스를 상단(top)으로 배치한다.

Flex Box Layout

- **flexbox 두 가지 축 - 수직 방향 정렬 선언** (align-items)

주축 내부의 수직 방향 정렬을 결정하는 align-items의 속성을 사용
주로 내부 자식 요소의 **세로방향 정렬을 결정하는 선언**으로 작성

align-items : stretch | flex-start | flex-end | center | baseline



<Product by 안재욱>

Flex Box Layout

- **flexbox 두 가지 축 - 수직 방향 공간 분배 선언** (align-content)

주축 내부의 수직 방향 공간 분배를 결정하는 align-content 의 속성을 사용
주로 내부 자식 요소의 **세로방향 공간 분배를 결정하는 선언**으로 작성

align-content : flex-start | flex-end | center | space-between | space-around | stretch

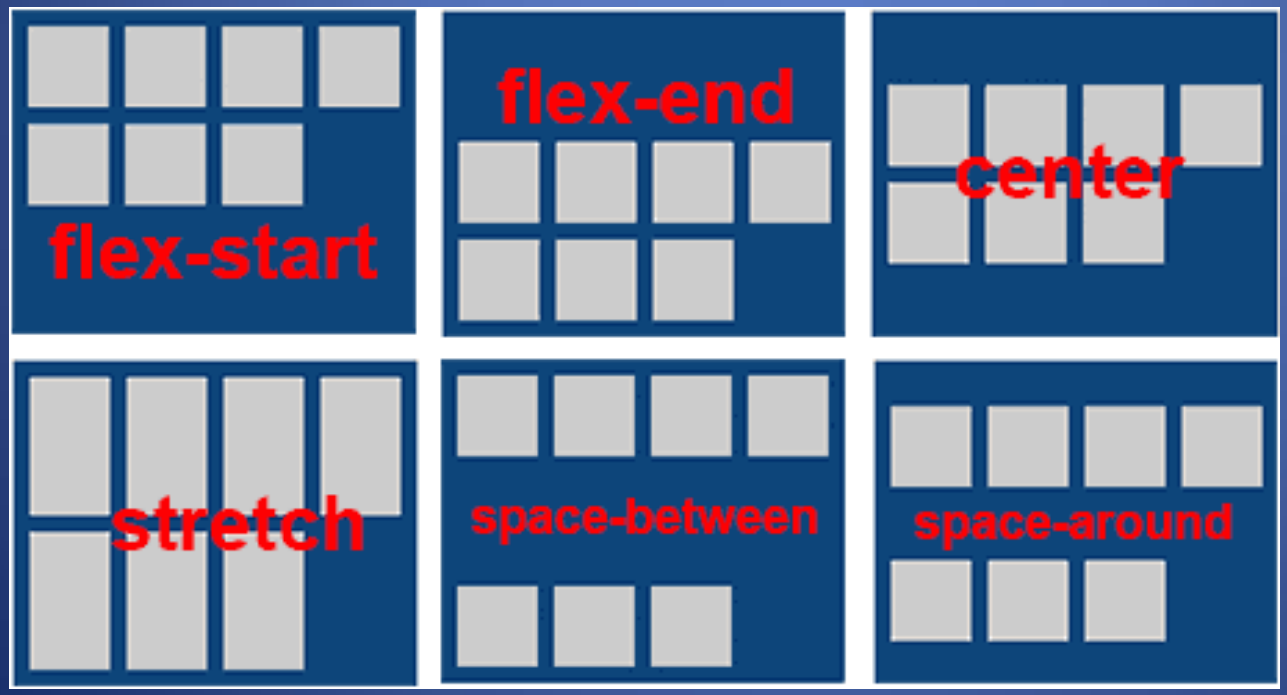
align-content: flex-start; ▶ 외부 박스 세로 기준으로 내부들을 박스의 상단(top)에 배치
align-content: flex-end; ▶ 외부 박스 세로 기준으로 내부들을 박스의 하단(bottom)에 배치
align-content: center; ▶ 외부 박스 세로 기준으로 내부들을 박스의 중앙(center)에 배치
align-content: space-between; ▶ 외부 박스 세로 기준으로 내부들을 박스의 모든 공간을 활용하여 상하로 배치
align-content: space-around; ▶ 외부 박스 세로 기준으로 내부들을 박스의 모든 공간의 여백 공간을 분할하여 배치
align-content: stretch; ▶ 외부 박스 세로 기준으로 내부들을 박스의 모든 공간을 활용하여 펼침 배치

Flex Box Layout

- **flexbox 두 가지 축 - 수직 방향 공간 분배 선언** (align-content)

주축 내부의 수직 방향 공간 분배를 결정하는 align-content 의 속성을 사용
주로 내부 자식 요소의 **세로방향 공간 분배를 결정하는 선언**으로 작성

align-content : flex-start | flex-end | center | space-between | space-around | stretch



Flex Box Layout

- **flex item – 수평 방향 공간 너비 선언** (flex-grow)
  - flex-grow 속성은 flex item의 **확장에 관련된 속성**
  - 0과 양의 정수를 속성값에 사용
  - 속성값이 0이면 flex container의 크기가 커져도 flex item의 크기가 커지지 않고 원래 크기로 유지
  - flex container의 크기가 커질 때 flex item의 크기도 커지게 하려면 1 이상의 값을 속성값으로 설정
  - 속성값이 1 이상이면 flex item의 원래 크기에 상관없이 flex container를 채우도록 flex item의 크기가 커짐

flex-grow : 양의 정수

Flex Box Layout

- **flex item – 수평 방향 공간 너비 선언** (flex-shrink)
  - flex-shrink 속성은 flex item의 **축소에 관련된 속성**
  - 0과 양의 정수를 속성값에 사용한다. 기본값은 1
  - 속성값이 0이면 flex container의 크기가 flex item의 크기보다 작아져도 flex item의 크기가 줄어들지 않고 원래 크기로 유지
  - 속성값이 1 이상이면 flex container의 크기가 flex item의 크기보다 작아질 때 flex item의 크기가 flex container의 크기에 맞추어 줄어듬

flex-shrink : 0 또는 양의 정수, 기본값 1

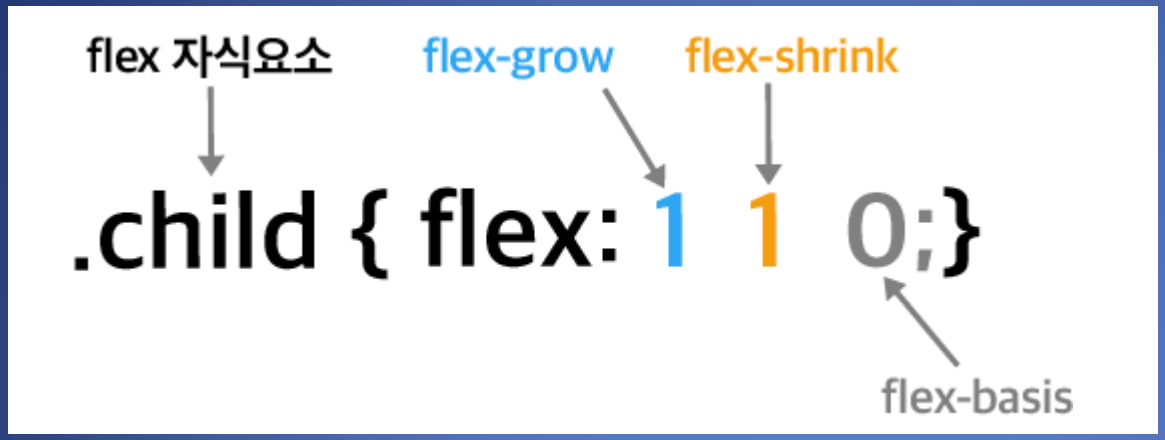
Flex Box Layout

- **flex item – 수평 방향 공간 너비 선언** (flex-basis)
  - flex-basis 속성은 flex item의 **기본 크기를 결정하는 속성**
  - 기본값은 auto
  - width 속성에서 사용하는 모든 단위(px, %, em, rem 등)를 속성값에 사용
  - flex-basis 속성의 값을 30px이나 30%와 같이 설정하면 flex item의 크기가 고정

flex-basis : 고정 단위(px, %, em, rem 등)

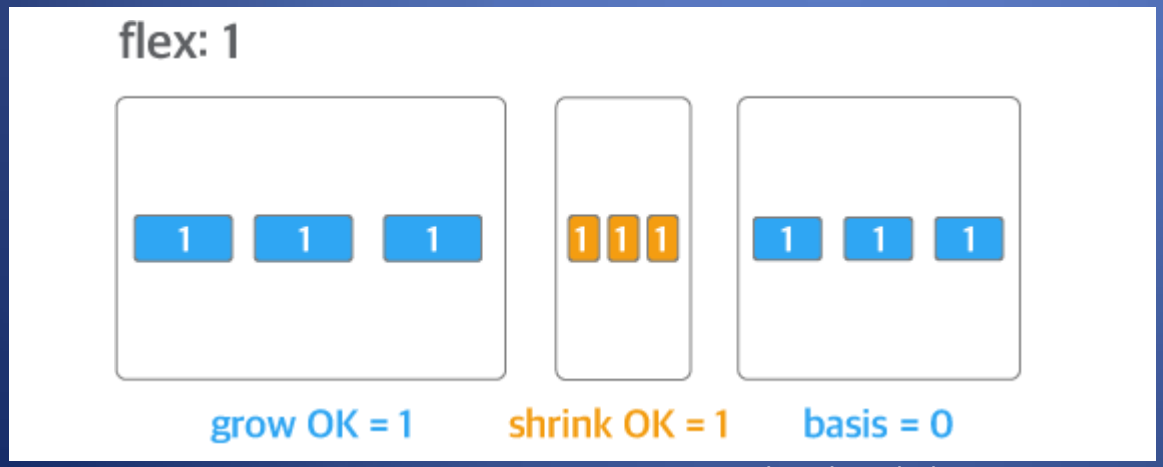
Flex Box Layout

- **flex item – 수평 방향 공간 너비 선언** (flex 통합 속성)




```
.child {
  flex-grow: 1;
  flex-shrink: 1;
  flex-basis: 0;
}
```

- flex: 1; 과도 동일한 개념으로 해석되는데 이에 대한 설명은 아래와 같음

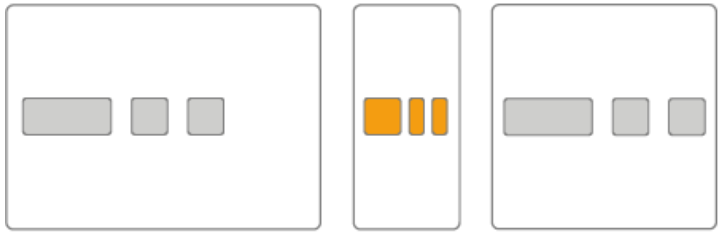


<Product by 안재욱>

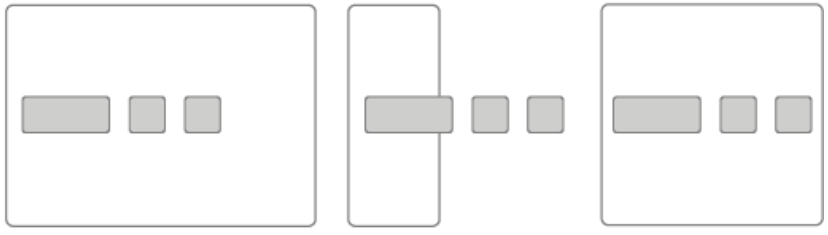
Flex Box Layout

- **flex item – 수평 방향 공간 너비 선언** (flex 통합 속성)

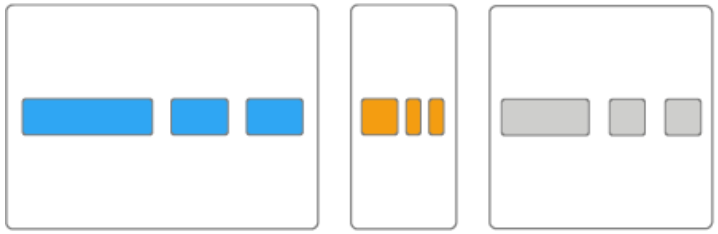
flex: initial(기본값)



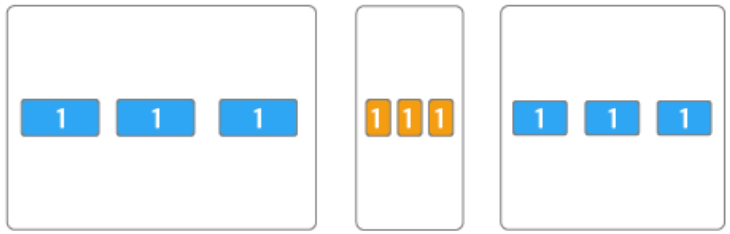
flex: none



flex: auto



flex: 1



display : grid

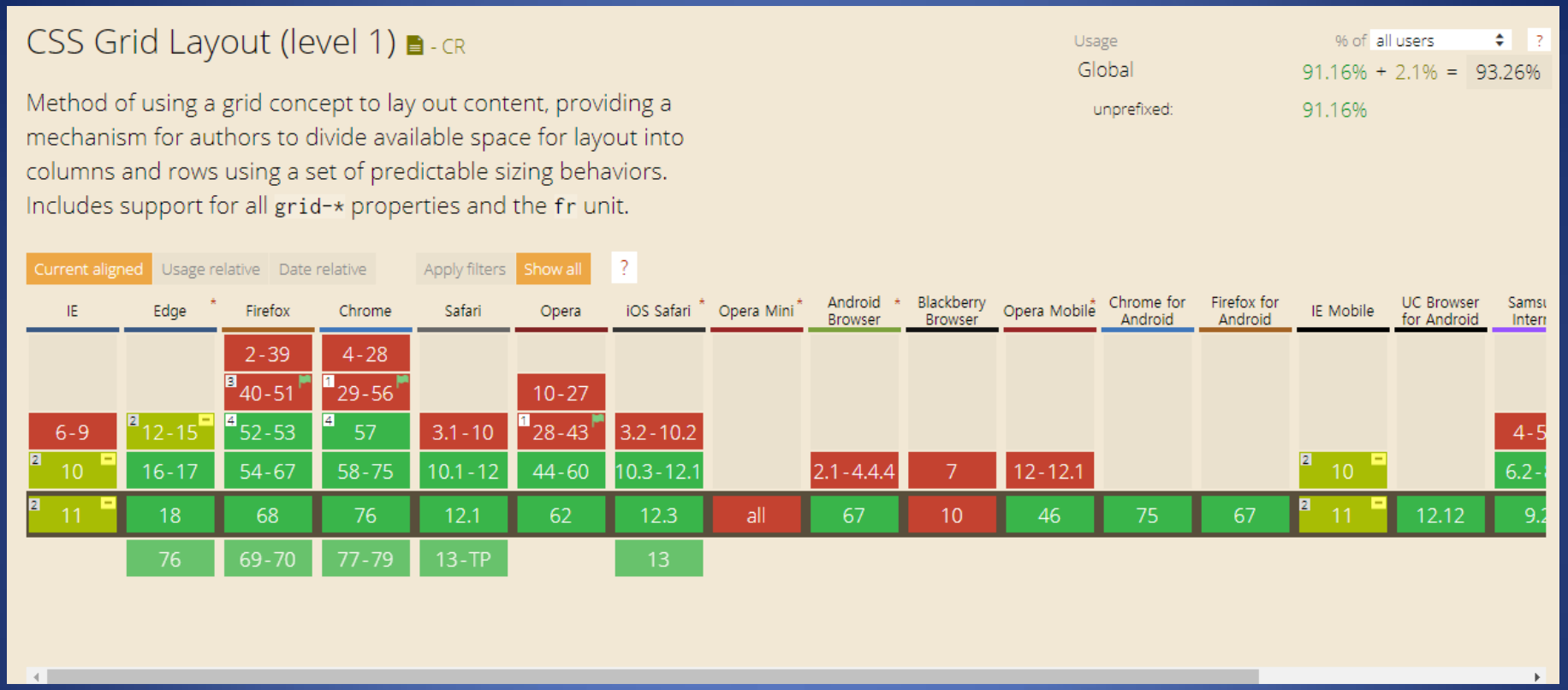
<Product by 안재욱>

Grid Layout

- CSS 그리드 레이아웃 (grid)

Grid Layout

- **CSS 그리드 레이아웃 (grid)**



- CSS 그리드 레이아웃은 모든 브라우저를 지원하고 있지 않지만 대부분의 최신 브라우저에서 지원하고 있는 상태
- **caniuse.com**에서 최신 브라우저에서 접목되고 있는 그리드 레이아웃의 분석자료 확인 가능

Grid Layout

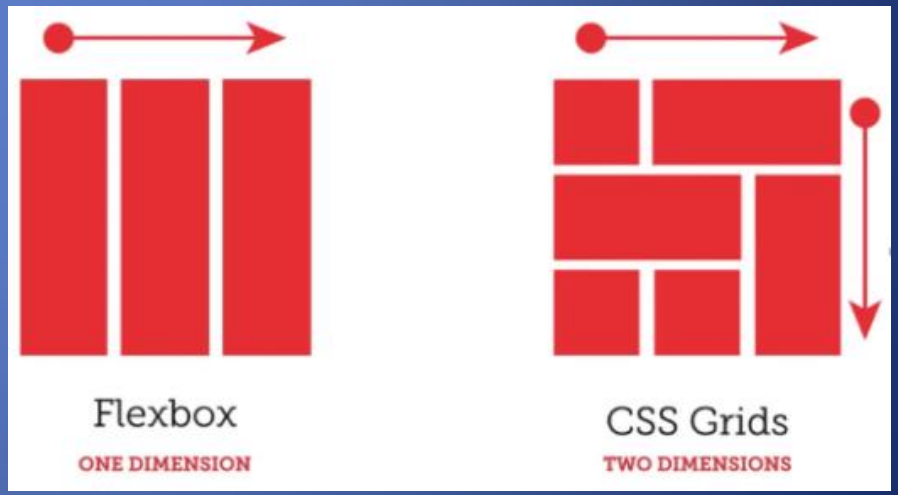
- **CSS 그리드 레이아웃 (grid)**
  - 가변 해상도가 다양해 질수록 기존의 레이아웃 개념(float / 기존의 display 방식 – block, inline, inline-block, table / position)만으로는 대처가 어려워지기 때문에 flex와 grid를 활용하면 더욱 다양한 공간 내부에서 접근이 가능

※ 플렉스 박스 레이아웃과 CSS 그리드 레이아웃의 차이점

- **플렉스 박스 레이아웃에서는 플렉스 아이템을 배치할 때 가로 또는 세로 중 하나를 주축으로 정해놓고 배치 (1차원 구성)**
- **반면, CSS 그리드 레이아웃에서는 그리드 항목을 배치할 때 가로와 세로를 모두 사용 (2차원 구성)**

Figure 1 *Exemplary Flex Layout Example*

Figure 2 *Exemplary Grid Layout Example*



Grid Layout

- **기본적인 그리드 레이아웃 만들기**
  - CSS 그리드 레이아웃 구성을 위해서는 레이아웃을 사용할 요소를 감싸는 '그리드 컨테이너'를 지정 후 그 내부에 그리드 항목을 넣어야 함.
  - 예를들어 .items 요소를 그리드 레이아웃으로 배치하기 위해서는 그 바깥을 감싸는 .wrapper 요소가 그리드 컨테이너가 됨

```
<div class="wrapper">
        <div class="items">Box_01</div>
        <div class="items">Box_02</div>
        <div class="items">Box_03</div>
        <div class="items">Box_04</div>
        <div class="items">Box_05</div>
</div>
```

Grid Layout

- **기본적인 그리드 레이아웃 만들기**

<display>

- 그리드 레이아웃을 만들 때 가장 우선 순위는 그리드 컨테이너를 구성하기
- .items를 배치할 외곽에 display 속성에 속성값으로 grid 또는 inline-grid로 지정

속성	설명
grid	그리드 레이아웃을 박스 레벨 요소로 정의
inline-grid	그리드 레이아웃을 인라인 레벨 요소로 정의

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

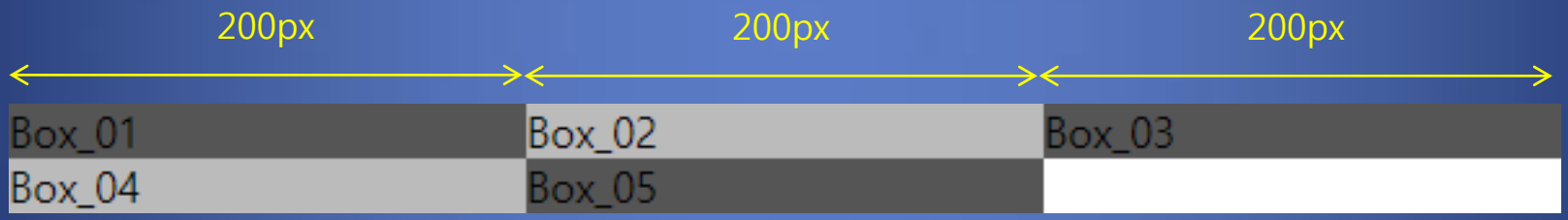
Grid Layout

- **기본적인 그리드 레이아웃 만들기**

<grid-template-columns>

- grid-template-columns 속성은 그리드 컨테이너 안의 항목을 몇 개의 컬럼(열)으로 배치할 지와 각 컬럼의 너비값을 지정
- 예를들어, 컬럼의 너비를 200px씩 지정하고 3개씩 배치한다면 아래와 같이 지정

```
.wrapper{
        display : grid;
        grid-template-columns : 200px 200px 200px;
}
```



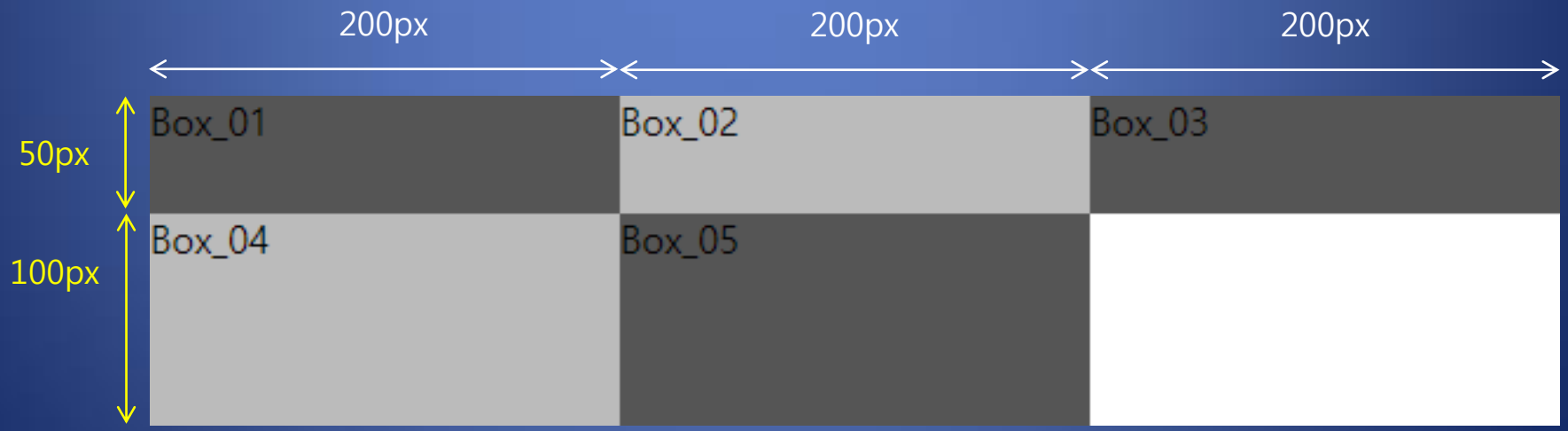
Grid Layout

- 기본적인 그리드 레이아웃 만들기

<grid-template-rows>

- 각 항목의 높이를 지정하려면 grid-template-rows 속성을 사용
- 예를들어, 아래와 같이 첫 번째 줄의 높이와 두 번째 줄의 높이를 각각 50px, 100px으로 지정

```
.wrapper{
        display : grid;
        grid-template-columns : 200px 200px 200px;
        grid-template-rows : 50px 100px;
}
```



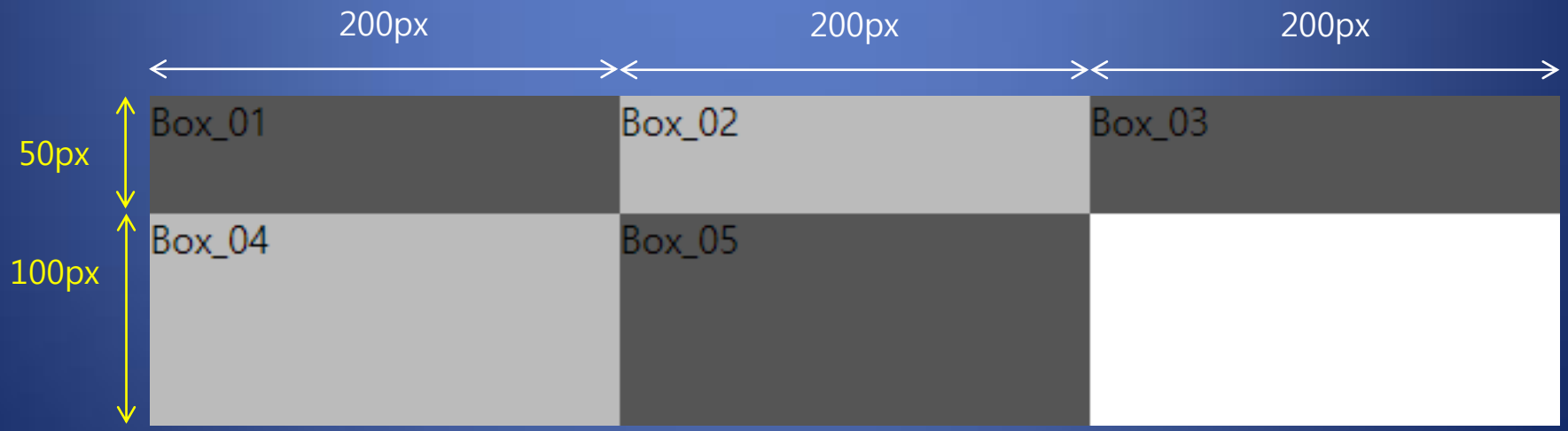
Grid Layout

- 기본적인 그리드 레이아웃 만들기

<grid-template-rows>

- 각 항목의 높이를 지정하려면 grid-template-rows 속성을 사용
- 예를들어, 아래와 같이 첫 번째 줄의 높이와 두 번째 줄의 높이를 각각 50px, 100px으로 지정

```
.wrapper{
        display : grid;
        grid-template-columns : 200px 200px 200px;
        grid-template-rows : 50px 100px;
}
```

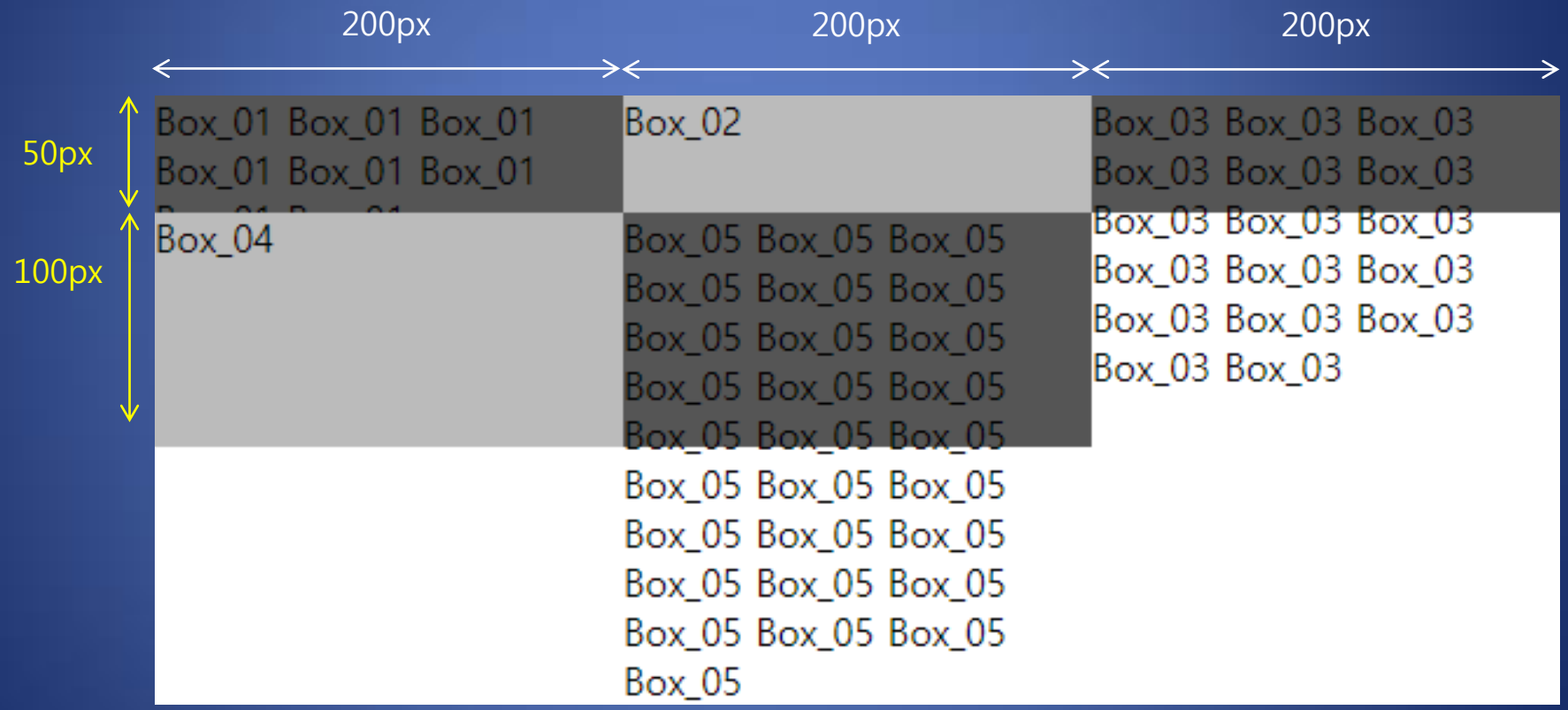


Grid Layout

- **기본적인 그리드 레이아웃 만들기**

<grid-template-rows>

- 만약 grid-template-rows를 설정하였으나 각 내부 박스의 서로 영역보다 내용이 많을 경우 자동 조정이 되지 않아서 콘텐츠가 영역 밖으로 이탈하는 경우가 있음



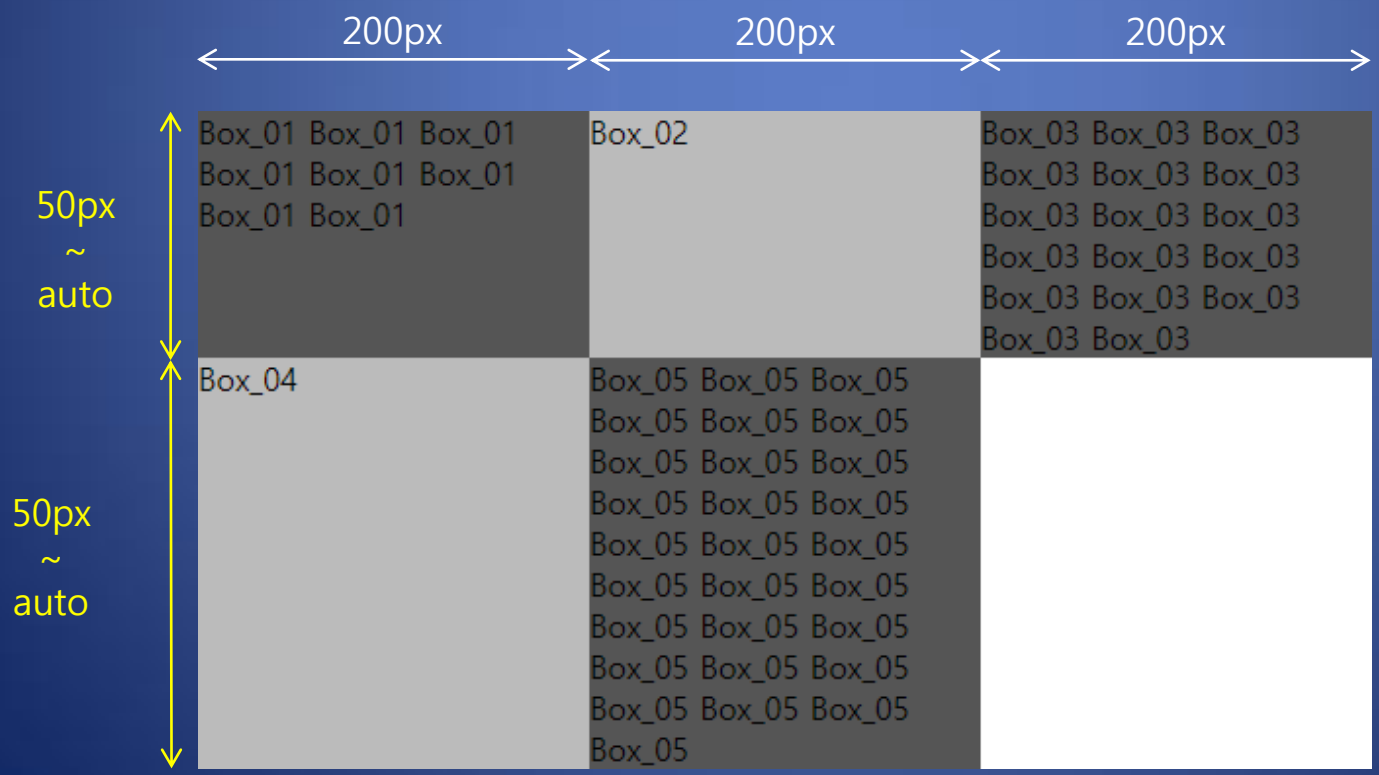
Grid Layout

- 기본적인 그리드 레이아웃 만들기

<grid-template-rows>

- 앞선 영역 이탈을 방지하기 위해서 사용하는 함수가 minmax() : 줄의 높이를 지정할 때 **최소값과 최대값을 지정**

```
.wrapper{
        display : grid;
        grid-template-columns : 200px 200px 200px;
        grid-template-rows : minmax(50px auto);
}
```



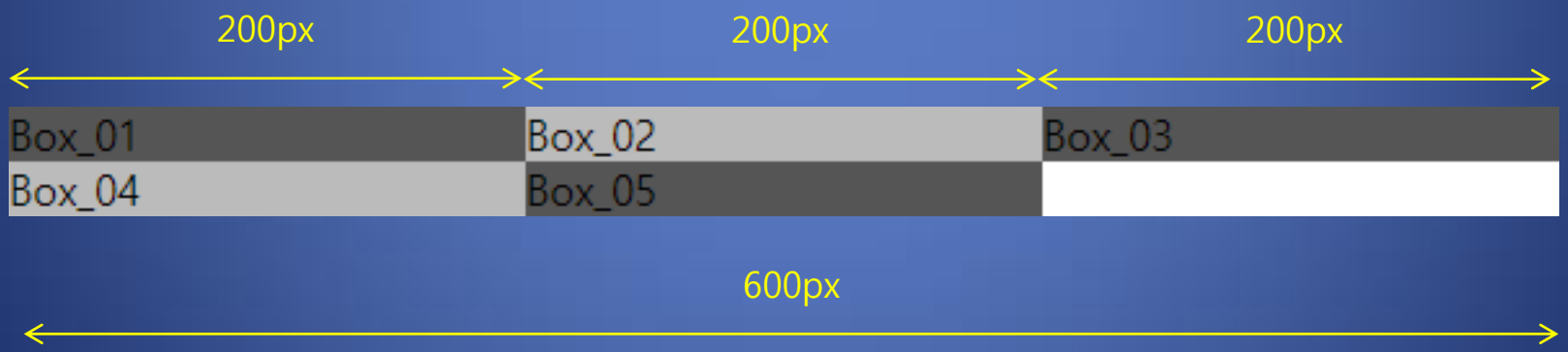
Grid Layout

- **기본적인 그리드 레이아웃 만들기**

<grid-template-columns에서 사용하는 단위 fr>

- 앞선 자료처럼 그리드 레이아웃에서 고정형으로 컬럼의 너비값을 px로 지정하면 크기가 항상 고정되기 때문에 반응형에서는 적합하지 않음
- 상대적인 크기를 지정할 수 있는 **fr**(Fraction : 부분 또는 분수) 단위를 사용
- 예를들어, 컬럼의 너비를 200px씩 지정하고 3개씩 배치한다면 아래와 같이 지정

```
.wrapper{
        width : 600px;
        display : grid;
        grid-template-columns : 1fr 1fr 1fr;
}
```



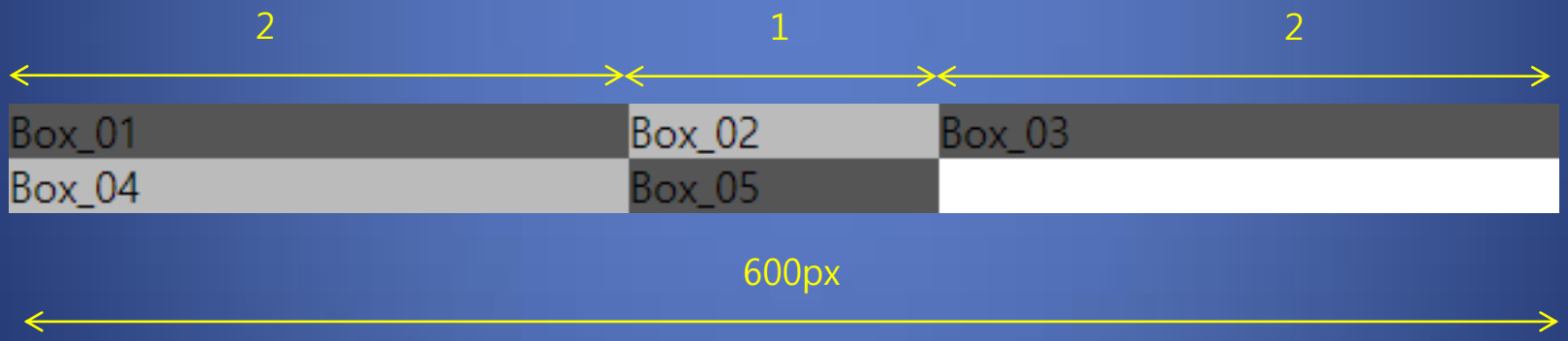
Grid Layout

- 기본적인 그리드 레이아웃 만들기

<grid-template-columns에서 사용하는 단위 fr>

- 상대적인 크기를 지정할 수 있는 **fr**(Fraction : 부분 또는 분수) 단위를 사용
- 예를들어, 컬럼의 너비를 동일 부모 공간 대비 **2:1:2로 지정하기 위해서**는 아래와 같이 작성

```
.wrapper{
        width : 600px;
        display : grid;
        grid-template-columns : 2fr 1fr 2fr;
}
```



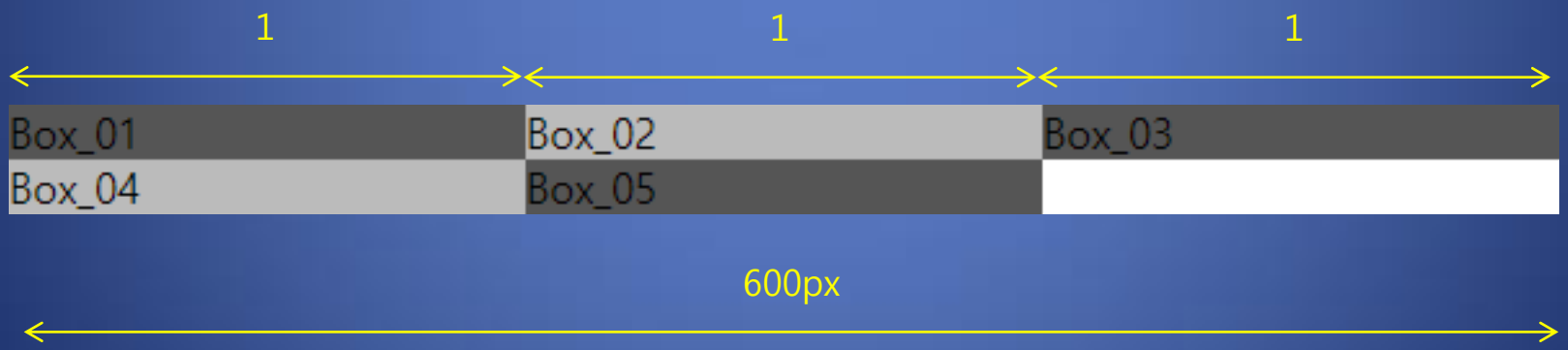
Grid Layout

- 기본적인 그리드 레이아웃 만들기

<grid-template-columns에서 사용하는 단위 fr>

- fr 단위를 사용할 때 모두 같은 값을 사용한다면 repeat() 함수를 사용해 간단하게 줄여 표현 가능
- repeat()은 그리드 레이아웃에서 사용하는 함수
- 예를들어, 컬럼의 너비를 1fr 3개씩 배치한다면 아래와 같이 지정

```
.wrapper{
        width : 600px;
        display : grid;
        grid-template-columns : repeat(3, 1fr);
}
```



Grid Layout

- 기본적인 그리드 레이아웃 만들기

<grid-column-gap, grid-row-gap, grid-gap>

- 기본적으로 각 그리드의 항목은 모두 간격을 갖고 있지 않음
- 항목과 항목의 사이를 조정하기 위해서는 아래와 같은 속성을 선언
- 속성값을 선언시에는 px, % 등의 값을 부여

속성	설명
grid-column-gap	컬럼과 컬럼 사이의 간격
grid-row-gap	줄과 줄 사이의 간격
grid-gap	컬럼과 줄 사이의 간격을 한번에 지정

```
.wrapper{
        width : 600px;
        display : grid;
        grid-template-columns : repeat(3, 1fr);
        grid-column-gap : 30px;  /*컬럼 사이의 간격 30px*/
        grid-row-gap : 20px;  /*줄 사이의 간격 20px*/
}
```

Grid Layout

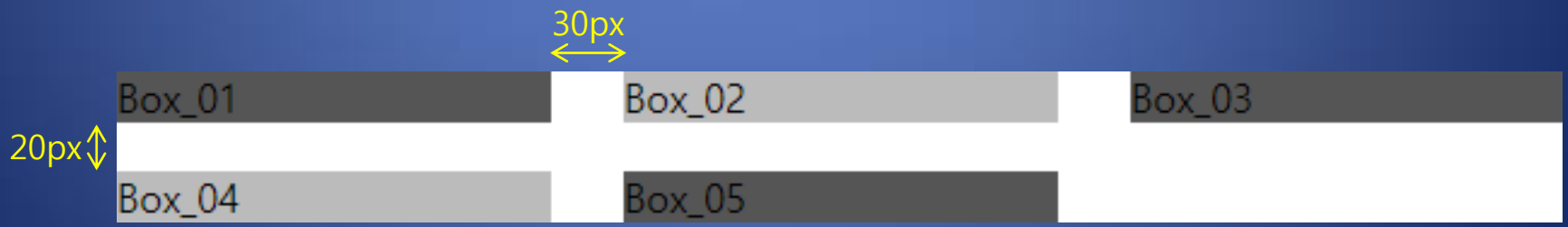
- 기본적인 그리드 레이아웃 만들기

<grid-column-gap, grid-row-gap, grid-gap>

- 컬럼의 간격과 줄의 간격을 통합하여 **grid-gap**으로 작성 가능
- 속성에 대한 값은 아래와 같이 정의됨

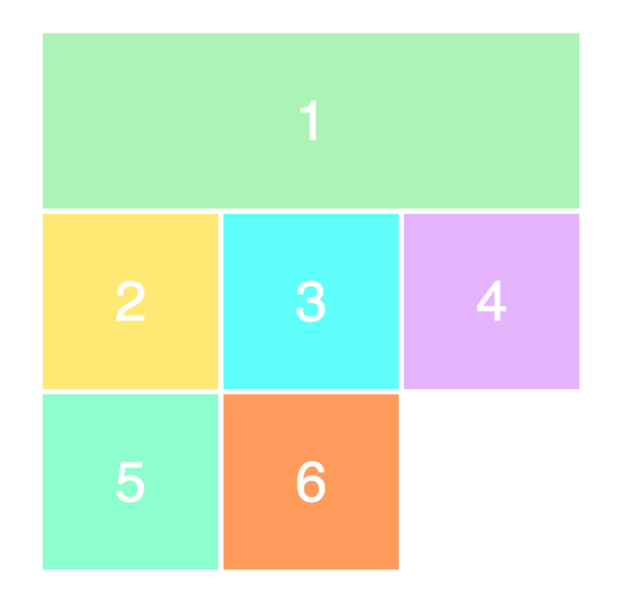
```
grid-gap : (grid-row-gap의 값) (grid-column-gap의 값);
```

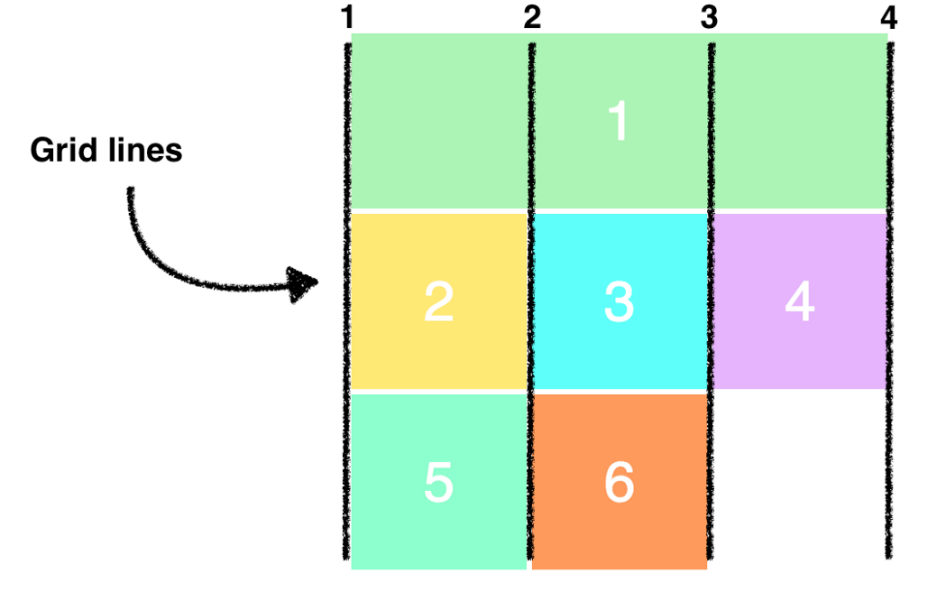
```
.wrapper{
        width : 600px;
        display : grid;
        grid-template-columns : repeat(3, 1fr);
/*
        grid-column-gap : 30px;  /*컬럼 사이의 간격 30px*/
        grid-row-gap : 20px;  /*줄 사이의 간격 20px*/
*/
        grid-gap : 20px 30px;  /*간격 조정통합 grid-gap : (줄 사이의 간격) (컬럼 사이의 간격)*/
}
```



Grid Layout

- **그리드 라인을 사용해 배치하기**
  - 그리드 레이아웃에는 보이지 않는 그리드 라인(Grif Line)이 포함되어 있음.
  - 3개의 컬럼과 3개의 줄로 이뤄진 그리드 레이아웃을 만들기 위해 수평 방향으로 4개의 그리드 라인과 수직 방향으로 4개의 그리드 라인으로 구성되어 있음.





Grid Layout

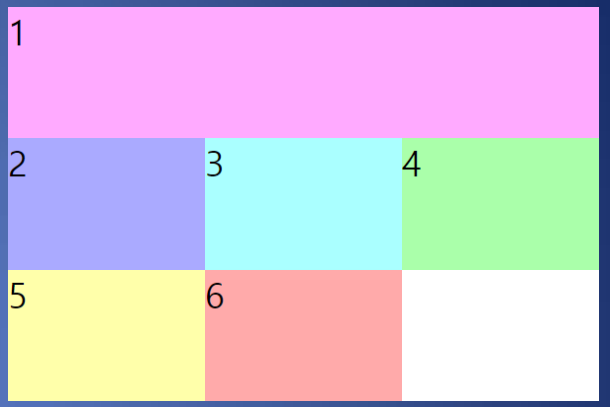
- **그리드 라인을 사용해 배치하기**
  - 그리드 라인으로 그리드 항목을 배치가 가능한데 아래와 같은 속성을 적용

속성	설명
grid-column-start	컬럼 시작 번호
grid-column-end	컬럼 끝 번호
grid-column	컬럼 시작/끝 번호
grid-row-start	줄 시작 번호
grid-row-end	줄 끝 번호
grid-row	줄 시작/끝 번호

Grid Layout

- 그리드 라인을 사용해 배치하기
  - 아래와 같은 구성을 만들기 위해서 코딩 구조를 작성(다음 장에 계속)

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



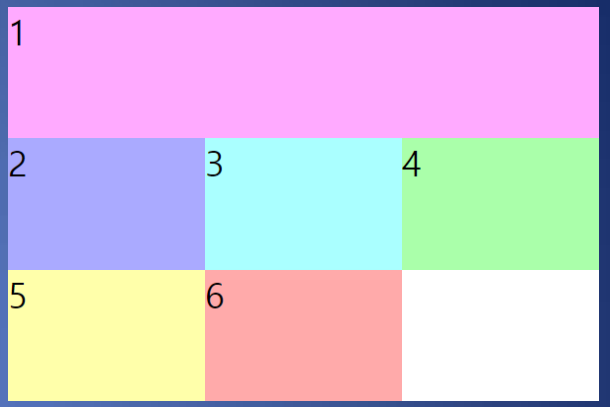

```
.container{
        width : 600px;
        height : 400px;
        display : grid;
        grid-template-columns : repeat(3, 1fr);
}

.item{width : auto; height : auto; font-size : 36px;}
.item:nth-child(1){background : #ffaaff;}
.item:nth-child(2){background : #aaaaff;}
.item:nth-child(3){background : #aaffff;}
.item:nth-child(4){background : #aaffaa;}
.item:nth-child(5){background : #ffffaa;}
.item:nth-child(6){background : #ffaaaa;}
```

Grid Layout

- 그리드 라인을 사용해 배치하기
  - 첫 번째 .item의 컬럼 라인은 1번부터 4번까지 차지하므로 다음과 같이 지정

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

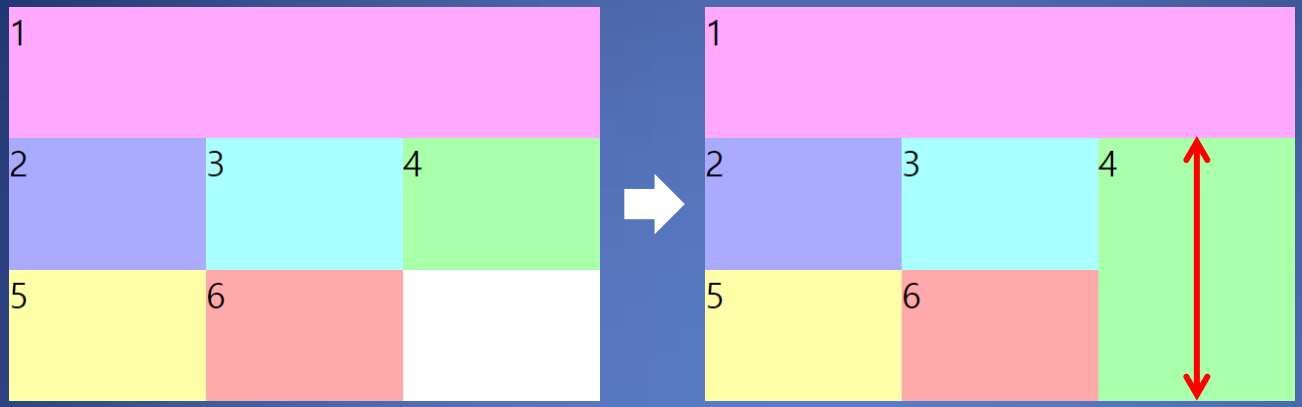



```
.container{
        width : 600px;
        height : 400px;
        display : grid;
        grid-template-columns : repeat(3, 1fr);
}

.item{width : auto; height : auto; font-size : 36px;}
.item:nth-child(1){background : #ffaaff; grid-column : 1/4;}
.item:nth-child(2){background : #aaaaff;}
.item:nth-child(3){background : #aaffff;}
.item:nth-child(4){background : #aaffaa;}
.item:nth-child(5){background : #ffffaa;}
.item:nth-child(6){background : #ffaaaa;}
```

Grid Layout

- 그리드 라인을 사용해 배치하기
  - 네 번째 .item의 컬럼 라인은 3번부터 4번까지 차지하고 줄 라인은 2번부터 4번까지 확장



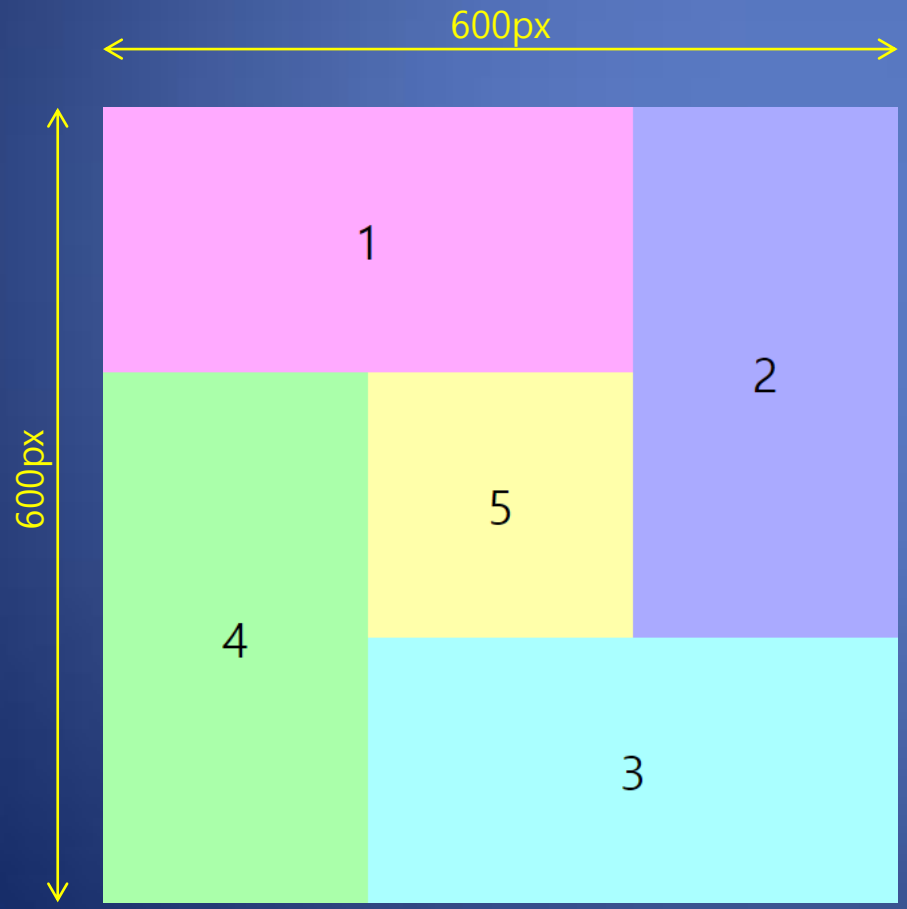

```
.container{
        width : 600px;
        height : 400px;
        display : grid;
        grid-template-columns : repeat(3, 1fr);
}

.item{width : auto; height : auto; font-size : 36px;}
.item:nth-child(1){background : #ffaaff; grid-column : 1/4;}
.item:nth-child(2){background : #aaaaff;}
.item:nth-child(3){background : #aaffff;}
.item:nth-child(4){background : #aaffaa; grid-row-start : 2; grid-row-end : 4; grid-column : 3/4;}
.item:nth-child(5){background : #ffffaa;}
.item:nth-child(6){background : #ffaaaa;}
```

Grid Layout

[실습 응용-01. 그리드 레이아웃을 활용한 공간 구성하기]

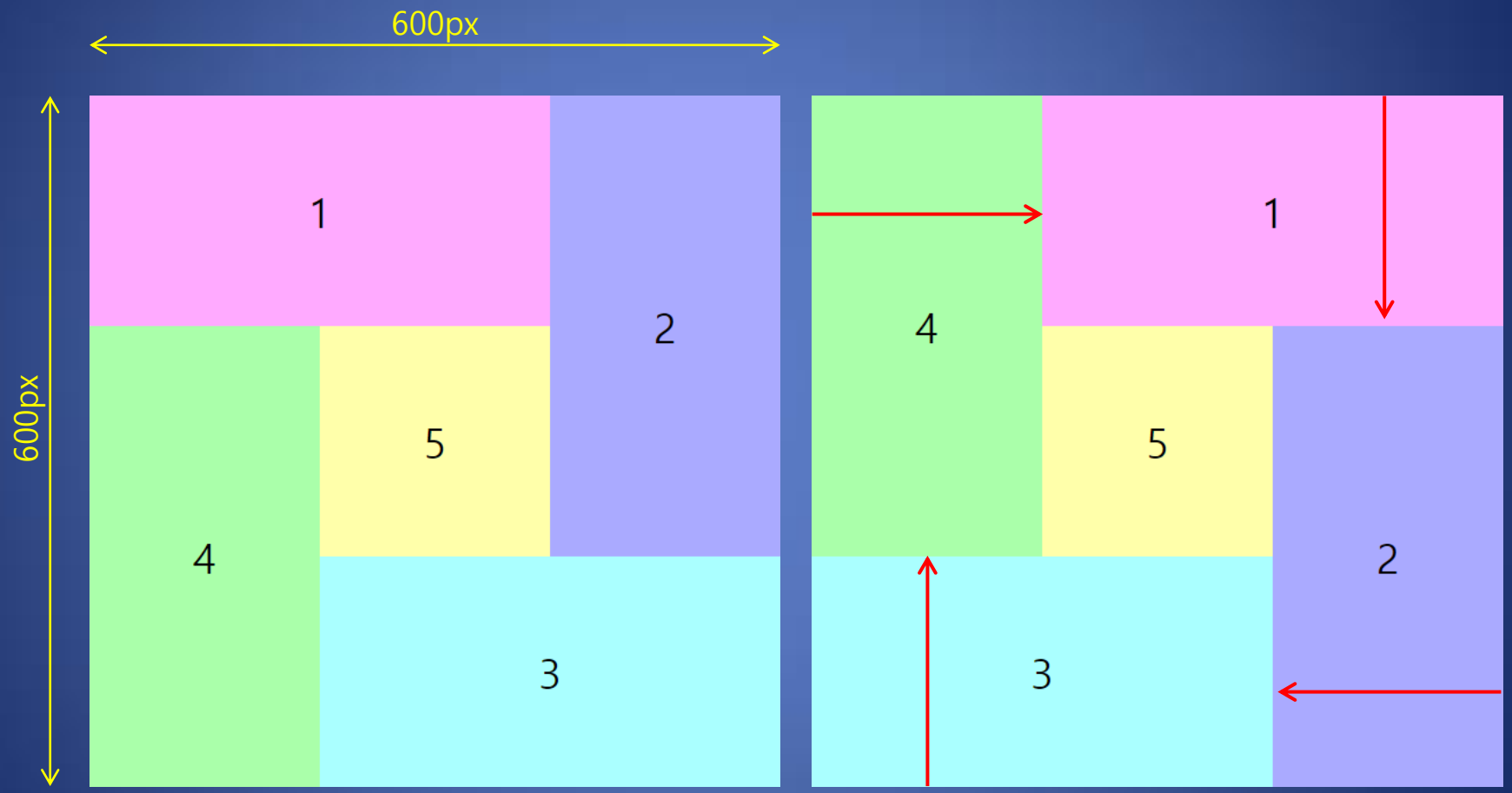
- 가로 600px, 세로 600px 사이즈의 외부박스를 구성하고 내부에 1~5번까지 순서대로 작성하여 내부의 공간을 채우시오. 박스의 영역은 추후 반응형을 기준으로 3등분하여 그리드 라인을 구성하시오.
- 내부 박스의 색상은 1번 박스부터 5번 박스까지 #ffaaff, #aaaaff, #aaffff, #aaffaa, #ffffaa로 적용하시오.
- 박스 내부 글자는 각 영역을 기준으로 중앙배치(table, table-cell)하여 적용하시오. (폰트 크기는 36px, 폰트 색상은 #000;)



Grid Layout

[실습 응용-02. 그리드 레이아웃을 활용한 공간 구성하기]

- 앞서 구성한 외부 박스에 마우스 오버시 1번, 2번, 3번, 4번 박스의 위치를 이동시키시오.



<Default Mode>

<MouseOver Mode>

Grid Layout

[실습 응용-03. 그리드 레이아웃을 활용한 몬드리안 작품 만들기]

- 외부 박스 가로 820px, 세로 820px을 활용하여 내부 공간을 그리드 레이아웃을 활용하여 몬드리안 작품을 구성하세요. (외/내부 박스 모두 상하좌우 border 10px solid #000 포함)
- 색상 적용은 red, blue, yellow, white 적용