> 14. DIV,span 관련 css

DIV태그는 영역을 블록 형태로 분할 하는 대표적인 태그다.

span태그는 영역을 인라인 형태로 분할 하는 대표적인 태그다.

블록 태그는 한줄을 다 차지하는 줄단위로 영역을 가지고, 인라인 태그는 글씨 쓰는 것처럼 데이터를 감싸서 옆으로 기술된다.

HTML 태그는 보통 두 가지 주요 카테고리인 "블록(Block)" 및 "인라인(Inline)"으로 나뉩니다. 이 두 카테고리는 요소가 페이지에서 어떻게 표시되고 어떤 형식을 가지는지를 정의합니다.

블록(Block) 요소: 블록 요소는 페이지의 레이아웃을 만드는 데 사용되며, 보통 새로운 줄에서 시작하여 가로 전체의 공간을 차지합니다. 주요 블록 요소로는 다음과 같은 것들이 있습니다:

: 구획을 나누기 위해 사용되는 일반적인 블록 요소.

<p>: 단락을 나타내는 블록 요소.

<h1>, <h2>, <h3>, ... <h6>: 제목 요소.

, <ol>, <li>: 목록 요소.

<table>, <tr>, <td>: 표를 나타내는 요소.

인라인(Inline) 요소: 인라인 요소는 블록 요소 안에 들어갈 수 있으며, 새로운 줄에서 시작하지 않고 필요한 공간만 차지합니다. 주요 인라인 요소로는 다음과 같은 것들이 있습니다:

: 인라인 컨테이너 요소로, 텍스트나 다른 인라인 요소를 둘러싸기 위해 사용됩니다.

<a>: 하이퍼링크를 만드는 데 사용되는 인라인 요소.

, : 텍스트 강조를 나타내는 요소.

<img>: 이미지를 나타내는 요소.

: 줄 바꿈 요소.

이러한 블록과 인라인 요소의 사용은 웹 페이지의 레이아웃 및 내용 구조를 설계하는 데 중요합니다. 블록 요소는 주로 구획을 만들고 레이아웃을 지정하며, 인라인 요소는 텍스트 스타일링 및 하이퍼링크를 만드는 데 사용됩니다.

모든 블록 태그는 영역에 적용할 수 있는 다음과 같은 css를 가지고 있다.

border-radius: 요소의 모서리를 둥글게 만드는 속성.

background: 요소의 배경 색상 또는 이미지를 설정하는 속성.

padding: 요소 내부의 여백을 지정하여 요소 내부 콘텐츠와 테두리 사이의 간격을 조절하는 속성.

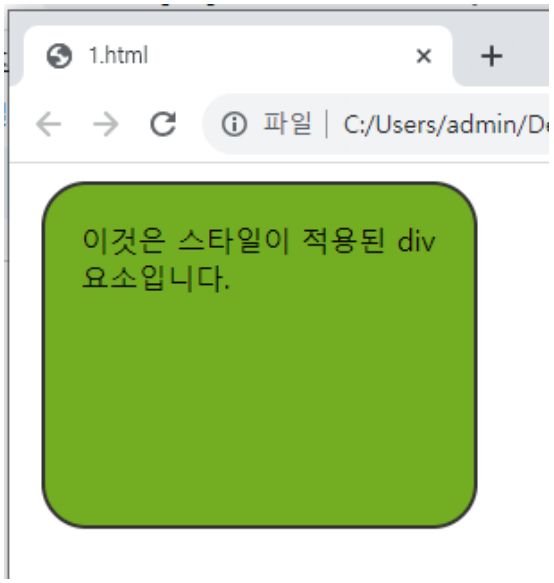
width: 요소의 가로 크기를 설정하는 속성.

height: 요소의 세로 크기를 설정하는 속성.

margin: 요소 주위의 외부 여백을 지정하여 요소 사이의 간격을 조절하는 속성.

border: 요소 주위에 테두리를 추가하는 속성으로, 두께, 스타일 및 색상을 설정하여 요소 외형을 변경하는 데 사용 됩니다. 다음은 간단한 예제이다.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        div {
            border-radius: 25px; /* 모서리를 둥글게 만듭니다. */
            background: #73AD21; /* 배경 색상을 녹색으로 설정합니다. */
            padding: 20px; /* 내부 여백을 20px로 설정합니다. */
            width: 200px; /* 너비를 200px로 설정합니다. */
            height: 150px; /* 높이를 150px로 설정합니다. */
            margin: 10px; /* 마진을 10px로 설정합니다. */
            border: 2px solid #333; /* 2px 두께의 실선 테두리, 색상은 #333 */
        }
    </style>
</head>
```




```
<body>
<div>이것은 스타일이 적용된 div 요소입니다.</div>
</body>
</html>
```

padding 속성에 값을 지정할 때, 여러 가지 방식으로 값을 설정할 수 있습니다. 아래의 예제에서는 각각의 padding 설정 방법을 설명합니다:

padding: 10px;: 이 설정은 모든 네 면(위, 오른쪽, 아래, 왼쪽)에 10px의 내부 여백을 설정합니다. 네 면이 모두 동일한 값으로 설정됩니다.

padding: 10px 20px;: 이 설정은 위와 아래 면에 10px의 내부 여백을, 그리고 왼쪽과 오른쪽 면에 20px의 내부 여백을 설정합니다.

padding: 10px 20px 30px;: 이 설정은 위 면에 10px, 왼쪽과 오른쪽 면에 20px, 아래 면에 30px의 내부 여백을 설정합니다.

padding: 10px 20px 30px 40px;: 이 설정은 위 면에 10px, 오른쪽 면에 20px, 아래 면에 30px, 왼쪽 면에 40px의 내부 여백을 설정합니다.

따라서 padding 속성을 사용하여 각 면의 내부 여백을 설정할 때, 순서에 따라 위, 오른쪽, 아래, 왼쪽 순서로 값을 지정하면 됩니다.

다음과 같은 방법도 있다.

padding-top: 요소의 위쪽 패딩을 설정합니다.

padding-right: 요소의 오른쪽 패딩을 설정합니다.

padding-bottom: 요소의 아래쪽 패딩을 설정합니다.

padding-left: 요소의 왼쪽 패딩을 설정합니다.

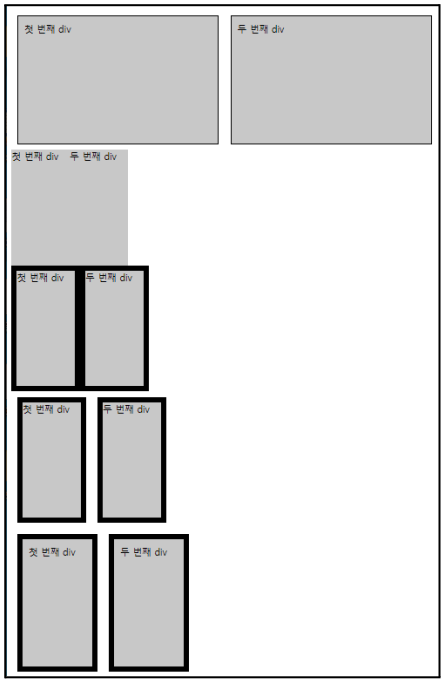
padding,margin은 다음과 같은 형태로 사용하면 된다.

```
div {
      padding-top: 10px; /* 위쪽 패딩을 10px로 설정 */
      padding-right: 20px; /* 오른쪽 패딩을 20px로 설정 */
      padding-bottom: 30px; /* 아래쪽 패딩을 30px로 설정 */
      padding-left: 40px; /* 왼쪽 패딩을 40px로 설정 */
      padding: 10px; /* 이 값만 적용됨 */
      padding: 10px 20px; /* 이 값만 적용됨 */
      padding: 10px 20px 30px; /* 이 값만 적용됨 */
      padding: 10px 20px 30px 40px; /* 이 값만 적용됨 */
}
div {
```

```
    margin-top: 10px; /* 위쪽 마진을 10px로 설정 */
    margin-right: 20px; /* 오른쪽 마진을 20px로 설정 */
    margin-bottom: 30px; /* 아래쪽 마진을 30px로 설정 */
    margin-left: 40px; /* 왼쪽 마진을 40px로 설정 */
    margin: 10px; /* 이 값만 적용됨 */
    margin: 10px 20px; /* 이 값만 적용됨 */
    margin: 10px 20px 30px; /* 이 값만 적용됨 */
    margin: 10px 20px 30px 40px; /* 이 값만 적용됨 */
}
```

border속성 사용방법은 다음과 같다.

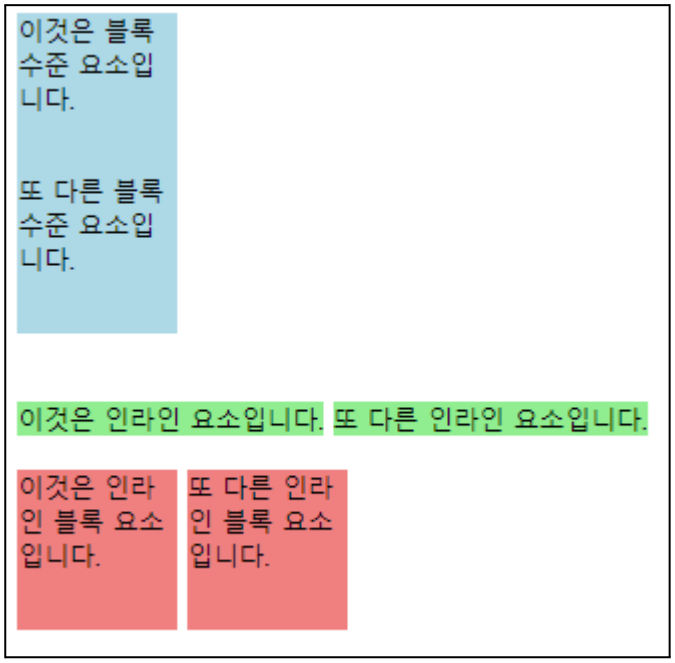
```
div {
    border-top: 10px solid red; /* 위쪽 테두리를 10px 높이로 설정하고 빨간색*/
    border-right: 20px solid blue; /*오른쪽 테두리를 20px 너비로 설정하고 파란색*/
    border-bottom: 30px solid green; /*아래쪽 테두리를 30px 높이로 설정하고 녹색*/
    border-left: 40px solid orange; /*왼쪽 테두리를 40px 너비로 설정하고 주황색*/
    border: 5px dashed purple; /* 모든 테두리를 5px 높이의 점선 */
}
```



width height으로 크기를 설정 할 수 있다. div 하나의 크기를 100px로 설정 한다고 해서 해당 div크기가 100이 되는 보장은 없다. width를 100으로 설정해도 border, margin, padding 크기에 따라 실제 width값이 달라진다. border, margin, padding 값은 실제 width 값에 영향을 준다. 왼쪽 이미지에서 2개의 div 크기가 명확한것은 2번째 div이다. border, margin, padding이 없어서 크기가 명확하게 100px이지만 나머지는 border, margin, padding가 존재하면 해당 속성의 크기 만큼 width 속성크기가 커진다.

```html
<!DOCTYPE html>
<html>
<head>
  <style>
    .container {
      display: flex;
    }
    .box {
      width: 45%;
      height: 200px;
      background-color: #ccc;
      border: 1px solid #000;
      margin: 10px; /* 마진 추가 */
      padding: 10px; /* 패딩 추가 */
    }
    .box1 {
      width: 100px; /* 화면 너비의 일부만 차지하도록 설정 */
      height: 200px;
      background-color: #ccc;
    }
    .box2 {
      width: 100px; /* 화면 너비의 일부만 차지하도록 설정 */
      height: 200px;
      background-color: #ccc;
      border: 10px solid #000;
    }
    .box3 {
      width: 100px; /* 화면 너비의 일부만 차지하도록 설정 */
      height: 200px;
      background-color: #ccc;
      border: 10px solid #000;
      margin: 10px; /* 마진 추가 */
    }
    .box4 {
      width: 100px; /* 화면 너비의 일부만 차지하도록 설정 */
      height: 200px;
      background-color: #ccc;
      border: 10px solid #000;
      margin: 10px; /* 마진 추가 */
      padding: 10px; /* 패딩 추가 */
    }
  </style>
</head>
<body>
  <div class="container">
```

```
    <div class="box">첫 번째 div</div>
    <div class="box">두 번째 div</div>
  </div>
  <div class="container">
    <div class="box1">첫 번째 div</div>
    <div class="box1">두 번째 div</div>
  </div>
  <div class="container">
    <div class="box2">첫 번째 div</div>
    <div class="box2">두 번째 div</div>
  </div>
  <div class="container">
    <div class="box3">첫 번째 div</div>
    <div class="box3">두 번째 div</div>
  </div>
  <div class="container">
    <div class="box4">첫 번째 div</div>
    <div class="box4">두 번째 div</div>
  </div>
</body>
</html>
```



각 "display" 속성 값에 대한 간단한 요약은 다음과 같습니다:

block: 이 값은 요소를 블록 레벨 요소로 표시하며, 요소는 다음 줄에 표시되고 가로 너비 전체를 차지합니다. 다른 요소와 수직으로 쌓입니다.

inline: 이 값은 요소를 인라인 요소로 표시하며, 요소는 다른 인라인 요소와 같은 줄에 표시되며 필요한 만큼의 너비만 차지하여 width, height 같은 속성이 적용되지 않는다.

inline-block: 이 값은 요소를 인라인 블록 요소로 표시하며, 다른 인라인 요소와 같은 줄에 표시되지만 가로 너비와 세로 높이를 지정할 수 있습니다.

none: 이 값은 요소를 화면에서 숨기는 역할을 합니다. 요소는 화면에 나타나지 않고, 레이아웃에 차지하는 공간도 없습니다. 이것은 JavaScript를 사용하여 상호작용하는 요소를 처리할 때 유용하게 쓰입니다.

```
<!DOCTYPE html>
<html>
<head>
<style>
  .block-example {
    display: block;    width: 100px;
    height: 100px;    background-color: lightblue;
  }
  .inline-example {
    display: inline;    width: 100px;
    height: 100px;    background-color: lightgreen;
  }
  .inline-block-example {
    display: inline-block;    width: 100px;
    height: 100px;    background-color: lightcoral;
  }
  .none-example {
    display: none; /* 이 부분을 추가하여 요소를 숨깁니다. */
  }
</style>
</head>
<body>
  <div class="block-example">이것은 블록 수준 요소입니다.</div>
  <div class="block-example">또 다른 블록 수준 요소입니다.</div><br><br>
  <div class="inline-example">이것은 인라인 요소입니다.</div>
  <div class="inline-example">또 다른 인라인 요소입니다.</div><br><br>
  <div class="inline-block-example">이것은 인라인 블록 요소입니다.</div>
  <div class="inline-block-example">또 다른 인라인 블록 요소입니다.</div>
  <div class="none-example">이것은 숨겨진 요소입니다.</div> <!-- 이 부분을
추가하여 숨긴 요소를 나타냅니다. -->
</body>
</html>
```

은 HTML에서 인라인 텍스트나 요소를 그룹화하고 스타일 또는 동작을 적용하기 위해 사용되고 모양을 꾸미는 속성들이 무시되는 시각적 요소입니다. 글관련 내용을 묶을때 사용한다. span과 div의 차이는 display 속성이 inline과 block 이라는 것이다.

이 문장은 스팬 요소를 포함합니다. 너비와 높이 설정은 적용 안됨

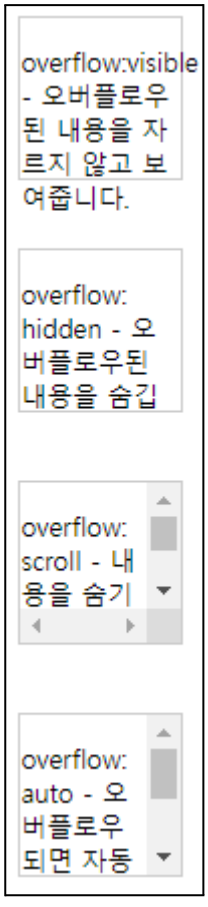
```
<!DOCTYPE html>
<html>
<head>
<style>
  .custom-span {
    width: 100px;
    height: 50px;
```

```
    background-color: lightblue;
  }
</style>
</head>
<body>
  <p>이 문장은 <span class="custom-span">스팬 요소</span>를 포함합니다. 너비와 높이 설정은 적용 안됨</p>
</body>
</html>
```

visibility"는 CSS 속성 중 하나로, HTML 요소의 가시성 여부를 제어하는 데 사용됩니다. 이 속성을 사용하면 요소를 숨기거나 다시 표시할 수 있습니다.
"visibility" 속성은 두 가지 주요 값이 있습니다:
visible: 기본값입니다. 요소가 보이도록 설정됩니다.
hidden: 요소가 숨겨집니다. 그러나 요소의 공간은 여전히 유지됩니다. 다시 말해, 요소는 화면에서 사라지지만 다른 요소의 레이아웃에는 영향을 주지 않습니다.
아래 코드를 실행한 결과를 보면 start와 end 사이에 div를 어떻게 보여 줄지 결정한다. visibility:hidden 같은 경우 존재하는데 안보여주는 것이고 display:none 경우에는 화면에 존재하지도 않는 상태이다.


```
<!DOCTYPE html>
<html>
<head>
<style>
  .hidden-visibility {
    visibility: hidden;
  }
  .none-display {
    display: none;
  }
  .visible {
    border: 1px solid #000;
```

```
    padding: 10px;
    margin: 10px;
    background-color: red;
  }
</style>
</head>
<body>
  <div class="visible">visibility: hidden start</div>
  <div class="visible hidden-visibility">이 요소는 visibility: hidden</div>
  <div class="visible">visibility: hidden end</div>
  <div class="visible">display: none start</div>
  <div class="visible none-display">이 요소는 display: none</div>
  <div class="visible">display: none end</div>
</body>
</html>
```



overflow-x 및 overflow-y: 수평 및 수직 오버플로우를 각각 제어하는 속성입니다.

overflow: 수평 및 수직 오버플로우를 동시에 제어하는 속성입니다.

visible: 오버플로우된 내용을 자르지 않고 보여줍니다.

hidden: 오버플로우된 내용을 숨깁니다.

scroll: 오버플로우된 내용을 숨기고 스크롤바를 표시합니다.

auto: 내용이 오버플로우되면 자동으로 스크롤바를 표시합니다.

```
<!DOCTYPE html>
<html>
<head>
    <style>
        .container {
            width: 100px; /* 컨테이너의 너비를 100px로 설정 */
            height: 100px; /* 컨테이너의 높이를 100px로 설정 */
            border: 1px solid #ccc; /* 테두리를 그림 */
        }
        .overflow-visible {
            overflow: visible; /* 오버플로우된 내용을 자르지 않고 보여줍니다. */
        }
        .overflow-hidden {
            overflow: hidden; /* 오버플로우된 내용을 숨깁니다. */
        }
        .overflow-scroll {
```

```
            overflow: scroll; /* 오버플로우된 내용을 숨기고 스크롤바 표시*/
        }
        .overflow-auto {
            overflow: auto; /* 내용이 오버플로우되면 자동으로 스크롤바 표시*/
        }
    </style>
</head>
<body>
    <div class="container overflow-visible">
        <p>overflow:visible - 오버플로우된 내용을 자르지 않고 보여줍니다.</p>
    </div>
<br><br>
    <div class="container overflow-hidden">
        <p>overflow: hidden - 오버플로우된 내용을 숨깁니다.</p>
    </div>
<br><br>
    <div class="container overflow-scroll">
        <p>overflow: scroll - 내용을 숨기고 수평 및 수직 스크롤바를 표시</p>
    </div>
<br><br>
    <div class="container overflow-auto">
        <p>overflow: auto - 오버플로우되면 자동으로 수평 수직 스크롤바 표시</p>
    </div>
</body>
</html>
```

position 속성 "position" 속성의 주요 값들을 간단히 정리해 드리겠습니다:

static: 이 값은 요소를 일반 문서 흐름에 따라 배치하며 위치를 추가로 조정할 수 없습니다.

relative: 이 값은 요소를 일반 문서 흐름에 따라 배치하면서 상대적으로 위치를 조정할 수 있습니다.

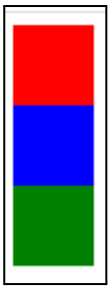
absolute: 이 값은 요소를 가장 가까운 위치 지정 부모을 기준으로 배치하며, 일반 문서 흐름에서 제거됩니다.

fixed: 이 값은 요소를 뷰포트(브라우저 창)를 기준으로 배치하며 스크롤에 관계 없이 위치가 고정됩니다.

sticky: 이 값은 요소를 일반 문서 흐름에 따라 배치하면서 스크롤되면 지정된 위치에 고정됩니다.

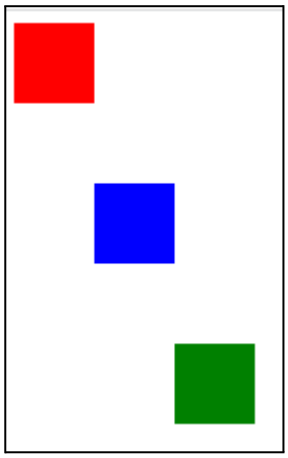
이러한 "position" 속성 값들을 사용하여 웹 페이지의 레이아웃을 제어하고 요소를 원하는 위치에 배치할 수 있습니다.

다음은 static 예제이다. static은 top,left속성이 전혀 적용되지 않는다.



```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
  width: 50px;  height: 50px;
  position: static;
}
.box1 {
  background: red;
  top: 0px;  left: 0px;
}
.box2 {
  background: blue;
  top: 50px;  left: 50px;
}
.box3 {
  background: green;
  top: 100px;  left: 100px;
}
</style>
</head>
<body>
<div class="box box1"></div>
<div class="box box2"></div>
<div class="box box3"></div>
</body>
</html>
```

다음은 relative예제이다. relative는 static이였을때의 위치를 기준으로 top,left 속성이 움직인다.



```
<!DOCTYPE html>
<html>
<head>
<style>
.box {
width: 50px;  height: 50px;
position: relative;
}
.box1 {
background: red;
top: 0px;  left: 0px;
}
.box2 {
```

```
  background: blue;
  top: 50px;  left: 50px;
}
.box3 {
  background: green;
  top: 100px;  left: 100px;
}
</style>
</head>
<body>
<div class="box box1"></div>
<div class="box box2"></div>
<div class="box box3"></div>
</body>
</html>
```

fixed를 사용하면 스크롤이 움직여도 div가 고정되어 있는 것을 확인할 수 있다.

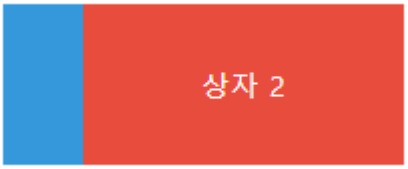

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
margin: 0;
padding: 0;
}
.header {
background-color: #333;
color: #fff;
text-align: center;
padding: 10px;
position: fixed;
top: 0;
  left: 0;
  width: 100%;
}
.logo {
  font-size: 24px;
  font-weight: bold;
}
.content {
  padding: 80px 20px 20px;
}
.ad {
  background-color: #f2f2f2;
  width: 200px;
```

```
    height: 400px;
    position: fixed;
    top: 0;
    right: 0;
    padding: 20px;
  }
</style>
</head>
<body>
<div class="header">
  <div class="logo">My Website</div>
</div>
<div class="content">
  <h1>내 웹사이트에 오신 것을 환영합니다</h1>
  <p>이것은 고정된 헤더의 간단한 예제입니다.</p>
  <!-- 내용 부분에 더 많은 내용을 추가하세요 -->
</div>
<div class="ad">
  <h2>광고</h2>
  <p>여기에 광고 내용을 추가하세요.</p>
</div>
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
br><br><br>
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
br><br><br>
맨하단
</body>
</html>
```

stycky속성의 경우 평상시는 왼쪽 이미지 처럼 설정된 곳에 있다가 스크롤되면 fixed처럼 상단에 고정된다.


```
<!DOCTYPE html>
<html>
<head>
<style>
```

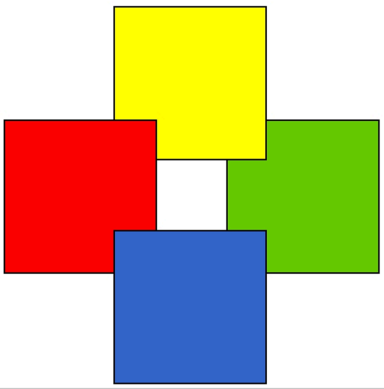
```
  body {
    margin: 0;    padding: 0;
    font-family: Arial, sans-serif;
  }
  .header {
    background-color: #333;
    color: #fff;    text-align: center;
    padding: 10px;    position: sticky;
    top: 0;
  }
  .logo {
    font-size: 24px;    font-weight: bold;
  }
  .content {
    padding: 80px 20px;
  }
  .sticky-element {
    background-color: #f2f2f2;
    padding: 10px;    position: sticky;
    top: 60px; /* 스티키가 적용될 위치 (헤더 아래) */
  }
</style>
</head>
<body>
<div class="header">
  <div class="logo">My Sticky Website</div>
</div>
<div class="content">
  <h1>스티키 예제</h1>
  <p>이것은 스티키 요소를 사용한 간단한 예제입니다.</p>
  <div class="sticky-element">
    <h2>스티키 요소</h2>
    <p>이 스티키 요소는 스크롤 시 헤더 아래로 고정됩니다.</p>
  </div>
  <p>내용을 스크롤하면 스티키 요소는 스크롤하면서 헤더 아래에 고정됩니다.</p>
  <p>더 많은 내용을 추가할 수 있습니다.</p>
<br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><
br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><b
r><br><br><br><br><br>
</div>
</body>
</html>.
```



z-index는 div가 겹쳤을때 누구를 위에 놓을 것인지 결정하는 예제이다. position 속성이 기본값 static일때는 사용할 수 없다. 숫자가 클수록 위에 나타난다.

```
<!DOCTYPE HTML>
<html>
<head>
  <style>
    .box {
      width: 200px;              height: 100px;
      position: absolute;        top: 50px;
      color: white;              font-size: 18px;
      text-align: center;        line-height: 100px;
    }
    .box1 {
      left: 50px;       background-color: #3498db; /* 파란색 배경 */
      z-index: 1;
    }
    .box2 {
      left: 100px;       background-color: #e74c3c; /* 빨간색 배경 */
      z-index: 2;
    }
  </style>
</head>
<body>
  <div class="box box1">상자 1</div>
  <div class="box box2">상자 2</div>
</body>
</html>
```

다음과 같은 div모양을 만들어 보자.



> 14. DIV float

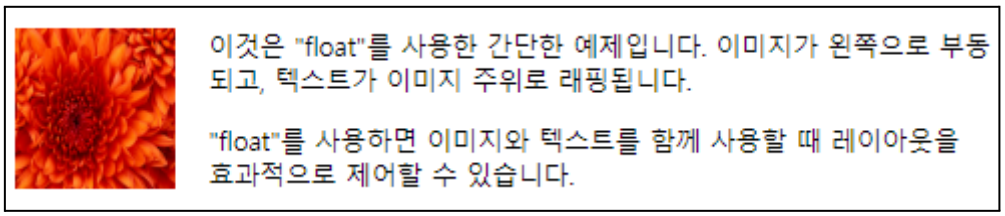
"float" 속성은 웹 페이지에서 블럭 태그로 인해 발생한 빈공간을 다른 블록 요소로 왼쪽 또는 오른쪽으로 이동시켜서 채우게 만드는 CSS 속성입니다. 이것을 사용하여 이미지와 텍스트를 한줄에 함께 표시하거나 여러 div를 한줄에 생성 할 수 있습니다.

float: left; 왼쪽으로 요소를 부동시킵니다. 요소는 페이지에서 왼쪽에 배치됩니다.

float: right; 오른쪽으로 요소를 부동시킵니다. 요소는 페이지에서 오른쪽에 배치됩니다.

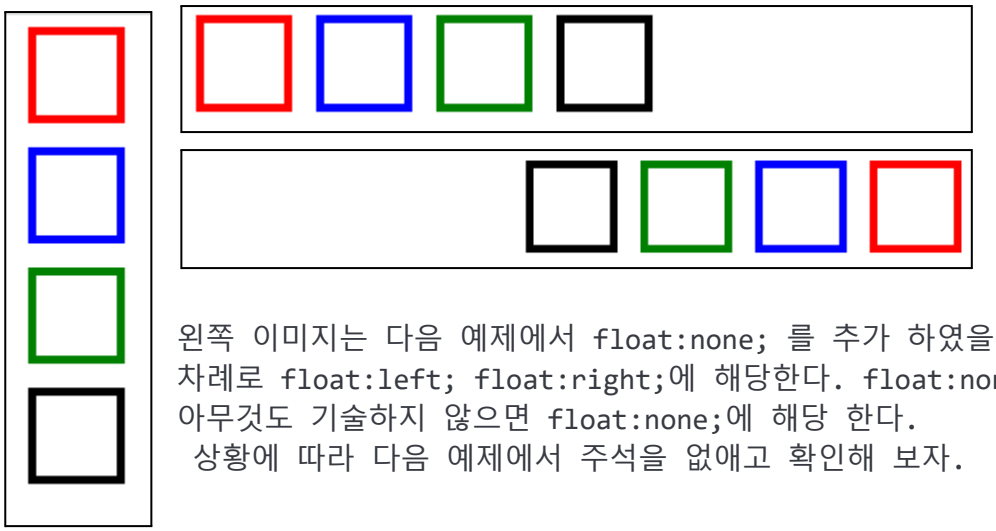
float: none; 부동을 해제한다. 기본 값으로 한줄에 하나의 div를 배치합니다.

다음 예제는 float를 이용해서 한줄에 이미지와 글씨를 배열한 것이다.




```
<!DOCTYPE html>
<html>
<head>
  <style>
    .image-container {
      float: left; /* 이미지를 왼쪽으로 부동시킵니다. */
      margin-right: 20px; /* 오른쪽 여백을 설정합니다. */
    }
  </style>
</head>
<body>
<div class="image-container">
  <img src="1.jpg" alt="이미지" width="100" height="100">
</div>
<div>
  <p>
    이것은 "float"를 사용한 간단한 예제입니다. 이미지가 왼쪽으로 부동되고, 텍스트가 이미지 주위로 래핑됩니다.
  </p>
  <p>
    "float"를 사용하면 이미지와 텍스트를 함께 사용할 때 레이아웃을 효과적으로 제어할 수 있습니다.
```

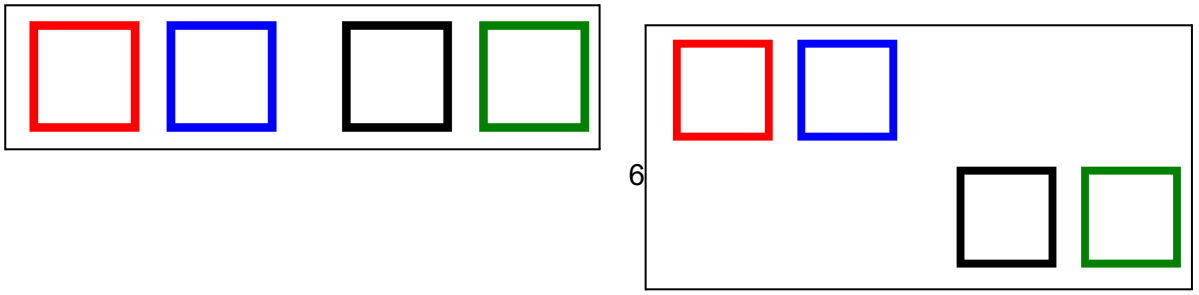
```
  </p>
</div>
</body>
</html>
```



왼쪽 이미지는 다음 예제에서 float:none; 를 추가 하였을 때의 모양이고 차례로 float:left; float:right;에 해당한다. float:none;은 기본값으로 아무것도 기술하지 않으면 float:none;에 해당 한다.
 상황에 따라 다음 예제에서 주석을 없애고 확인해 보자.

```
<html>
<head>
<style>
div{
	width:50px;
	height:50px;
	margin-left:15px;
	margin-bottom:15px;
	/* float:none; 기본값 float속성이 적용되지 않아 한줄에 한개씩 div 정렬*/
	/* float:left; div가 왼쪽으로 정렬 */
	/* float:right; 오른쪽 정렬 */
}
</style>
</head>
<body>
<div style="border:5px solid red" ></div>
<div style="border:5px solid blue" ></div>
<div style="border:5px solid green" ></div>
<div style="border:5px solid black" ></div>
</body>
</html>
```

clear: both;는 이전에 나온 태그의 float속성을 다음 나올 태그에 더 이상 적용하지 않을때 사용한다. 요소들 주변에 영향을 주지 않도록 이후 나오는 태그에 float 속성을 적용 한다.

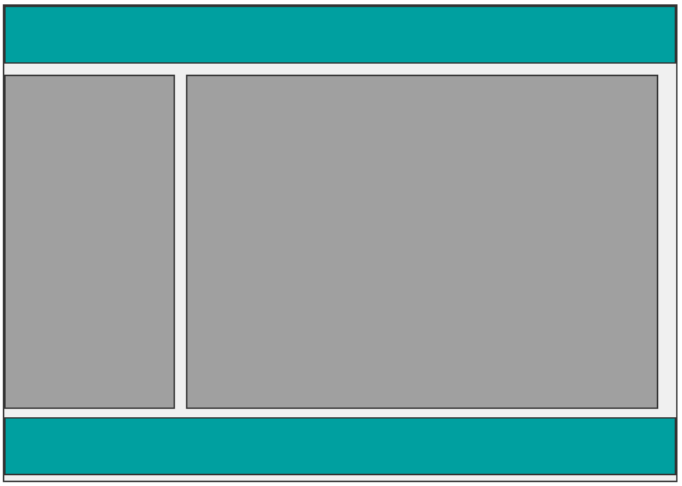


기본 적으로 빨강 파랑 div를 float left로 왼쪽에 정렬하고 검정,녹색 div를 float right정렬하면 한줄에 오른쪽으로 정렬되어 상위 왼쪽 이미지 처럼 정렬된다.

만약 파랑과 검정 사이에 div3 클래스에 clear: both; 속성을 적용한 빈 를 추가한다면 빈 이전에 기술된 빨강 파랑 div의 float left속성은 첫줄 왼쪽에 생성되고 빈 다음에 기술된 div들에게 clear both속성이 적용되어 더이상 float으로 생성된 공간에 들어 갈수 없어 다음 줄에 float right속성을 적용한 녹색, 검정 div는 오른쪽 정렬 하게된다. 결국 오른쪽 이미지 처럼 정렬된다.

```
<html>
<head>
<style>
.div1{
      width:50px;
      height:50px;
      margin-left:15px;
      margin-bottom:15px;
       float:left;
}
.div2{
      width:50px;
      height:50px;
      margin-left:15px;
      margin-bottom:15px;
       float:right;
}
/*
.div3{
       clear:both;
}*/
</style>
</head>
<body>
<div class="div1" style="border:5px solid red" ></div>
<div class="div1" style="border:5px solid blue" ></div>
<div class="div3" ></div>
<div class="div2" style="border:5px solid green" ></div>
<div class="div2" style="border:5px solid black" ></div>
</body>
</html>
```

> 15. div 사용한 화면 구성



웹페이지를 만들때 복잡한 화면을 div로 나눠서 잘 쌓아서 만든다. 왼쪽은 가장 대표적인 화면 분할 형태를 나타내고 있다. 다음 예제는 왼쪽 이미지 같은 화면을 만드는 과정이다. 과정별 코드를 확인해서 만들어보고 이해해 보자.

예제 1: 기본 레이아웃

첫 번째 예제는 가장 기본적인 웹 페이지 레이아웃을 설정합니다.

#wrap라는 ID를 가진 요소를 사용하여 웹 페이지를 감싸는 컨테이너를 생성합니다.

이 컨테이너는 특정 폭(610px), 높이(430px), 여백 및 테두리 스타일을 가지며 밝은 회색 배경색을 가집니다.

```
<!DOCTYPE HTML>
<html>
 <head>
  <title>DIV Layout Coding</title>
  <style>
    #wrap {
      width: 610px;
```

```
      height:430px;
      margin: auto;
      border: 2px solid #333;
      background-color: #f0f0f0; /* 밝은 회색 배경색 */
    }
   </style>
  </head>
  <body>
    <div id="wrap">
    </div>
  </body>
 </html>
```

예제 2: 헤더와 푸터 추가

두 번째 예제는 첫 번째 예제에서 레이아웃을 확장하여 헤더와 푸터를 추가합니다.

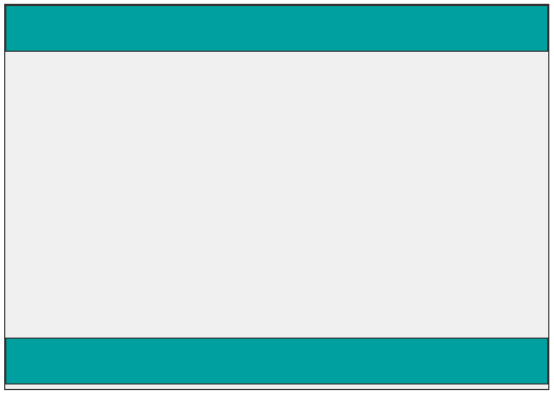
#header와 #footer라는 두 개의 새로운 요소가 #wrap 컨테이너 내에 추가됩니다.

헤더와 푸터는 특정 높이(50px), 청록색 배경, 및 테두리 스타일을 가지며, 푸터는 상단 여백도 가집니다.

```
<!DOCTYPE HTML>
<html>
 <head>
  <title>DIV Layout Coding</title>
  <style>
    #wrap {
      width: 610px;       height:430px;
      margin: auto;       border: 2px solid #333;
      background-color: #f0f0f0; /* 밝은 회색 배경색 */
    }
    #header {
      height: 50px;       background: #00a0a0; /* 밝은 청록색 */
      border: 1px solid #333;
    }
    #footer {
      height: 50px;       margin-top: 10px;
```

```
      background: #00a0a0; /* 밝은 청록색 */
      border: 1px solid #333;
    }

  </style>
 </head>
 <body>
   <div id="wrap">
     <div id="header"></div>
     <div id="footer"></div>
   </div>
 </body>
</html>
```

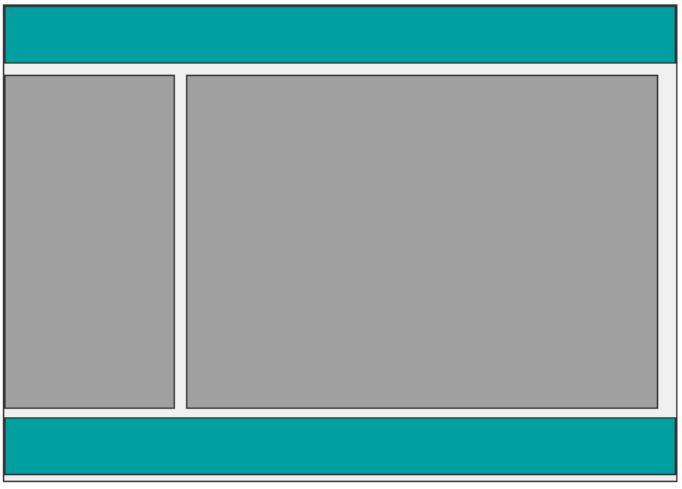
예제3:본문 내용 추가 영역설정



#main라는 요소가 추가되어 본문의 중앙에 위치하며 상단 여백과 높이를 가집니다. 중앙에 2개의 div를 float를 이용해서 넣을 예정이어서 문제가 생기지 않도록 #main div로 감싸는 작업을 하였다.

```
<!DOCTYPE HTML>
<html>
 <head>
  <title>DIV Layout Coding</title>
  <style>
    #wrap {
      width: 610px;
      height:430px;
      margin: auto;
      border: 2px solid #333;
      background-color: #f0f0f0; /* 밝은 회색 배경색 */
    }
    #header {
      height: 50px;
      background: #00a0a0; /* 밝은 청록색 */
      border: 1px solid #333;
    }
    #main {
      margin-top: 10px;
```

```
      height:300px;
    }

    #footer {
      height: 50px;
      margin-top: 10px;
      background: #00a0a0; /* 밝은 청록색 */
      border: 1px solid #333;
    }
    .clear {
      clear: both;
    }
  </style>
 </head>
 <body>
   <div id="wrap">
     <div id="header"></div>
     <div id="main">

     </div>
     <div id="footer"></div>
   </div>
 </body>
</html>
```



예제 4: 최종 완성

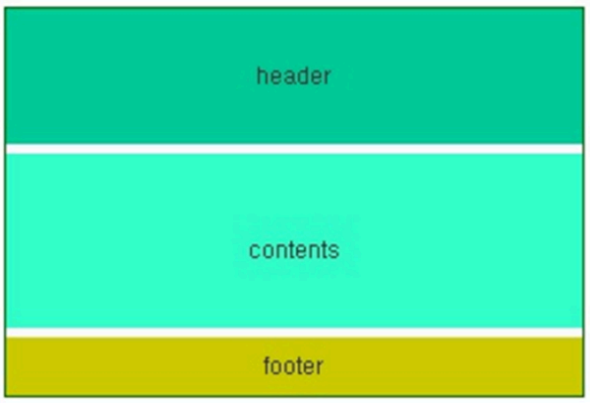
#main안에 다음 작업을 하였다. #left_main과 #right_main이라는 두 부분으로 본문 내용을 나눕니다. 이러한 부분은 왼쪽으로 부유(floating)하여 높이와 백분율로 너비를 가집니다.

부유된 요소를 올바르게 정리하기 위해 .clear 클래스가 사용하였다.

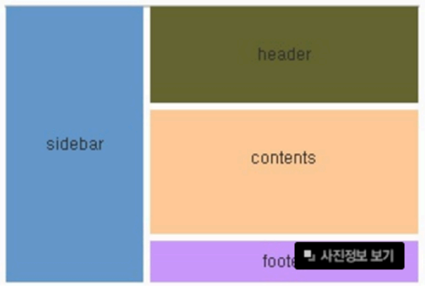
```
<!DOCTYPE HTML>
<html>
 <head>
  <title>DIV Layout Coding</title>
  <style>
    #wrap {
      width: 610px;      height:430px;
```

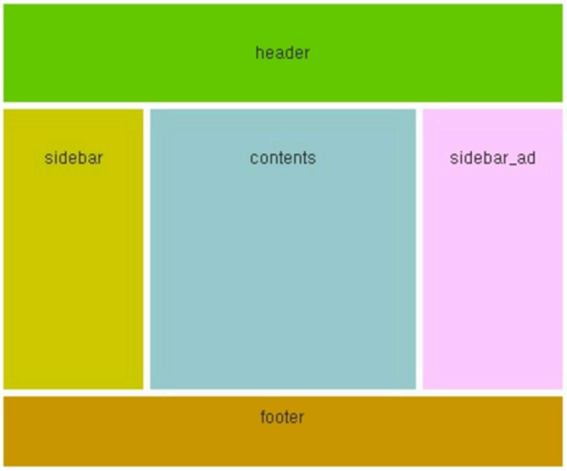
```
      margin: auto;      border: 2px solid #333;
      background-color: #f0f0f0; /* 밝은 회색 배경색 */
    }
    #header {
      height: 50px;      background: #00a0a0; /* 밝은 청록색 */
      border: 1px solid #333;
    }
    #main {
      margin-top: 10px;
    }
    #left_main {
      height: 300px;      width: 25%;
      background: #a0a0a0; /* 밝은 회색 배경색 */
      margin-right: 10px;      float: left;
      border: 1px solid #333;
    }
    #right_main {
      height: 300px;      width: 70%;
      background: #a0a0a0; /* 밝은 회색 배경색 */
      float: left;      border: 1px solid #333;
    }
    #footer {
      height: 50px;      margin-top: 10px;
      background: #00a0a0; /* 밝은 청록색 */
      border: 1px solid #333;
    }
    .clear {
      clear: both;
    }
  </style>
 </head>
 <body>
   <div id="wrap">
     <div id="header"></div>
     <div id="main">
       <div id="left_main"></div>
       <div id="right_main"></div>
       <div class="clear"></div>
     </div>
     <div id="footer"></div>
   </div>
 </body>
</html>
```

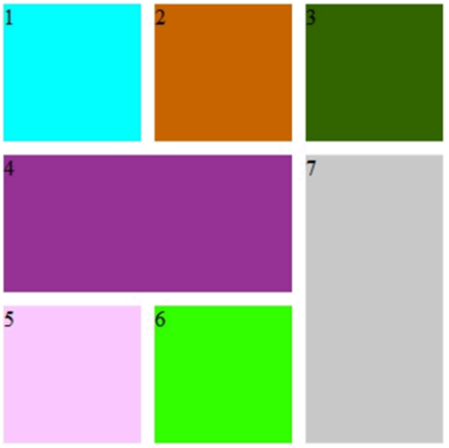
다음 div를 만들어 보자.

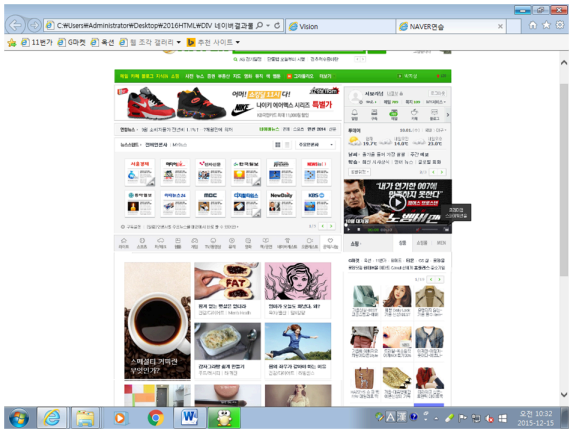

header
contents
footer

레이아웃실습2


sidebar
header
contents
foote
사진정보 보기


header
sidebar
contents
sidebar_ad
footer

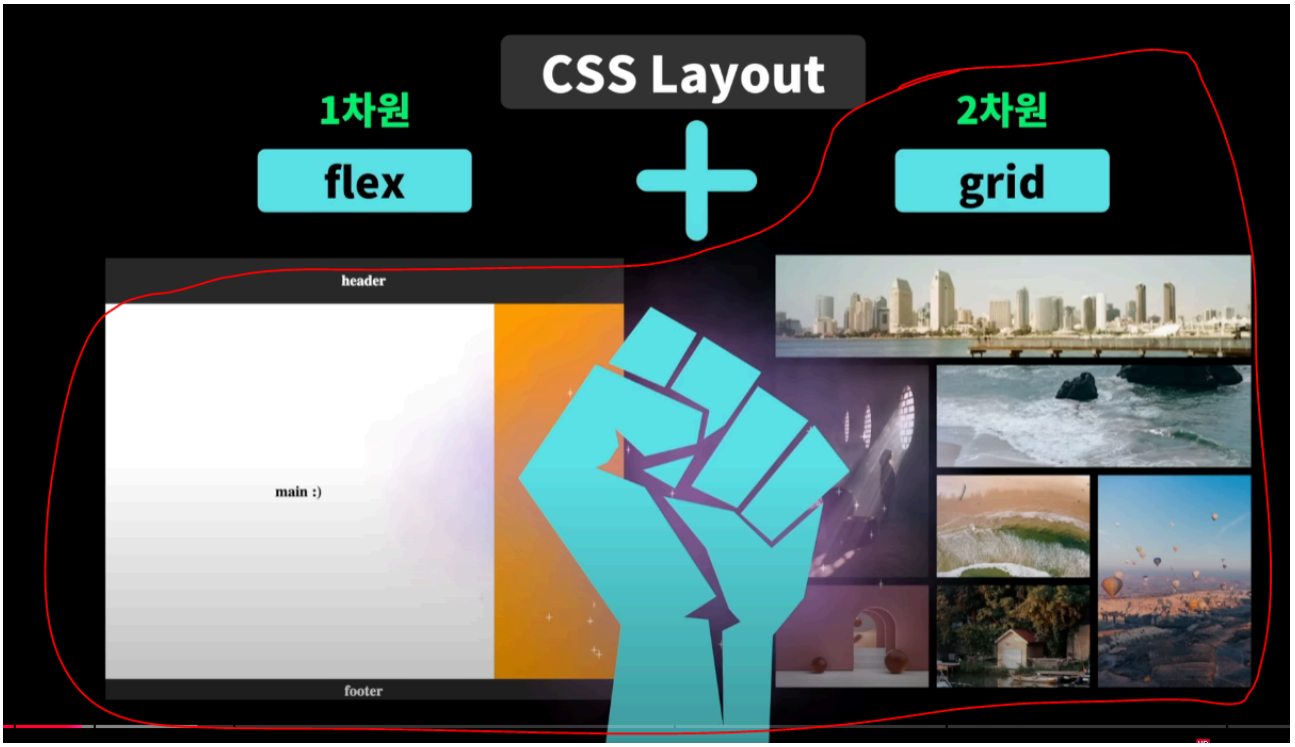

1
2
3
4
7
5
6



ppt에서 메인 사이트 메뉴와 를 만들어 보자.

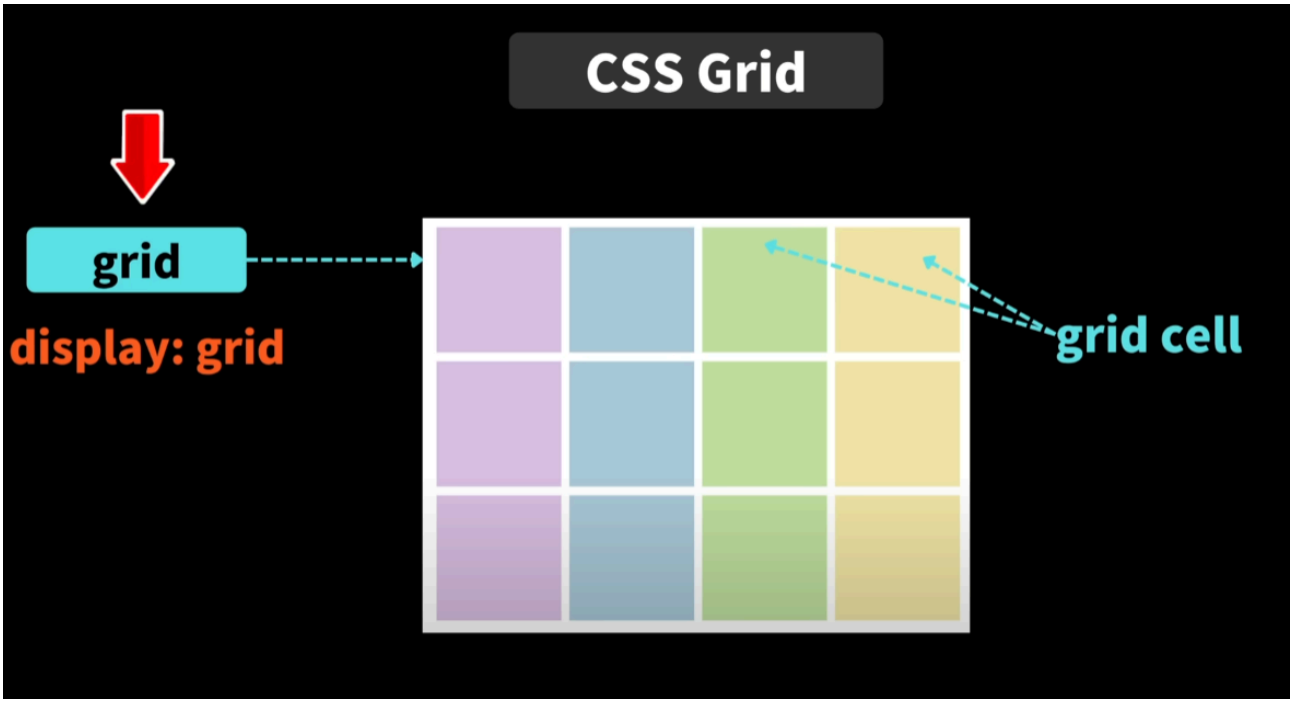
> 16. grid

CSS Grid는 웹 레이아웃을 만들기 위한 강력하고 유연한 레이아웃 시스템입니다. Grid는 2차원(행과 열)을 기반으로 요소를 배치할 수 있어 복잡한 레이아웃도 간단하게 구현할 수 있습니다.



사용방법

특정 div에 display:grid를 추가하면 모든자식이 grid cell로 변환이 된다.



---

1. CSS Grid의 기본 개념

- **컨테이너(container)**: `display: grid`를 적용하는 부모 요소.
- **아이템(item)**: Grid 컨테이너 내부에 있는 직계 자식 요소.
- **행(row)과 열(column)**: Grid는 행과 열로 구성되며, 각 셀에 아이템이 배치됩니다

1	2
3	4

2. CSS Grid 기본 설정

컨테이너에 Grid 설정

Grid를 사용하려면 컨테이너에 display: grid를 설정해야 합니다.

html

코드 복사

```
<!DOCTYPE html>
<html>
<head>



<style>
  .grid-container {
    display: grid; /* Grid 레이아웃 활성화 */
    grid-template-columns: 1fr 1fr; /* 두 개의 열 */
    grid-template-rows: auto auto; /* 두 개의 행 */
    gap: 10px; /* 셀 간격 */
      background-color: dodgerblue;
  }


.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
```

```
}

</style>



</head>
<body>



<div class="grid-container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
</div>
</body>
</html>



grid-template-columns: 1fr 1fr; /* 두 개의 열 */

grid-template-rows: 1fr auto; /* 첫 번째 행은 남는 공간을 차지, 두 번째 행은 내용 크기만큼 */
```

grid와 inline-grid의 차이는 그리드에 내용을 꽉 채우고 싶으면 grid 공관을 만들고 싶으면 inline-grid를 사용하면된다.

2fr의 의미

- fr(fraction, 비율 단위)은 **사용 가능한 공간을 나누는 단위**입니다.
- grid-template-rows: 2fr 1fr; 이라고 하면:
  - 첫 번째 행이 전체 가용 공간의 **2배를 차지**
  - 두 번째 행이 **1배를 차지**

`grid`와 `inline-grid`는 **CSS Grid Layout**을 기반으로 한 두 가지 레이아웃 방식으로, 핵심 차이점은 **컨테이너의 블록 성격**입니다.

차이점 요약

속성	`grid`	`inline-grid`
동작 방식	블록 요소처럼 동작	인라인 요소처럼 동작
기본 크기	부모 컨테이너의 너비를 채움	내부 콘텐츠 크기에 따라 크기 결정
다른 요소와의 배치	다른 블록 요소와 줄 바꿈	다른 인라인 요소와 한 줄에 배치
사용 예	페이지의 주요 레이아웃 구조	텍스트, 버튼 등과 함께 사용

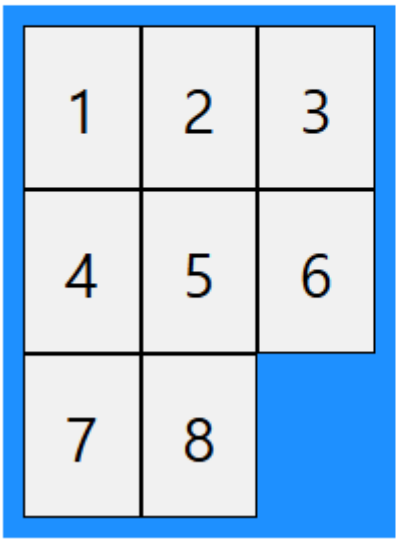
display: grid

Use display: grid; to make a block-level grid container:

1	2	3
4	5	6
7	8	

display: inline-grid

Use display: inline-grid; to make an inline grid container:



grid는 화면 전체를 차지하고 grid-inline은 내용 만큼만 차지 한다.

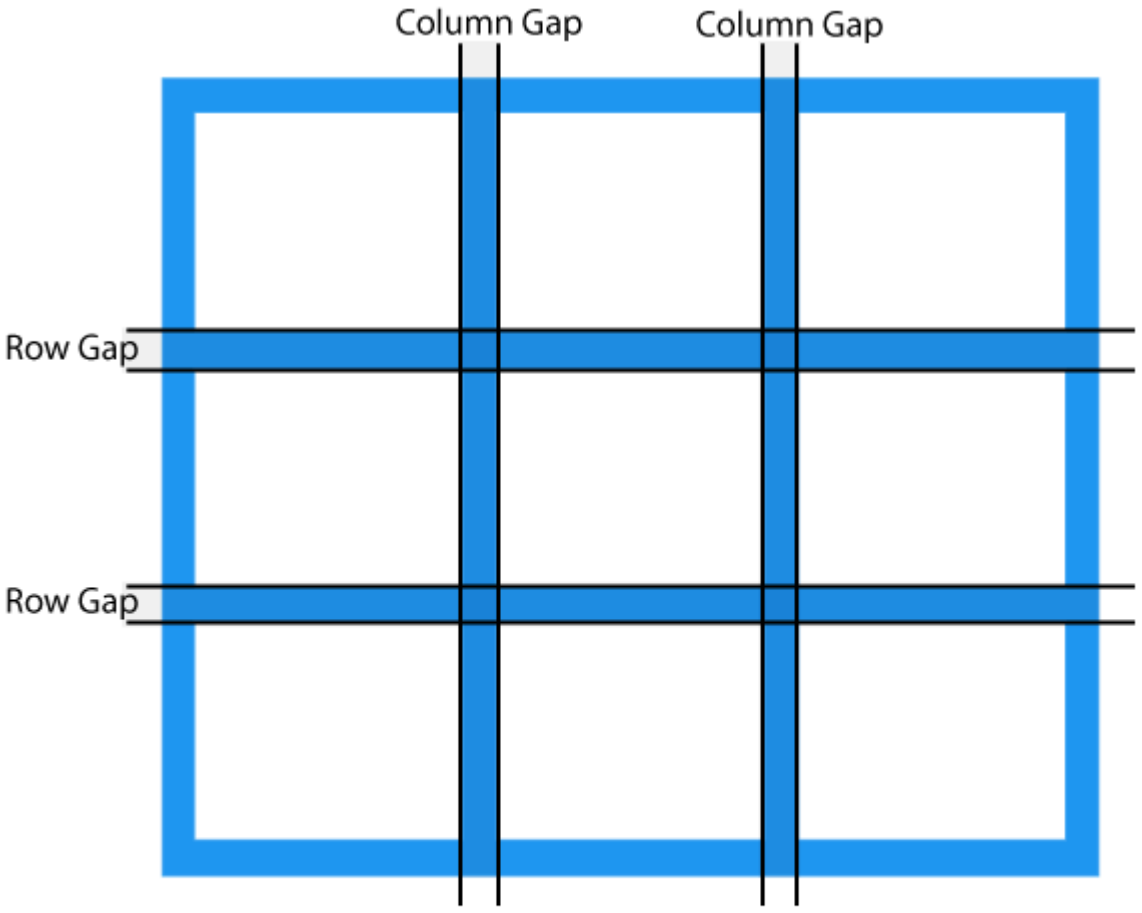
```
<!DOCTYPE html>
<html>
<head>
<style>
.container {
  display: inline-grid;/*  display: grid; 모양이 달라진다.*/
  grid-template-columns: auto auto auto;
  background-color: dodgerblue;
  padding: 10px;
}
```

```
.container > div {
  background-color: #f1f1f1;
  border: 1px solid black;
  padding: 20px;
  font-size: 30px;
  text-align: center;
}
</style>
</head>
<body>

<h1>display: inline-grid</h1>

<p>Use display: inline-grid; to make an inline grid container:</p>

<div class="container">
  <div>1</div>
  <div>2</div>
  <div>3</div>
  <div>4</div>
  <div>5</div>
  <div>6</div>
  <div>7</div>
```

```
  <div>8</div>
</div>
</body>
</html>
```

3.2. 간격 설정

- `gap`: 행과 열 간격을 동시에 설정.
- `row-gap` 및 `column-gap`: 각각 행과 열의 간격 설정.



You can adjust the gap size by using one of the following properties:

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

```css
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 20px; /* 모든 간격 20px */
}
.container {
  display: grid;
  column-gap: 50px; /*컬럼 간격 50px*/
}
```

Grid Lines

1		2
3	4	5
6	7	8

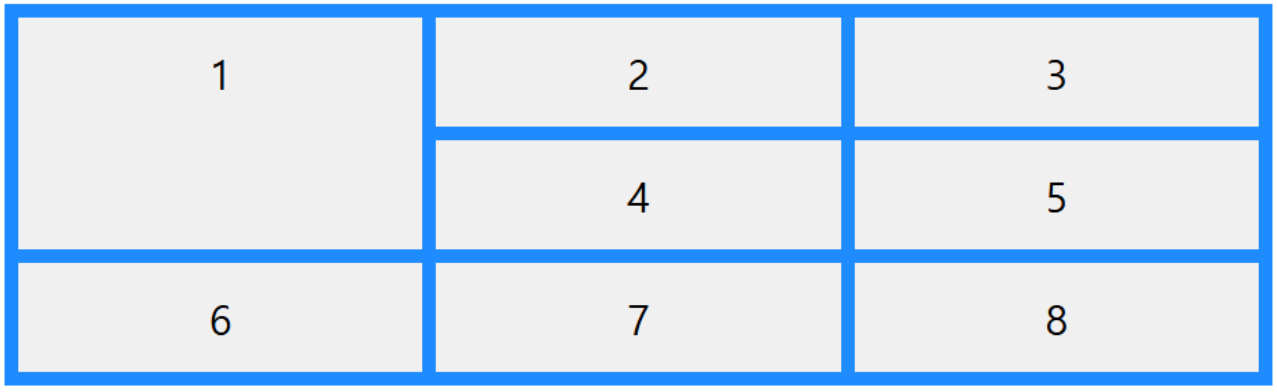
You can refer to line numbers when placing grid items.

```
<!DOCTYPE html>
<html>
<head>
<style>
```

```
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}

.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}

.item1 {
  grid-column-start: 1;
  grid-column-end: 3;
}
</style>
</head>
<body>

<h1>Grid Lines</h1>

<div class="grid-container">
  <div class="item1">1</div>
```

```
  <div class="item2">2</div>

  <div class="item3">3</div>

  <div class="item4">4</div>

  <div class="item5">5</div>

  <div class="item6">6</div>

  <div class="item7">7</div>

  <div class="item8">8</div>

</div>


<p>You can refer to line numbers when placing grid items.</p>


</body>

</html>
```

Grid Lines



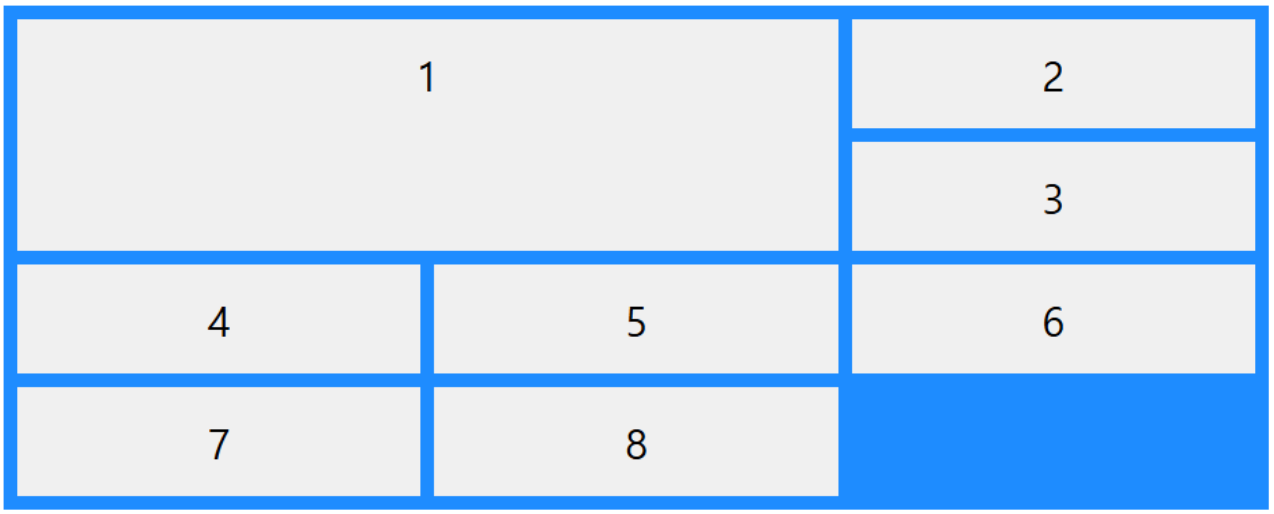
You can refer to line numbers when placing grid items.

```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: auto auto auto;
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}
```

```
.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}

.item1 {
  grid-row-start: 1;
  grid-row-end: 3;
}
</style>
</head>
<body>

<h1>Grid Lines</h1>

<div class="grid-container">
  <div class="item1">1</div>
  <div class="item2">2</div>
  <div class="item3">3</div>
  <div class="item4">4</div>
  <div class="item5">5</div>
  <div class="item6">6</div>
  <div class="item7">7</div>
  <div class="item8">8</div>
</div>

<p>You can refer to line numbers when placing grid items.</p>

</body>
</html>
```

합치면 다음과 같은 모양을 만들수 있다.

Grid Lines



You can refer to line numbers when placing grid items.

CSS Grid에서 아이템 배치를 위한 속성인 grid-column과 grid-row를 사용하여 특정 위치에 아이템을 배치하는 방법을 자세히 살펴보겠습니다. 이를 이해하기 위해 전체 HTML과 CSS 코드를 구성하고 설명을 추가합니다.

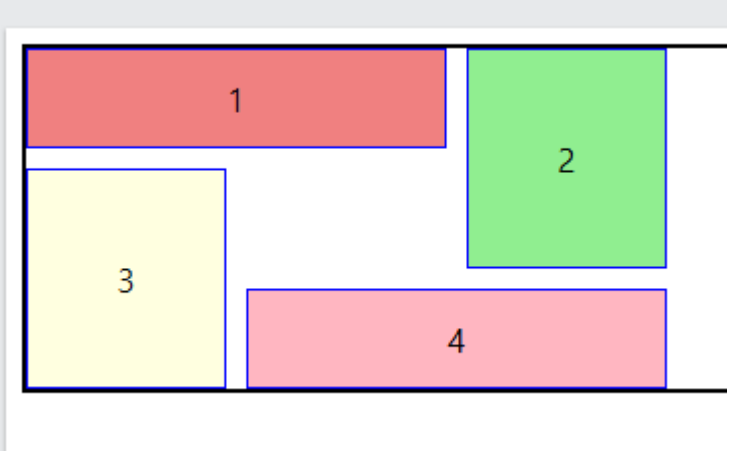
전체 코드 예제

HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>CSS Grid Example</title>
  <style>
    .grid-container {
      display: grid; /* Grid 레이아웃 활성화 */
      grid-template-columns: 100px 100px 100px; /* 3개의 열, 각각 100px */
      grid-template-rows: 50px 50px 50px; /* 3개의 행, 각각 50px */
      gap: 10px; /* 셀 간격 */
      border: 2px solid black;
    }
    .grid-container div {
      background-color: lightblue;
```

```
  border: 1px solid blue;
  display: flex;
  align-items: center;
  justify-content: center;
}
.grid-container div:nth-child(1) {
  grid-column: 1 / 3; /* 1번째 열부터 2번째 열까지 차지 */
  grid-row: 1 / 2;    /* 1번째 행만 차지 */
  background-color: lightcoral;
}
.grid-container div:nth-child(2) {
  grid-column: 3 / 4; /* 3번째 열만 차지 */
  grid-row: 1 / 3;    /* 1번째 행부터 2번째 행까지 차지 */
  background-color: lightgreen;
}
.grid-container div:nth-child(3) {
  grid-column: 1 / 2; /* 1번째 열만 차지 */
  grid-row: 2 / 4;    /* 2번째 행부터 3번째 행까지 차지 */
  background-color: lightyellow;
}
.grid-container div:nth-child(4) {
  grid-column: 2 / 4; /* 2번째 열부터 3번째 열까지 차지 */
  grid-row: 3 / 4;    /* 3번째 행만 차지 */
  background-color: lightpink;
```

```
    }
  </style>
</head>
<body>
  <div class="grid-container">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
  </div>
</body>
</html>
```



---

(1) 첫 번째 아이템

css

코드 복사

```
.grid-container div:nth-child(1) {
  grid-column: 1 / 3; /* 1번째 열부터 2번째 열까지 차지 */
  grid-row: 1 / 2;    /* 1번째 행만 차지 */
  background-color: lightcoral;
}
```

- **grid-column: 1 / 3**: 첫 번째 열부터 세 번째 열 직전(두 번째 열)까지 차지.
- **grid-row: 1 / 2**: 첫 번째 행만 차지.
- 배경색을 lightcoral로 설정.

---

(2) 두 번째 아이템

css

코드 복사

```
.grid-container div:nth-child(2) {
  grid-column: 3 / 4; /* 3번째 열만 차지 */
  grid-row: 1 / 3;    /* 1번째 행부터 2번째 행까지 차지 */
  background-color: lightgreen;
}
```

- **grid-column: 3 / 4**: 세 번째 열만 차지.
- **grid-row: 1 / 3**: 첫 번째 행부터 세 번째 행 직전(두 번째 행)까지 차지.
- 배경색을 lightgreen으로 설정.

---

(3) 세 번째 아이템

css

코드 복사

```
.grid-container div:nth-child(3) {
  grid-column: 1 / 2; /* 1번째 열만 차지 */
  grid-row: 2 / 4;    /* 2번째 행부터 3번째 행까지 차지 */
  background-color: lightyellow;
}
```

- **grid-column: 1 / 2**: 첫 번째 열만 차지.
- **grid-row: 2 / 4**: 두 번째 행부터 네 번째 행 직전(세 번째 행)까지 차지.
- 배경색을 lightyellow로 설정.

---

(4) 네 번째 아이템

css

코드 복사

```
.grid-container div:nth-child(4) {
  grid-column: 2 / 4; /* 2번째 열부터 3번째 열까지 차지 */
  grid-row: 3 / 4;    /* 3번째 행만 차지 */
  background-color: lightpink;
}
```

- **grid-column: 2 / 4**: 두 번째 열부터 네 번째 열 직전(세 번째 열)까지 차지.
- **grid-row: 3 / 4**: 세 번째 행만 차지.
- 배경색을 lightpink로 설정.

결과

1. 아이템들이 지정된 행과 열에 맞게 배치됩니다.
2. Grid의 **셀 간격, 크기** 및 **배경색**으로 구분하기 쉬운 레이아웃이 생성됩니다.

> 16. flex

1. CSS Flexbox란?

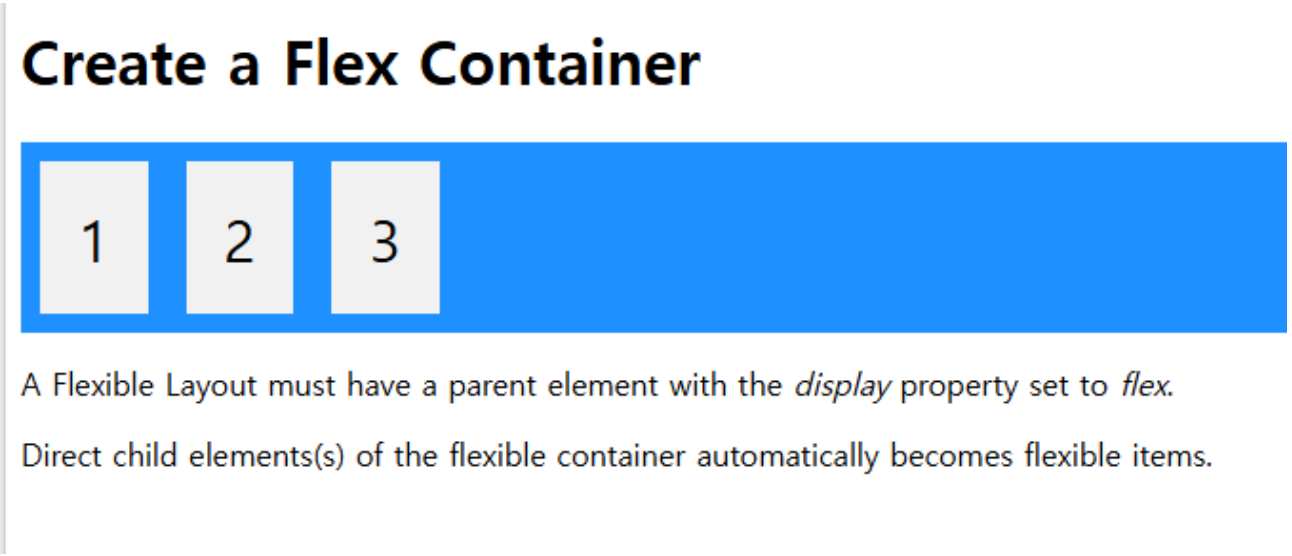
CSS Flexbox (Flexible Box Layout)는 요소들을 행(row) 또는 열(column) 방향으로 배치하는 CSS의 레이아웃 시스템입니다. Flexbox는 요소들이 크기가 동적으로 변하더라도 유연하게 배치되도록 도와줍니다. 이를 통해 **float**나 **positioning**과 같은 오래된 방법을 사용할 필요 없이 반응형 레이아웃을 쉽게 설계할 수 있습니다.

- Flexbox는 레이아웃을 설계할 때 부모 컨테이너의 크기에 맞춰 자식 요소들을 자동으로 정렬하고 배치합니다.

2. CSS Flexbox 구성 요소

Flexbox는 두 가지 주요 요소로 구성됩니다:

- **Flex Container (플렉스 컨테이너)**: Flexbox 레이아웃을 적용할 부모 요소입니다. 이 부모 요소에 `display: flex;`를 설정하면 그 안의 자식 요소들이 Flexbox 레이아웃에 따라 배치됩니다.
- **Flex Items (플렉스 아이템)**: Flex 컨테이너 내부의 자식 요소들입니다. Flex 컨테이너 안에 있으면 자동으로 Flex 아이템으로 취급됩니다.



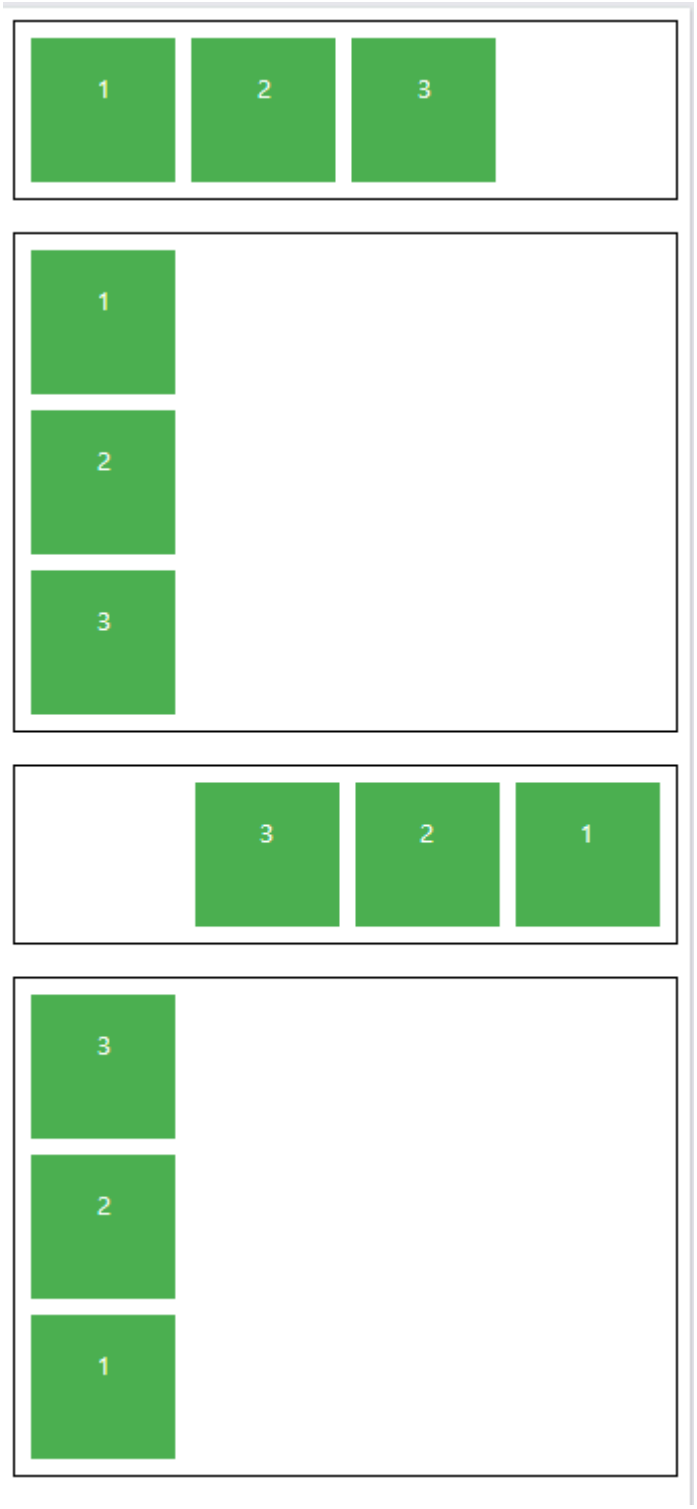

```
<!DOCTYPE html>

<html>
```

```
<head>
<style>
.flex-container {
  display: flex;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  margin: 10px;
  padding: 20px;
  font-size: 30px;
}
</style>
</head>
<body>

<h1>Create a Flex Container</h1>

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

```
<p>A Flexible Layout must have a parent element with the <em>display</em>
property set to <em>flex</em>.</p>


<p>Direct child elements(s) of the flexible container automatically becomes
flexible items.</p>


</body>

</html>
```

1. **flex-direction: row**: 기본값으로, 플렉스 아이템들이 **수평**(가로)로 배치됩니다. 첫 번째 `<div>`가 왼쪽에, 두 번째가 중간, 세 번째가 오른쪽에 배치됩니다.
2. **flex-direction: column**: 아이템들이 **수직**(세로)로 배치됩니다. 첫 번째 `<div>`가 상단, 두 번째가 중간, 세 번째가 하단에 배치됩니다.
3. **flex-direction: row-reverse**: 아이템들이 **수평**으로 배치되지만, 순서가 **반대로** 배치됩니다. 첫 번째 `<div>`가 오른쪽에, 두 번째가 중간, 세 번째가 왼쪽에 배치됩니다.
4. **flex-direction: column-reverse**: 아이템들이 **수직**으로 배치되지만, 순서가 **반대로** 배치됩니다. 첫 번째 `<div>`가 하단, 두 번째가 중간, 세 번째가 상단에 배치됩니다.

이 예제를 실행하면 각 `flex-direction` 값에 따라 요소들이 배치되는 방식이 다르게 나타납니다.




```
<!DOCTYPE html>

<html lang="en">
```

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flex Direction Examples</title>
  <style>
    .flex-container {
      display: flex;
      margin-bottom: 20px;
      border: 2px solid #000;
      padding: 10px;
      gap: 10px;
    }
    .flex-container div {
      background-color: #4CAF50;
      color: white;
      padding: 20px;
      text-align: center;
      width: 50px;
      height: 50px;
    }

    .row {
      flex-direction: row; /* 기본값: 왼쪽에서 오른쪽으로 */
    }
```

```
    .column {
      flex-direction: column; /* 위에서 아래로 */
    }

    .row-reverse {
      flex-direction: row-reverse; /* 오른쪽에서 왼쪽으로 */
    }

    .column-reverse {
      flex-direction: column-reverse; /* 아래에서 위로 */
    }
  </style>
</head>
<body>

  <div class="flex-container row">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container column">
    <div>1</div>
```

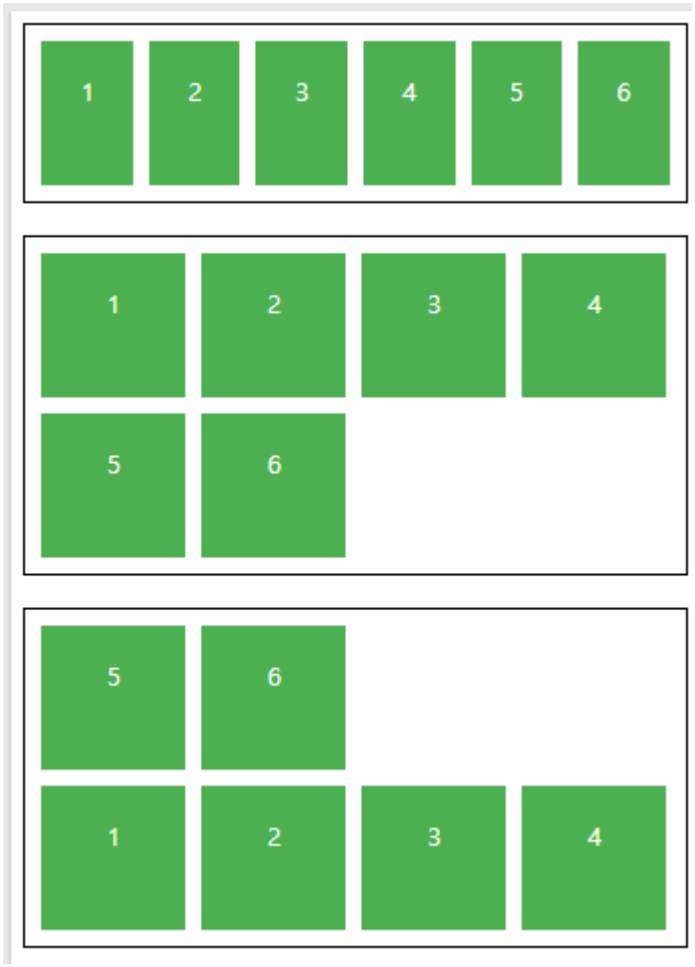
```
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container row-reverse">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container column-reverse">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

</body>
</html>
```

1. **lex-wrap: nowrap;**: 모든 아이템이 한 줄에 배치됩니다. 컨테이너의 너비를 초과할 경우, 아이템들이 축소되어 한 줄로 배치됩니다.
2. **flex-wrap: wrap;**: 아이템들이 한 줄에 다 들어가지 않으면, 두 번째 줄로 넘어가서 배치됩니다. 남은 공간에 맞게 아이템들이 여러 줄로 배치됩니다.
3. **flex-wrap: wrap-reverse;**: wrap과 비슷하지만, 아이템들이 반대 순서로 배치됩니다. 첫 번째 줄은 아래쪽에, 두 번째 줄은 그 위에 배치됩니다.

각 속성의 차이:

- `nowrap`: 모든 아이템이 한 줄에 배치됩니다.
- `wrap`: 공간을 넘어갈 경우 여러 줄로 아이템들이 배치됩니다.
- `wrap-reverse`: `wrap`과 같지만, 줄의 순서가 반대로 배치됩니다.

이 예제 코드를 실행하면 각 `flex-wrap` 속성에 따라 아이템들이 어떻게 배치되는지를 시각적으로 확인할 수 있습니다.




```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Flex Wrap Examples</title>
```

```
<style>
  .flex-container {
    display: flex;
    gap: 10px;
    margin-bottom: 20px;
    border: 2px solid #000;
    padding: 10px;
  }

  .flex-container div {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
    text-align: center;
    width: 50px;
    height: 50px;
  }

  .nowrap {
    flex-wrap: nowrap; /* 아이템들이 한 줄에 배치 */
  }

  .wrap {
    flex-wrap: wrap; /* 아이템들이 여러 줄로 배치 */
```

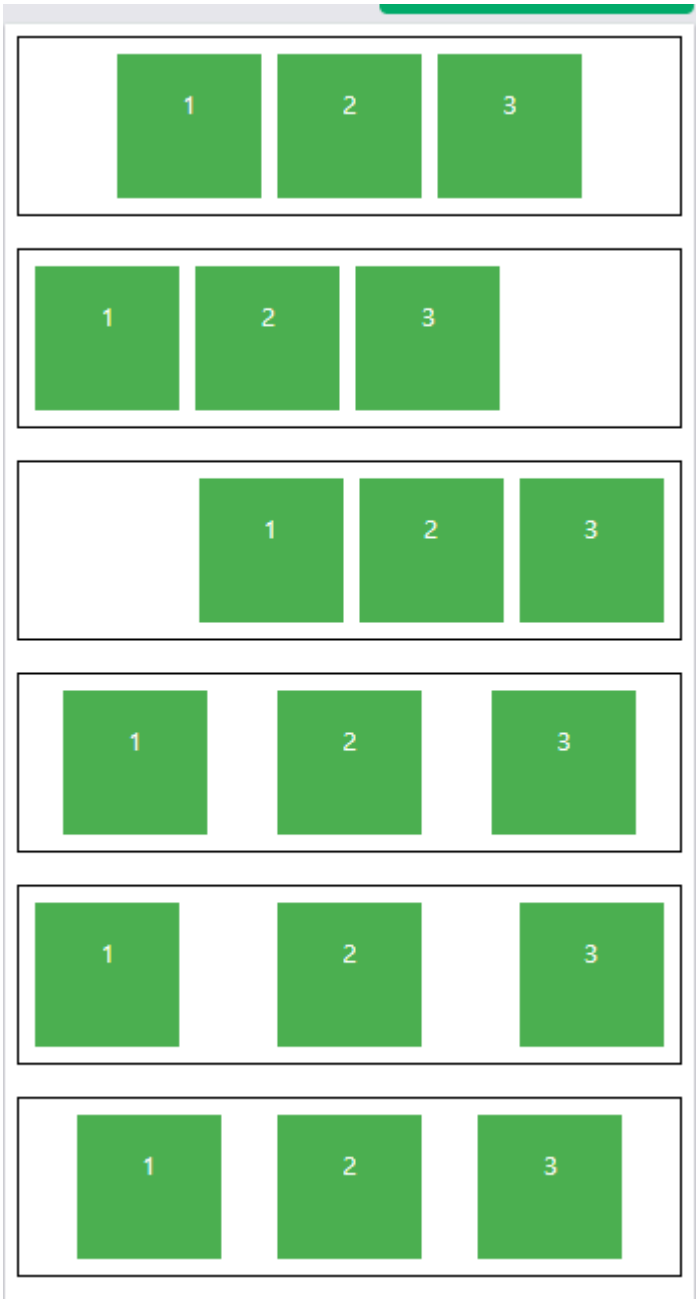
```
    }

    .wrap-reverse {
      flex-wrap: wrap-reverse; /* 아이템들이 반대 순서로 여러 줄로 배치 */
    }
  </style>
</head>
<body>

  <div class="flex-container nowrap">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>

  <div class="flex-container wrap">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
```

```
    <div>6</div>
  </div>

  <div class="flex-container wrap-reverse">
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <div>4</div>
    <div>5</div>
    <div>6</div>
  </div>

</body>
</html>
```

각 속성에 대한 설명:

1. **`justify-content: center;`**
    - 아이템들이 컨테이너의 수평 중앙에 정렬됩니다.
2. **`justify-content: flex-start;`**
    - 아이템들이 컨테이너의 시작점(왼쪽)으로 정렬됩니다.
3. **`justify-content: flex-end;`**
    - 아이템들이 컨테이너의 끝점(오른쪽)으로 정렬됩니다.
4. **`justify-content: space-around;`**
    - 아이템들 사이에 균등한 간격을 배분하되, 양 끝에도 동일한 간격을 제공합니다. 따라서 양 끝에도 여백이 있습니다.
5. **`justify-content: space-between;`**
    - 첫 번째 아이템은 왼쪽 끝에, 마지막 아이템은 오른쪽 끝에 배치되고, 나머지 아이템들은 그 사이에 균등한 간격을 두고 배치됩니다.
6. **`justify-content: space-evenly;`**
    - 아이템들 사이에 동일한 간격을 배분합니다. 양 끝도 포함하여 모든 간격이 동일합니다.

이 코드를 실행하면 각 `justify-content` 속성이 어떻게 작용하는지를 시각적으로 확인할 수 있습니다.




```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>Flex Justify-Content Examples</title>
<style>
  .flex-container {
    display: flex;
    gap: 10px;
    margin-bottom: 20px;
    border: 2px solid #000;
    padding: 10px;
  }

  .flex-container div {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
    text-align: center;
    width: 50px;
    height: 50px;
  }

  .center {
    justify-content: center; /* 중앙 정렬 */
  }

  .flex-start {
```

```
      justify-content: flex-start; /* 시작점(왼쪽) 정렬 */
    }

    .flex-end {
      justify-content: flex-end; /* 끝점(오른쪽) 정렬 */
    }

    .space-around {
      justify-content: space-around; /* 아이템 사이에 균등하게 공간 배분 */
    }

    .space-between {
      justify-content: space-between; /* 양 끝에 배치하고, 나머지 사이에 균등한
간격 */
    }

    .space-evenly {
      justify-content: space-evenly; /* 아이템 사이에 동일한 간격 배분 */
    }
  </style>
</head>
<body>

  <div class="flex-container center">
    <div>1</div>
```

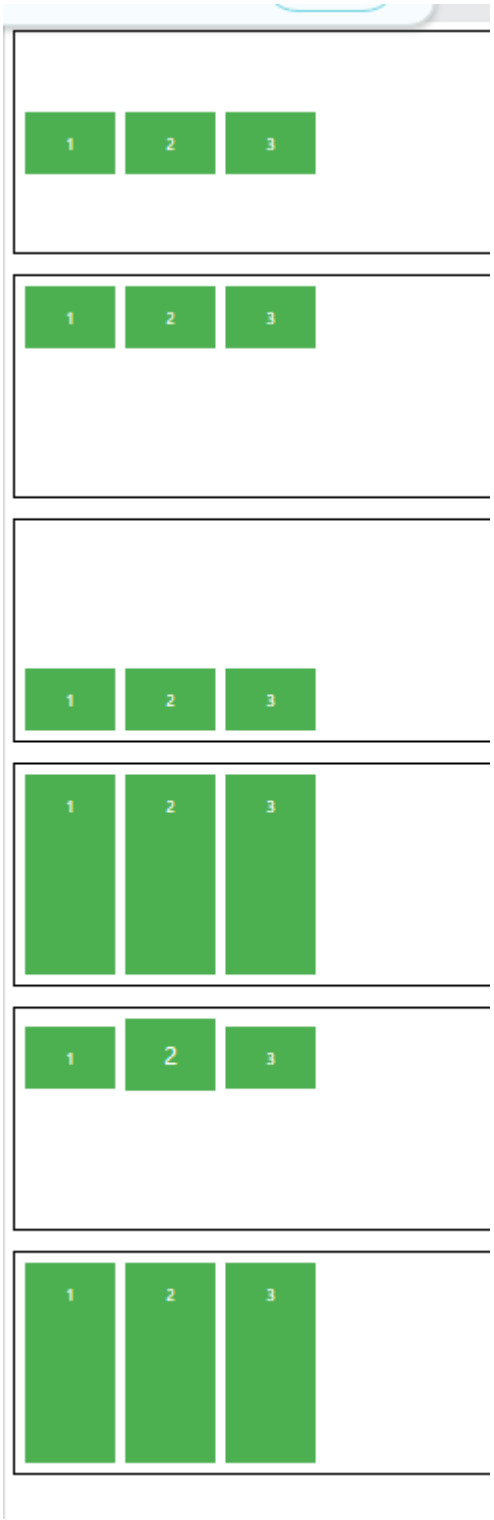
```
  <div>2</div>
  <div>3</div>
</div>

<div class="flex-container flex-start">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<div class="flex-container flex-end">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<div class="flex-container space-around">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<div class="flex-container space-between">
  <div>1</div>
```

```
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container space-evenly">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

</body>
</html>
```

다음은 `align-items` 속성의 다양한 값들 (`center`, `flex-start`, `flex-end`, `stretch`, `baseline`, `normal`)을 사용하여 플렉스 컨테이너 내 아이템들이 어떻게 수직 방향으로 정렬되는지 확인할 수 있는 예제입니다.

설명:

- **`center`**: 아이템들이 컨테이너의 수직 중앙에 정렬됩니다.
- **`flex-start`**: 아이템들이 컨테이너의 상단에 정렬됩니다.
- **`flex-end`**: 아이템들이 컨테이너의 하단에 정렬됩니다.
- **`stretch`**: 아이템들이 컨테이너의 높이에 맞춰 늘어나도록 합니다. 기본값입니다.
- **`baseline`**: 아이템들의 첫 번째 텍스트 라인이 기준이 되어 정렬됩니다.
- **`normal`**: 기본값이며, `stretch`와 비슷하게 동작합니다.


1
2
3
1
2
3
1
2
3
1
2
3
1
2
3
1
2
3

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flex Align-Items Examples</title>
  <style>
    .flex-container {
      display: flex;
      height: 200px;
      margin-bottom: 20px;
      border: 2px solid #000;
      padding: 10px;
      gap: 10px;
    }

    .flex-container div {
      background-color: #4CAF50;
      color: white;
      padding: 20px;
      text-align: center;
      width: 50px;
    }
```

```
.center {
  align-items: center; /* 수직 중앙 정렬 */
}

.flex-start {
  align-items: flex-start; /* 상단 정렬 */
}

.flex-end {
  align-items: flex-end; /* 하단 정렬 */
}

.stretch {
  align-items: stretch; /* 아이템을 컨테이너의 높이에 맞게 늘림 */
}

.baseline {
  align-items: baseline; /* 텍스트 라인 기준으로 정렬 */
}

.normal {
  align-items: normal; /* 기본값, stretch와 비슷하게 동작 */
}
```

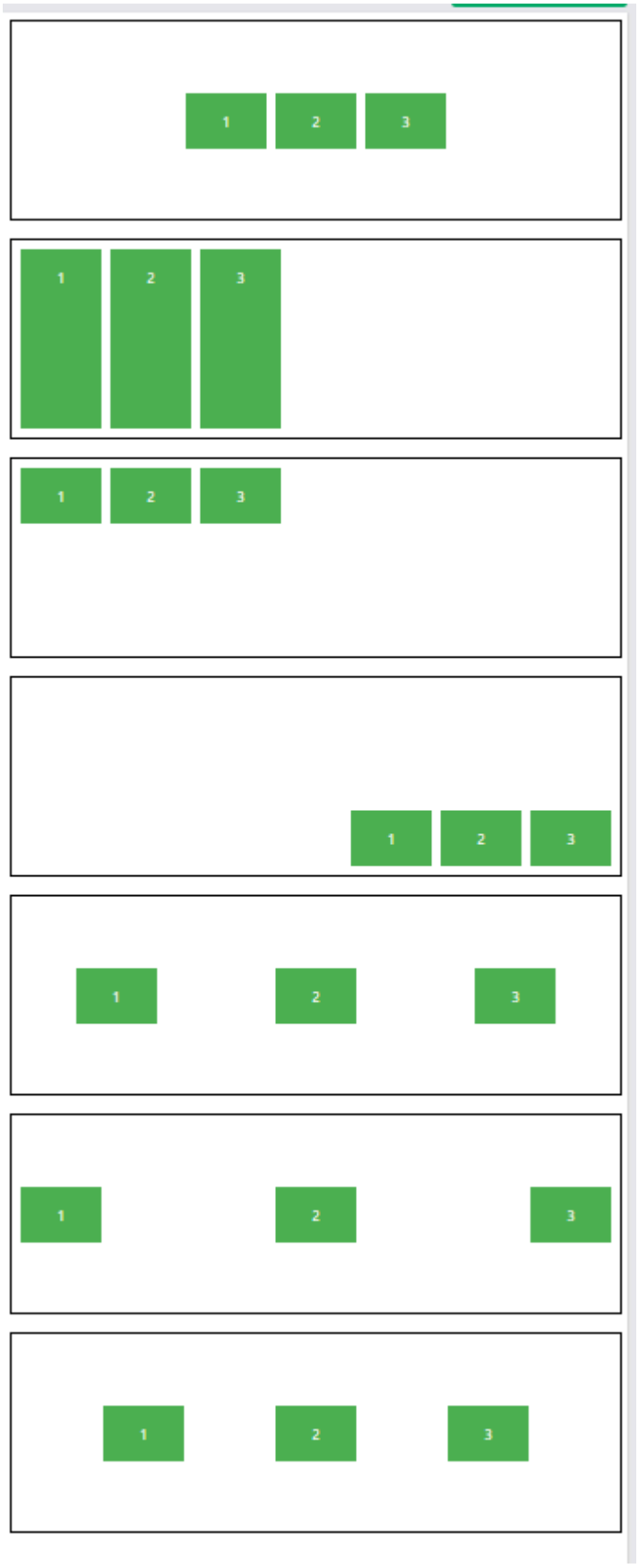
```
  </style>
</head>
<body>

  <div class="flex-container center">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container flex-start">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container flex-end">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container stretch">
    <div>1</div>
```

```
    <div>2</div>
    <div>3</div>
  </div>

  <div class="flex-container baseline">
    <div>1</div>
    <div style="font-size: 24px;">2</div>
    <div>3</div>
  </div>

  <div class="flex-container normal">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

</body>
</html>
```

다음은 `justify-content`와 `align-items` 속성의 다양한 값을 사용하여 플렉스 컨테이너 내 아이템들의 정렬을 확인할 수 있는 예제입니다. 각 속성은 수평 정렬 (`justify-content`)과 수직 정렬 (`align-items`)을 정의합니다.

설명:

- **`center`**: 아이템들을 중앙에 정렬합니다.
- **`stretch`**: 아이템들이 컨테이너의 크기에 맞춰 늘어납니다. (기본값)
- **`flex-start`**: 아이템들이 시작 부분에 정렬됩니다.
- **`flex-end`**: 아이템들이 끝 부분에 정렬됩니다.
- **`space-around`**: 아이템들 사이에 동일한 간격을 배치합니다.
- **`space-between`**: 아이템들 사이에 동일한 간격을 배치하고, 첫 번째 아이템과 마지막 아이템은 컨테이너의 양 끝에 배치됩니다.
- **`space-evenly`**: 아이템들 사이에 균등한 간격을 배치합니다.




```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Alignment Examples</title>
  <style>
    .flex-container {
      display: flex;
      height: 200px;
      margin-bottom: 20px;
      border: 2px solid #000;
      padding: 10px;
      gap: 10px;
    }

    .flex-container div {
      background-color: #4CAF50;
      color: white;
      padding: 20px;
      text-align: center;
      width: 50px;
    }

    .center {
```

```
  justify-content: center; /* 수평 중앙 정렬 */
  align-items: center;     /* 수직 중앙 정렬 */
}

.stretch {
  justify-content: flex-start; /* 수평 시작 정렬 */
  align-items: stretch;       /* 수직 늘리기 */
}

.flex-start {
  justify-content: flex-start; /* 수평 시작 정렬 */
  align-items: flex-start;     /* 수직 시작 정렬 */
}

.flex-end {
  justify-content: flex-end;   /* 수평 끝 정렬 */
  align-items: flex-end;       /* 수직 끝 정렬 */
}

.space-around {
  justify-content: space-around; /* 아이템들 사이에 동일한 간격 */
  align-items: center;           /* 수직 중앙 정렬 */
}
```

```
    .space-between {
      justify-content: space-between; /* 아이템들 사이에 균등한 간격 (양 끝 아이템은 컨테이너 끝에 정렬) */
      align-items: center;           /* 수직 중앙 정렬 */
    }

    .space-evenly {
      justify-content: space-evenly;  /* 아이템들 사이에 균등한 간격 */
      align-items: center;           /* 수직 중앙 정렬 */
    }
  </style>
</head>
<body>

  <!-- Centered Items -->
  <div class="flex-container center">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <!-- Stretch Items -->
  <div class="flex-container stretch">
    <div>1</div>
    <div>2</div>
```

```
  <div>3</div>
</div>

<!-- Flex-start Items -->
<div class="flex-container flex-start">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<!-- Flex-end Items -->
<div class="flex-container flex-end">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>

<!-- Space-around Items -->
<div class="flex-container space-around">
  <div>1</div>
  <div>2</div>
  <div>3</div>
</div>
```

```
  <!-- Space-between Items -->
  <div class="flex-container space-between">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

  <!-- Space-evenly Items -->
  <div class="flex-container space-evenly">
    <div>1</div>
    <div>2</div>
    <div>3</div>
  </div>

</body>
</html>
```

Perfect Centering

```
<!DOCTYPE html>
<html>
<head>
<style>
```

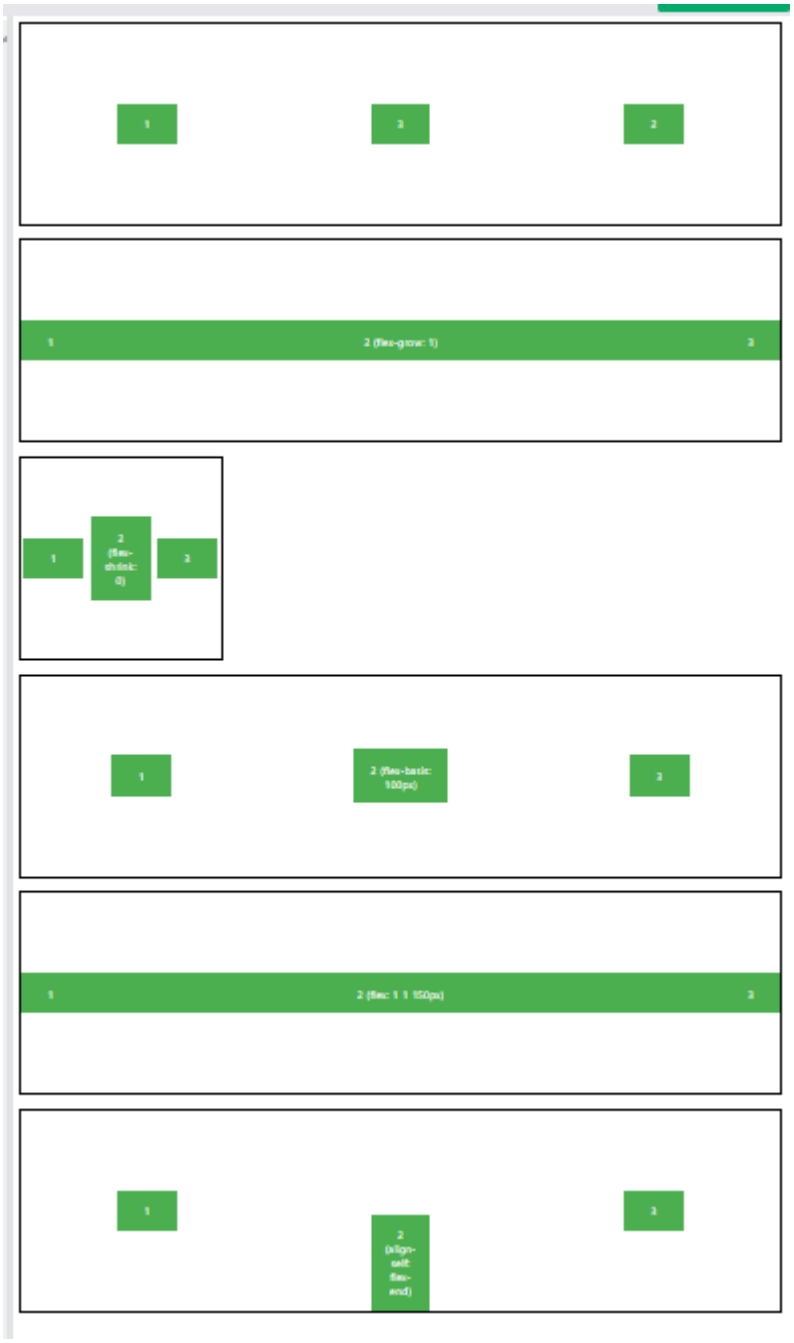
```
.flex-container {
  display: flex;
  justify-content: center;
  align-items: center;
  height: 300px;
  background-color: DodgerBlue;
}

.flex-container > div {
  background-color: #f1f1f1;
  color: white;
  width: 100px;
  height: 100px;
}
</style>
</head>
<body>

<h1>Perfect Centering</h1>

<p>A flex container with both the justify-content and the align-items properties set to <em>center</em> will align the item(s) in the center (in both axis).</p>

<div class="flex-container">
```

```
  <div></div>
</div>


</body>
</html>
```

다음은 `order`, `flex-grow`, `flex-shrink`, `flex-basis`, `flex`, `align-self` 속성들을 사용하여 아이템의 정렬, 크기 조정, 순서 등을 확인할 수 있는 예제입니다.

설명:

- **`order`**: Flex 아이템의 순서를 지정합니다. 기본값은 `0`이며, 값이 작을수록 앞에, 클수록 뒤에 배치됩니다.
- **`flex-grow`**: 아이템이 남은 공간을 얼마나 차지할지를 정의합니다. 기본값은 `0`이며, `1` 이상으로 설정하면 여유 공간을 비례적으로 나누어 가집니다.
- **`flex-shrink`**: 아이템이 공간이 부족할 때 얼마나 줄어들지를 정의합니다. 기본값은 `1`이며, `0`으로 설정하면 축소되지 않습니다.
- **`flex-basis`**: 아이템의 기본 크기를 설정합니다. 기본값은 `auto`입니다.
- **`flex`**: `flex-grow`, `flex-shrink`, `flex-basis`를 한 번에 설정할 수 있는 속성입니다.
- **`align-self`**: 개별 아이템의 수직 정렬을 조정합니다. 기본값은 `auto`로, 부모의 `align-items` 속성 값을 따릅니다.



```
<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <meta name="viewport" content="width=device-width, initial-scale=1.0">

  <title>Flexbox Properties Example</title>
```

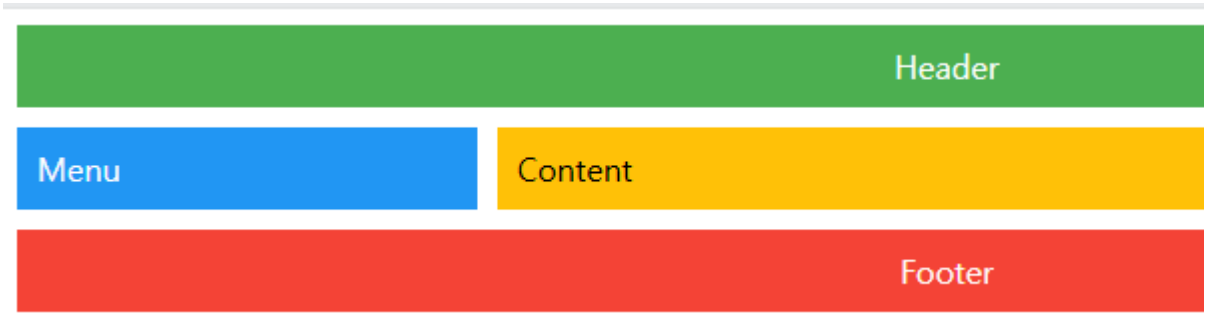
```
<style>
  .flex-container {
    display: flex;
    justify-content: space-around;
    align-items: center;
    height: 300px;
    border: 2px solid #000;
    margin-bottom: 20px;
  }

  .flex-container div {
    background-color: #4CAF50;
    color: white;
    padding: 20px;
    text-align: center;
    width: 50px;
  }

  /* Order Example */
  .order {
    order: 2;
  }

  /* Flex-grow Example */
```

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

  /* Flex-shrink Example */
  .flex-shrink {
    flex-shrink: 0;
  }

  /* Flex-basis Example */
  .flex-basis {
    flex-basis: 100px;
  }

  /* Flex Example */
  .flex {
    flex: 1 1 150px; /* flex-grow: 1, flex-shrink: 1, flex-basis: 150px */
  }

  /* Align-self Example */
  .align-self {
    align-self: flex-end;
  }
</style>
```

```
</head>
<body>

  <!-- Order Example -->
  <div class="flex-container">
    <div>1</div>
    <div class="order">2</div>
    <div>3</div>
  </div>

  <!-- Flex-grow Example -->
  <div class="flex-container">
    <div>1</div>
    <div class="flex-grow">2 (flex-grow: 1)</div>
    <div>3</div>
  </div>

  <!-- Flex-shrink Example -->
  <div class="flex-container" style="width: 300px;">
    <div>1</div>
    <div class="flex-shrink">2 (flex-shrink: 0)</div>
    <div>3</div>
  </div>
```

```
  <!-- Flex-basis Example -->
  <div class="flex-container">
    <div>1</div>
    <div class="flex-basis">2 (flex-basis: 100px)</div>
    <div>3</div>
  </div>

  <!-- Flex Example -->
  <div class="flex-container">
    <div>1</div>
    <div class="flex">2 (flex: 1 1 150px)</div>
    <div>3</div>
  </div>

  <!-- Align-self Example -->
  <div class="flex-container">
    <div>1</div>
    <div class="align-self">2 (align-self: flex-end)</div>
    <div>3</div>
  </div>

</body>
</html>
```

5. 명명된 영역



전체 구조 요약

- **그리드 구조**는 2개의 열과 3개의 행으로 구성됩니다. 각 요소(`header`, `menu`, `content`, `footer`)는 `grid-template-areas`에 정의된 대로 배치됩니다.
- **배경색**을 통해 각 영역을 시각적으로 구분하며, `grid-area`와 `grid-template-areas`로 각 영역의 위치를 정의합니다.

`grid-template-areas:`: 그리드 레이아웃의 영역을 정의합니다. `"header header"`는 `header` 영역이 2열을 차지하는 구조로, `"menu content"`는 `menu`가 왼쪽, `content`가 오른쪽에 위치하도록 설정됩니다. `"footer footer"`는 `footer`가 전체 너비를 차지하도록 합니다.

예시:

- `header`는 두 열을 차지하고
- `menu`는 왼쪽에, `content`는 오른쪽에
- `footer`는 마지막에 전체 너비를 차지합니다.

`grid-area`: 각 항목이 그리드에서 차지할 영역을 지정합니다. 예를 들어, `.header`는 `grid-area: header;`로 지정되어 `grid-template-areas`에서 정의한 `header` 위치에 배치됩니다.

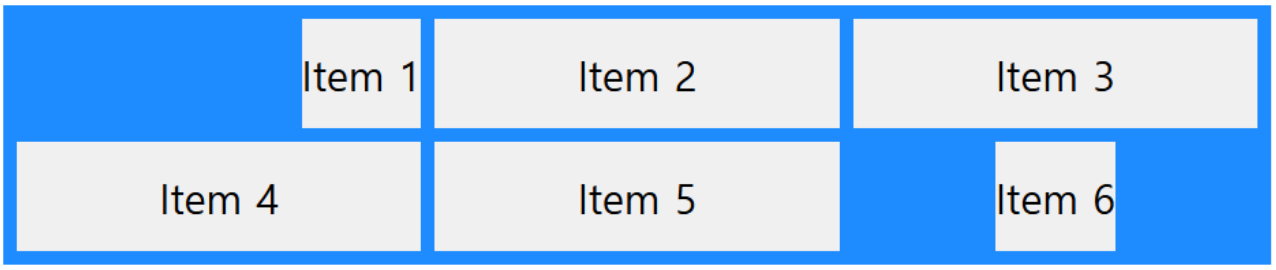
```
<style>

.grid-container {

  display: grid;
```

```
  grid-template-areas:
    "header header"
    "menu content"
    "footer footer";
  grid-template-columns: 1fr 3fr;
  grid-template-rows: auto 1fr auto;
  gap: 10px;
}

.header {
  grid-area: header;
  background-color: #4CAF50; /* 초록색 */
  color: white;
  text-align: center;
  padding: 10px;
}

.menu {
  grid-area: menu;
  background-color: #2196F3; /* 파란색 */
  color: white;
  padding: 10px;
}
```

```
.content {
  grid-area: content;
  background-color: #FFC107; /* 노란색 */
  color: black;
  padding: 10px;
}

.footer {
  grid-area: footer;
  background-color: #F44336; /* 빨간색 */
  color: white;
  text-align: center;
  padding: 10px;
}
</style>

<div class="grid-container">
  <div class="header">Header</div>
  <div class="menu">Menu</div>
  <div class="content">Content</div>
  <div class="footer">Footer</div>
</div>
```

justify-self

The justify-self property

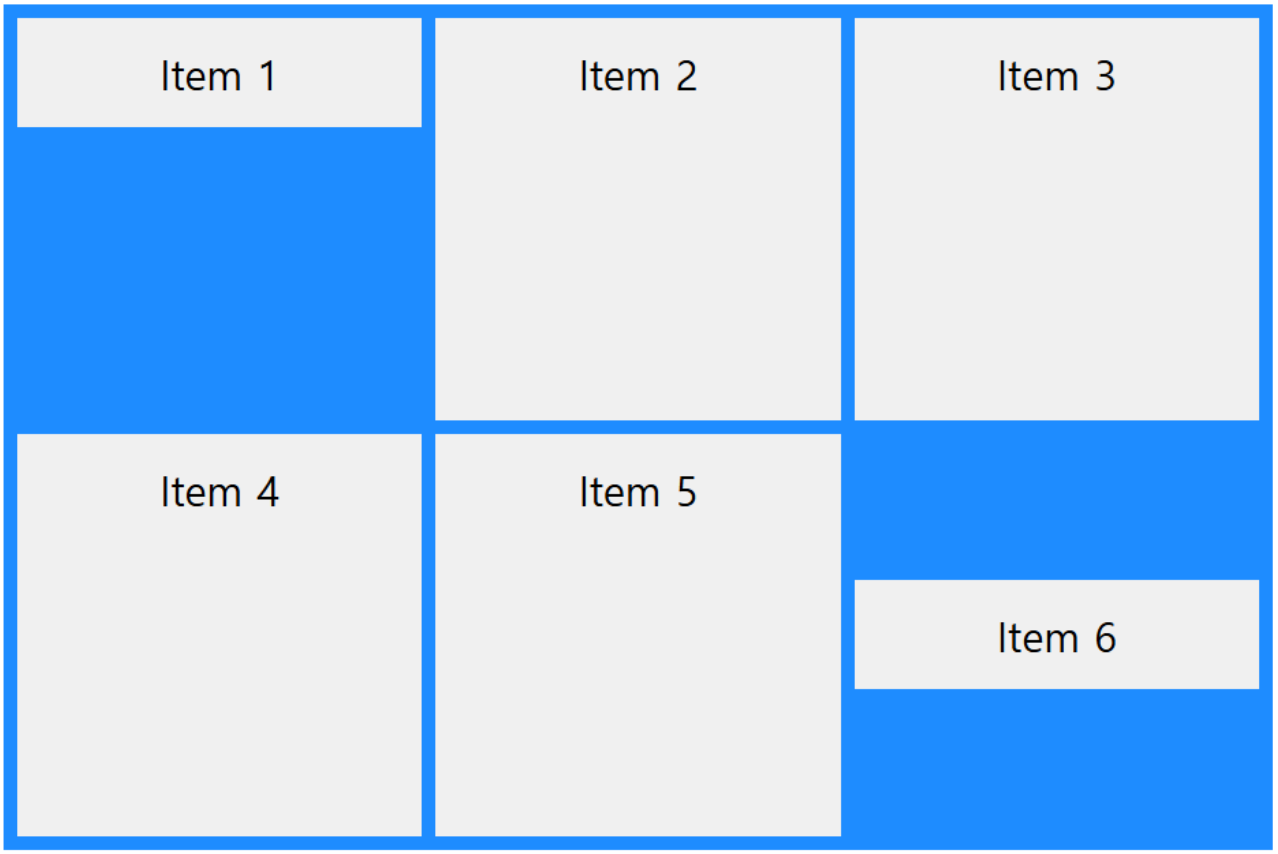



```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 10px;
  background-color: dodgerblue;
  padding: 10px;
}

.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
```

```
}

.item1 {
  justify-self: right;
}

.item6 {
  justify-self: center;
}
</style>
</head>
<body>
<h2>The justify-self property</h2>
<div class="grid-container">
  <div class="item1">Item 1</div>
  <div class="item2">Item 2</div>
  <div class="item3">Item 3</div>
  <div class="item4">Item 4</div>
  <div class="item5">Item 5</div>
  <div class="item6">Item 6</div>
</div>
</body>
</html>
```

align-self

The align-self property




```
<!DOCTYPE html>
<html>
<head>
<style>
.grid-container {
  display: grid;
  height: 600px;
  grid-template-columns: 1fr 1fr 1fr;
  gap: 10px;
```

```
  background-color: dodgerblue;
  padding: 10px;
}

.grid-container > div {
  background-color: #f1f1f1;
  text-align: center;
  padding: 20px 0;
  font-size: 30px;
}

.item1 {
  align-self: start;
}

.item6 {
  align-self: center;
}
</style>
</head>
<body>

<h2>The align-self property</h2>
```

```
<div class="grid-container">

  <div class="item1">Item 1</div>

  <div class="item2">Item 2</div>

  <div class="item3">Item 3</div>

  <div class="item4">Item 4</div>

  <div class="item5">Item 5</div>

  <div class="item6">Item 6</div>

</div>


</body>

</html>
```

> 16. css 반응형 웹

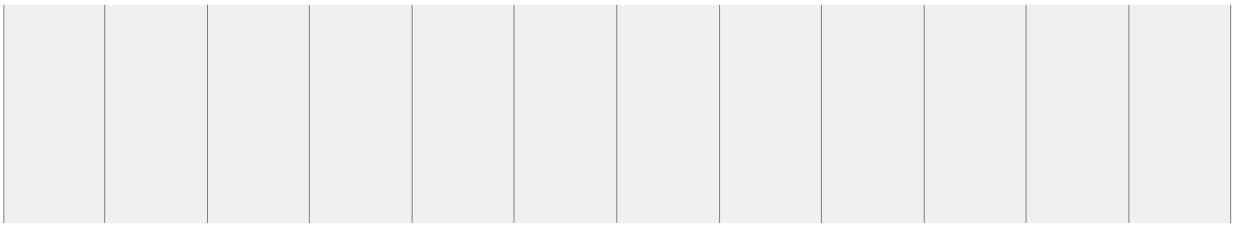
<meta name="viewport" content="width=device-width, initial-scale=1.0">: 이 라인은 모바일 기기에서 페이지가 화면 너비에 맞게 비율100%로 표시되도록 설정합니다.

Without the viewport meta tag

With the viewport meta tag

What is a Grid-View?

Many web pages are based on a grid-view, which means that the page is divided into columns:



Using a grid-view is very helpful when designing web pages. It makes it easier to place elements on the page.

```
.menu {
  width: 25%;
  float: left;
}
.main {
  width: 75%;
  float: left;
}
```


```
<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<style>

* {

  box-sizing: border-box;

}


.header {

  border: 1px solid red;

  padding: 15px;

}


.menu {

  width: 25%;

  float: left;
```

```
  padding: 15px;
  border: 1px solid red;
}

.main {
  width: 75%;
  float: left;
  padding: 15px;
  border: 1px solid red;
}
</style>
</head>
<body>

<div class="header">
  <h1>Chania</h1>
</div>

<div class="menu">
  <ul>
    <li>The Flight</li>
    <li>The City</li>
    <li>The Island</li>
    <li>The Food</li>
  </ul>
```

```
</div>


<div class="main">

  <h1>The City</h1>

  <p>Chania is the capital of the Chania region on the island of Crete. The city can be divided in two parts, the old town and the modern city.</p>

  <p>Resize the browser window to see how the content respond to the resizing.</p>

</div>


</body>

</html>
```

Add the following code in your CSS:

```
* {

  box-sizing: border-box;

}
```

* { box-sizing: border-box; }: 모든 요소의 크기 계산 방식을 변경하여 패딩과 테두리가 요소의 크기에 포함되도록 합니다.

```
  .row::after {
      content: "";
      clear: both;
      display: block;/*block이나 table 비슷하다.*/
    }
```

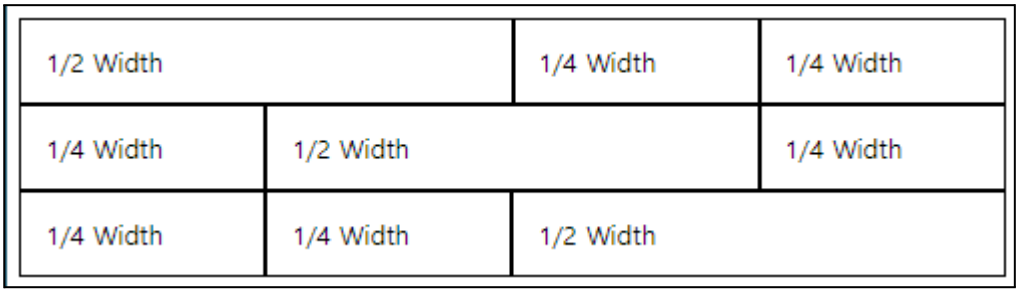
.row::after: .row 클래스의 가상 요소(::after)를 사용하여 부모 요소를 클리어(clear)합니다. 이렇게 하면 부모 요소가 하위 요소의 부유(floating)에 영향을 받지 않고 레이아웃이 유지됩니다.

[class*="col-"]: col-로 시작하는 클래스를 가진 모든 요소를 선택합니다. 이 클래스들은 그리드 시스템의 열(column)을 나타냅니다.

.col-1부터 .col-12까지: 1/12부터 12/12까지 다양한 너비를 가진 열 클래스를 정의합니다. 12를 100%으로 해서 .col-1: 1/12, 즉 약 8.33%의 너비 .col-2: 2/12, 즉 약 16.66%의 너비 .col-3: 3/12, 즉 25%의 너비 .col-4: 4/12, 즉 33.33%의 너비 .col-6: 6/12, 즉 50%의 너비 .col-12: 12/12, 즉 100%의 너비를 의미한다.

: .row 클래스로 묶인 열(column) 그룹을 생성합니다.

, : 다양한 열 클래스를 사용하여 열을 생성합니다. 이 열들은 다른 너비를 가지며, 그리드 시스템을 형성합니다. 12가 100% 이므로 col-6 col-3 col-3과 같이 하나의 row div에 col 크기 합산이 12가 되어 화면에 100이 되도록 출력하면 된다.




```
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <style>
    * {
      box-sizing: border-box;
    }
    .row::after {
      content: "";
      clear: both;
      display: block;/*block이나 table 비슷하다.*/
    }
```

```
    [class*="col-"] {
      float: left;
      padding: 15px;
      border: 1px solid #000;
    }
    .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%;}
  </style>
</head>
<body>
<div class="row">
  <div class="col-6">1/2 Width</div>
  <div class="col-3">1/4 Width</div>
  <div class="col-3">1/4 Width</div>
</div>
<div class="row">
  <div class="col-3">1/4 Width</div>
  <div class="col-6">1/2 Width</div>
  <div class="col-3">1/4 Width</div>
</div>
<div class="row">
  <div class="col-3">1/4 Width</div>
  <div class="col-3">1/4 Width</div>
  <div class="col-6">1/2 Width</div>
</div>
</body>
</html>
```

메디아 쿼리를 이용해서 화면 크기에 따라 다른 css를 적용할 수 있다. 다음 예제는 화면 크기에 따라

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<style>
.example {
  padding: 20px;
  color: white;
}
/* 초소형 기기 (스마트폰, 600px 이하) */
@media only screen and (max-width: 600px) {
  .example {background: red;}
}

/* 소형 기기 (태블릿 세로 모드 및 대형 스마트폰, 600px 이상) */
@media only screen and (min-width: 600px) {
  .example {background: green;}
}

/* 중형 기기 (태블릿 가로 모드, 768px 이상) */
@media only screen and (min-width: 768px) {
  .example {background: blue;}
}

/* 대형 기기 (노트북/데스크탑, 992px 이상) */
@media only screen and (min-width: 992px) {
  .example {background: orange;}
}

/* 초대형 기기 (큰 노트북 및 대형 데스크탑, 1200px 이상) */
@media only screen and (min-width: 1200px) {
  .example {background: pink;}
}
</style>
</head>
<body>

<h2>일반적인 미디어 쿼리 브레이크포인트</h2>
<p class="example">브라우저 창 크기를 조절하여 이 문단의 배경색이 다른 화면
크기에서 어떻게 변경되는지 확인하세요.</p>

</body>
</html>
```

다음은 큰 화면 일때 col- 화면이 적용 되지만 작은 화면 일때 하나의 col-값이 화면을 100% 사용 하는 예제이다. 실행후 화면을 작게 하면 한줄에 하나씩 col div가 출력되는 것을 확인할 수 있다.


```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<style>
* {
  box-sizing: border-box;
}

.row::after {
  content: "";
  clear: both;
  display: block;
}

[class*="col-"] {
  float: left;
  padding: 15px;
}

.header {
  background-color: #FF8F00; /* 가을색 */
  color: #fff;
  padding: 15px;
}

.menu ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}
```

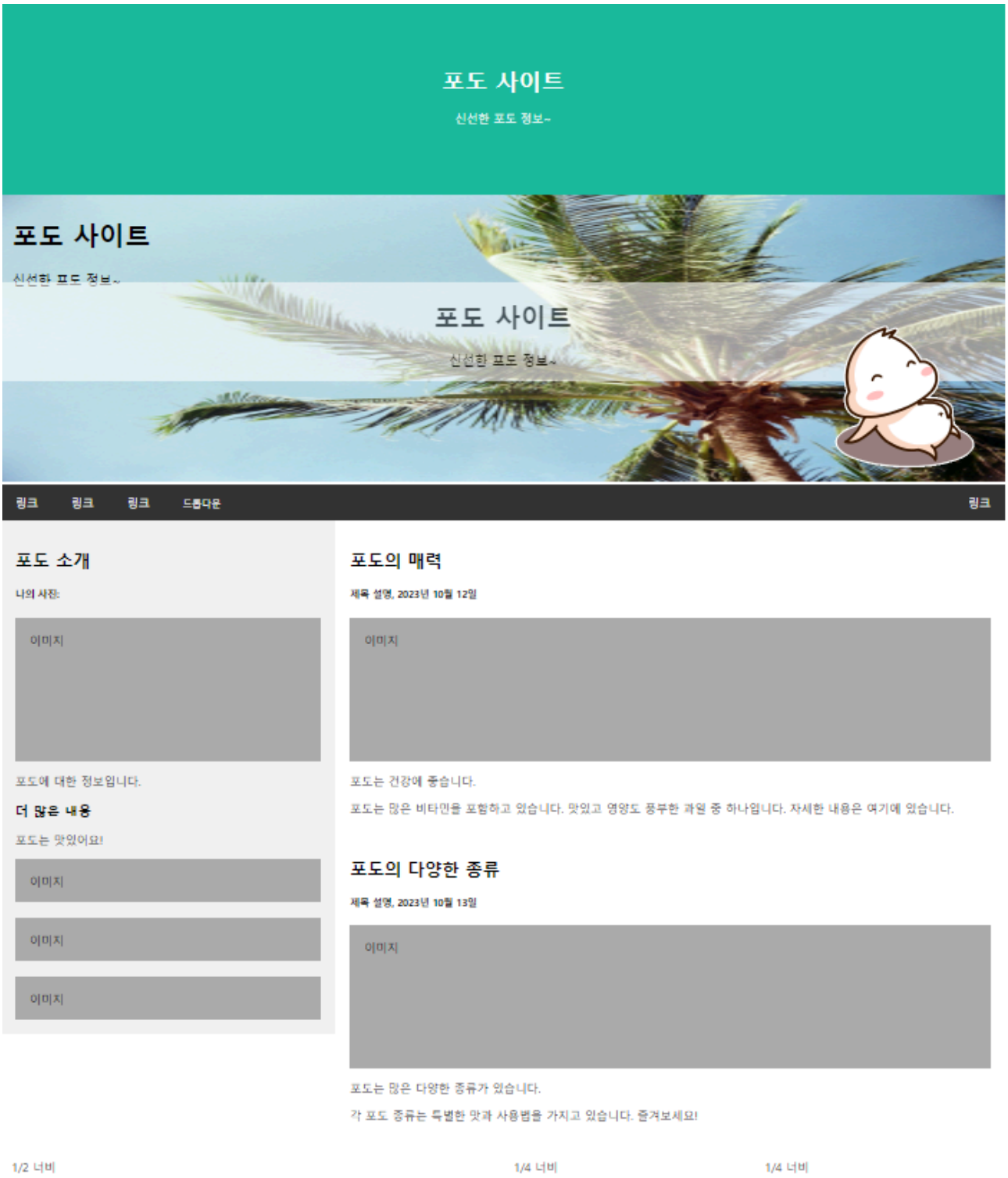
```
.menu li {
  padding: 8px;
  margin-bottom: 7px;
  background-color: #FFC107; /* 밝은 가을색 */
  color: #fff;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.menu li:hover {
  background-color: #FFA000; /* 강조된 가을색 */
}

.aside {
  background-color: #FFC107; /* 밝은 가을색 */
  padding: 15px;
  color: #fff;
  text-align: center;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0,0,0,0.12), 0 1px 2px rgba(0,0,0,0.24);
}

.footer {
  background-color: #FFA000; /* 강조된 가을색 */
  color: #fff;
  text-align: center;
  font-size: 12px;
  padding: 15px;
}

/* 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%;}

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

```
    width: 100%;
  }
}
</style>
</head>
<body>
<div class="header">
  <h1>가을과일마을</h1>
</div>
<div class="row">
  <div class="col-3 menu">
    <ul>
      <li>사과</li>
      <li>바나나</li>
      <li>오렌지</li>
      <li>포도</li>
    </ul>
  </div>

  <div class="col-6">
    <h1>사과</h1>
    <p>사과는 다양한 색과 맛을 가진 맛있는 과일로, 비타민과 식이섬유를 풍부하게 함유하고 있어 건강한 간식입니다.</p>
  </div>

  <div class="col-3">
    <div class="aside">
      <h2>이점</h2>
      <p>사과는 심장 건강을 개선하고 체중 감량을 돕고 면역 체계를 강화하는 데 도움이 됩니다.</p>
      <h2>다양한 종류</h2>
      <p>주로 볼 수 있는 사과 종류로는 레드 디리셔스, 그랜니 스미스, 후지가 있습니다.</p>
    </div>
  </div>
</div>
<div class="footer">
  <p>화면 크기를 조절하여 콘텐츠가 크기에 따라 어떻게 반응하는지 확인해보세요.</p>
</div>
</body>
</html>
```

다음 사이트에서 flex를 공부해 보자.

https://heropy.blog/2018/11/24/css-flexible-box/

다음 예제는 다음과 같은 사이트를 만드는 예제이다. 화면이 작아지면 한줄에 하나씩 div가 출력된다.



html

```
<!DOCTYPE html>
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>포도 웹사이트</title>
    <link rel="stylesheet" href="css.css">
</head>
<body>
    <!-- 1번째 해더-->
    <div class="header">
        <h1>포도 사이트</h1>
        <p>신선한 포도 정보~</p>
    </div>
    <!-- 2번째 해더-->
    <div class="container">
        <div class="container-topleft">
            <h1>포도 사이트</h1>
            <p>신선한 포도 정보~</p>
        </div>
        <div class="container-center  container-background
container-opacity">
            <h1>포도 사이트</h1>
            <p>신선한 포도 정보~</p>
        </div>
          <div class="container-bottomright">
           <img src="gif.gif">
        </div>
        <!-- <img src="bg.jpg" width="100%" height="auto" > -->
        <img src="bg.jpg" width="100%" height="400px" >
    </div>
    <div class="navbar sticky">
        <a href="#">링크</a>
        <a href="#">링크</a>
        <a href="#">링크</a>
        <a href="#" class="right">링크</a>
        <!--드롭다운 메뉴 시작-->
        <div class="dropdown">
            <button>드롭다운</button>
            <div class="dropdown-content">
                <a href="#">링크</a>
                <a href="#">링크</a>
                <a href="#">링크</a>
                <div class="dropdown">
                    <button>드롭다운</button>
                    <div class="dropdown-content">
                        <a href="#">링크링크링크링크링크</a>
                        <a href="#">링크</a>
```

```
                        <a href="#">링크</a>
                    </div>
                </div>
            </div>
        </div>
        <!--드롭다운 메뉴 종료-->
    </div>

    <div class="row">
        <div class="side col-4">
            <h2>포도 소개</h2>
            <h5>나의 사진:</h5>
            <div class="fakeimg" style="height:200px;">이미지</div>
            <p>포도에 대한 정보입니다.</p>
            <h3>더 많은 내용</h3>
            <p>포도는 맛있어요!</p>
            <div class="fakeimg" style="height:60px;">이미지</div><br>
            <div class="fakeimg" style="height:60px;">이미지</div><br>
            <div class="fakeimg" style="height:60px;">이미지</div>
        </div>
        <div class="main col-8">
            <h2>포도의 매력</h2>
            <h5>제목 설명, 2023년 10월 12일</h5>
            <div class="fakeimg" style="height:200px;">이미지</div>
            <p>포도는 건강에 좋습니다.</p>
            <p>포도는 많은 비타민을 포함하고 있습니다. 맛있고 영양도 풍부한 과일 중
하나입니다. 자세한 내용은 여기에 있습니다.</p>
            <br>
            <h2>포도의 다양한 종류</h2>
            <h5>제목 설명, 2023년 10월 13일</h5>
            <div class="fakeimg" style="height:200px;">이미지</div>
            <p>포도는 많은 다양한 종류가 있습니다.</p>
            <p>각 포도 종류는 특별한 맛과 사용법을 가지고 있습니다.
즐겨보세요!</p>
        </div>
    </div>
    <div class="row">
        <div class="col-6">1/2 너비</div>
        <div class="col-3">1/4 너비</div>
        <div class="col-3">1/4 너비</div>
    </div>
    <div class="footer">
        <h2>푸터</h2>
    </div>
    <br><br><br><br><br><br><br><br><br><br><br><br><br><br><br><br>
</body>
</html>
```

css

```css
*{
    box-sizing: border-box;
}

.header{
    text-align: center;
    padding: 80px;
    background: #1abc9c;
    color: white;
}

.container{
    position: relative;
}

.container-topleft{
    position: absolute;
    top:8px;
    left:16px;
    font-size: 18px;
}

.container-center{
    position: absolute;
    top:30%;
    width:100%;
    text-align: center;
    font-size: 18px;
}
.container-background{
    background: white;
}
.container-opacity{
    opacity: 0.6;
}

.container-bottomright{
    position: absolute;
    bottom:8px;
    right:16px;
    font-size: 18px;
}
```

```css
.navbar{
    background-color: #333;
    height:auto;
}

.navbar a{
    /*
    float:left;
    display:block;
    */
    display: inline-block;
    color:white;
    text-align: center;
    padding: 14px 20px;
    text-decoration: none;
}

.navbar a.right{
    float:right;
}

.navbar a:hover{
    background-color: #ddd;
    color:black;
}

/*dropdown menu*/
.navbar button{
    display:inline-block;
    border: none;
    background-color: #333;
    color: white;
    text-align: center;
    padding: 14px 20px;
}
.navbar button:hover{

    background-color:  white;
    color: #333;
}
.dropdown{
    position:relative;
    display: inline-block;
}

.dropdown-content{
    display:none;
```

```
    position: absolute;
    background-color: #333;
    padding: 12px 16px;
    white-space:nowrap;
/* 1번 */

}

.dropdown:hover>.dropdown-content{
    display: block;
}

.row::after {
  content: "";
  clear: both;
  display: block;
}
/* 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%;}

[class*="col-"] {
  float: left;
  padding: 15px;
}

  .side {

    background-color: #f1f1f1;
    padding: 20px;
  }

  /* Main column */
  .main {

    background-color: white;
    padding: 20px;
```

```
}

/* Fake image, just for this example */
.fakeimg {
  background-color: #aaa;
  width: 100%;
  padding: 20px;
}

/* Footer */
.footer {
  padding: 20px;
  text-align: center;
  background: #ddd;
}
div.sticky{
    position:sticky;
    top:0px;

}

@media screen and (max-width: 700px){
    .navbar a,.navbar button,.dropdown,.dropdown-content{
        width:100%;
    }
     .dropdown-content{
      white-space:normal;
   }


    div.sticky{
        position: relative;
    }
}
```

> 16. css 반응형 웹

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>팝업 메뉴</title>
  <style>
    body {
      margin: 0;
      font-family: Arial, sans-serif;
    }

    .navbar {
      display: flex;
      justify-content: space-between;
      align-items: center;
      background-color: #333;
      color: white;
      padding: 10px 20px;
    }

    .logo {
      font-size: 1.5em;
    }

    .menu-toggle {
      font-size: 1.5em;
      background: none;
      border: none;
      color: white;
      cursor: pointer;
    }

    .menu {
      display: none;
      position: fixed;
      top: 0;
      left: 0;
      width: 100%;
      height: 100%;
      background-color: rgba(0, 0, 0, 0.9);
      flex-direction: column;
      justify-content: center;
```

```
      align-items: center;
      list-style: none;
      margin: 0;
      padding: 0;
      z-index: 1000;
    }

    .menu li {
      margin: 20px 0;
    }

    .menu a {
      color: white;
      font-size: 1.5em;
      text-decoration: none;
      transition: color 0.3s;
    }

    .menu a:hover {
      color: #f0a500;
    }

    .menu.show {
      display: flex;
    }

    .close-btn {
      position: absolute;
      top: 20px;
      right: 20px;
      background: none;
      border: none;
      color: white;
      font-size: 1.5em;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <nav class="navbar">
    <div class="logo">My Site</div>
    <button class="menu-toggle" id="menuToggle">三</button>
  </nav>
  <ul class="menu" id="menu">
    <button class="close-btn" id="closeMenu">✕</button>
    <li><a href="#home">홈</a></li>
    <li><a href="#about">소개</a></li>
    <li><a href="#services">서비스</a></li>
```

```
    <li><a href="#contact">연락처</a></li>
  </ul>
  <script>
    const menuToggle = document.getElementById('menuToggle');
    const closeMenu = document.getElementById('closeMenu');
    const menu = document.getElementById('menu');

    // 메뉴 열기
    menuToggle.addEventListener('click', () => {
      menu.classList.toggle('show');
    });

    // 메뉴 닫기 (닫기 버튼 클릭)
    closeMenu.addEventListener('click', () => {
      menu.classList.remove('show');
    });

    // 메뉴 닫기 (배경 클릭)
    menu.addEventListener('click', (e) => {
      if (e.target === menu) {
        menu.classList.remove('show');
      }
    });
  </script>
</body>
</html>
```