Flexbox

[Flexbox](http://www.w3.org/TR/css3-flexbox/)의 정식 명칭은 ‘CSS Flexible Box Layout ‘이며 이름 그대로 유연한(flex) 박스 모델을 위한 CSS 스펙이다. Flexbox는 여러 요소들을 플렉스 컨테이너(flex container)로 감싸는 방식으로 구현한다. 이 플렉스 컨테이너 속에 들어간 요소(자식 박스)들은 가로 세로 아주 유연하게 배치될 수 있는 것이 Flexbox의 장점이다.

어떤 요소가 flex 컨테이너가 되면, 그 속의 자식 요소들은 ‘flex 아이템(item)’이 되어 flex box 모델의 적용을 받는다. flexbox 모델의 기본 값은 각각의 자식 요소들이 동일한 크기의 영역을 차지한다.

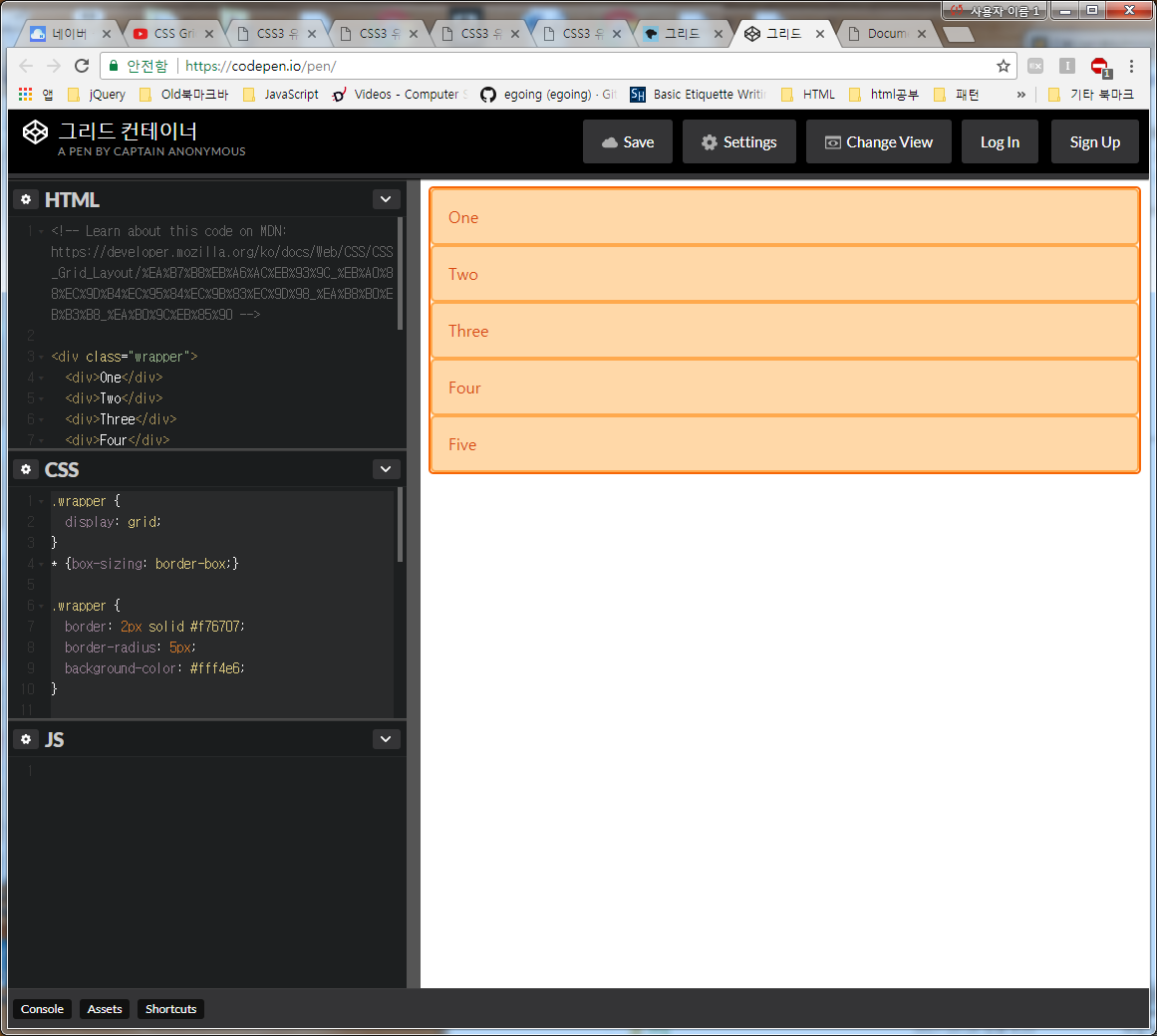
## CSS Grid Layout

[CSS Grid Layout](https://www.w3.org/TR/css-grid/)은 그야말로 ‘그리드 레이아웃’만을 위한 CSS 모듈이다. 가로세로 격자를 만들어 그 속에 디자인 요소들을 배치한다.

그리드 컨테이너는 요소에 display: grid 또는 display: inline-grid를 선언하여 만듭니다. 이렇게 하면 해당 요소 바로 밑에 있는 모든 자식 요소는 그리드 아이템이 됩니다.

속성들

* [grid-template-columns](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-columns)
* [grid-template-rows](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-rows)
* [grid-template-areas](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-areas)
* [grid-template](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template)
* [grid-auto-columns](https://developer.mozilla.org/ko/docs/Web/CSS/grid-auto-columns)
* [grid-auto-rows](https://developer.mozilla.org/ko/docs/Web/CSS/grid-auto-rows)
* [grid-auto-flow](https://developer.mozilla.org/ko/docs/Web/CSS/grid-auto-flow)
* [grid](https://developer.mozilla.org/ko/docs/Web/CSS/grid)
* [grid-row-start](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row-start)
* [grid-column-start](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column-start)
* [grid-row-end](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row-end)
* [grid-column-end](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column-end)
* [grid-row](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row)
* [grid-column](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column)
* [grid-area](https://developer.mozilla.org/ko/docs/Web/CSS/grid-area)
* [grid-row-gap](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row-gap)
* [grid-column-gap](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column-gap)
* [grid-gap](https://developer.mozilla.org/ko/docs/Web/CSS/grid-gap)



예제1)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

.wrapper {

display: grid;

}

\* {box-sizing: border-box;}

.wrapper {

border: 2px solid #f76707;

border-radius: 5px;

background-color: #fff4e6;

}

.wrapper > div {

border: 2px solid #ffa94d;

border-radius: 5px;

background-color: #ffd8a8;

padding: 1em;

color: #d9480f;

}

.nested {

border: 2px solid #ffec99;

border-radius: 5px;

background-color: #fff9db;

padding: 1em;

}

</style>

</head>

<body>

<div class="wrapper">

<div>One</div>

<div>Two</div>

<div>Three</div>

<div>Four</div>

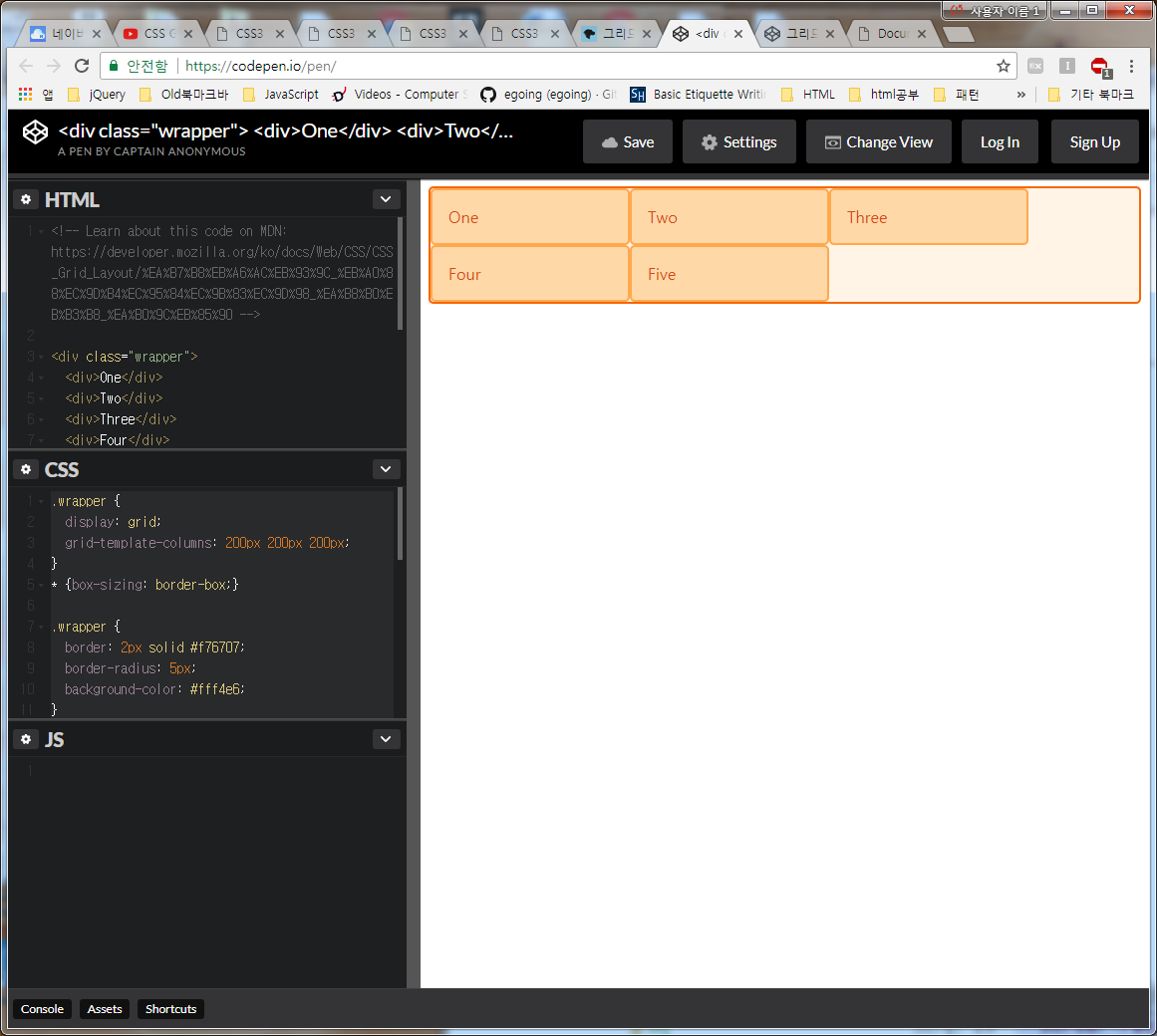
<div>Five</div>

</div>

</body>

</html>

그리드의 행과 열은 [grid-template-columns](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-columns) 및 [grid-template-rows](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-rows) 프로퍼티로 정의합니다.



예제2)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<title>Document</title>

<style>

.wrapper {

display: grid;

grid-template-columns: 200px 200px 200px;

}

\* {box-sizing: border-box;}

.wrapper {

border: 2px solid #f76707;

border-radius: 5px;

background-color: #fff4e6;

}

.wrapper > div {

border: 2px solid #ffa94d;

border-radius: 5px;

background-color: #ffd8a8;

padding: 1em;

color: #d9480f;

}

.nested {

border: 2px solid #ffec99;

border-radius: 5px;

background-color: #fff9db;

padding: 1em;

}

</style>

</head>

<body>

<div class="wrapper">

<div>One</div>

<div>Two</div>

<div>Three</div>

<div>Four</div>

<div>Five</div>

</div>

</body>

</html>

### **fr 단위**

트랙은 모든 종류의 길이 단위를 써서 정의할 수 있습니다. 또한, 그리드에는 유연한 크기의 그리드 트랙을 생성하는 데 사용할 수 있는 단위를 추가로 소개하고 있습니다. 새로 소개된 fr 단위는 그리드 컨테이너에 남아 있는 사용 가능한 공간의 일정 비율을 나타냅니다. 다음에 정의된 그리드에서는 남아 있는 공간에 따라 확장 및 축소합니다

위의 예제2)에서

Style의

.wrapper {

display: grid;

grid-template-columns: 1fr 1fr 1fr;

}

만 바꾼다.

또,

.wrapper {

display: grid;

grid-template-columns: 2fr 1fr 1fr;

}

다음 예제에서는 2fr 크기의 트랙 하나와 1fr 트랙 두 개를 지정하였습니다. 사용 가능한 공간은 4개로 나뉘면서, 공간 두 개는 첫 번째 트랙에 제공하고 나머지 공간 한 부분씩을 다음 두 트랙에 각각 제공합니다.

.wrapper {

display: grid;

grid-template-columns: 500px 1fr 2fr;

}

이번 마지막 예에서는 고정된 크기의 트랙과 비율 단위로 지정한 트랙이 섞여 있습니다. 첫 번째 트랙은 500픽셀의 공간을 차지하므로 이 고정 너비는 사용 가능한 공간에서 제외됩니다. 나머지 공간은 3개의 부분으로 나뉘며 2개의 유연한 트랙에 비율에 맞게 할당됩니다.

### **repeat() 표기법으로 트랙 나열**

많은 트랙을 포함하는 커다란 그리드는 repeat() 표기법을 사용하여 트랙의 전체 또는 일부분을 반복해서 나열해 줄 수 있습니다. 예를 들어 다음과 같이 정의된 그리드의 경우:

.wrapper {

display: grid;

grid-template-columns: 1fr 1fr 1fr;

} 를 아래와 같이 repeat() 함수를 이용할 수도 있다

.wrapper {

display: grid;

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

}

반복 표기법은 트랙의 목록 중 일부분에만 사용할 수도 있습니다. 아래 예제에서는 처음엔 20픽셀 크기의 트랙을 생성하고 다음에 1fr 크기의 트랙을 6번 반복해서 채운 후 마지막에 20픽셀 트랙을 붙여서 그리드를 완성합니다.

.wrapper {

display: grid;

grid-template-columns: 20px repeat(6, 1fr) 20px;

}

반복 표기법은 트랙의 목록도 함께 나열해서 지정할 수 있는데, 이렇게 하면 트랙의 반복 패턴을 생성해서 사용하게 됩니다. 다음 예제는 그리드가 10개의 트랙으로 구성되어 있으며, 1fr 크기의 트랙 다음에 2fr 크기 트랙이 위치하고, 이 형태가 5회 반복됩니다.

.wrapper {

display: grid;

grid-template-columns: repeat(5, 1fr 2fr);

}

### **잠재적 그리고 명시적 그리드**

지금까지는 그리드 예제를 만들 때마다 [grid-template-columns](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-columns) 프로퍼티를 써서 세로 열의 트랙을 정의해 주었습니다만, 가로 행은 그리드가 콘텐츠에 맞게 알아서 새로운 행을 만들도록 했습니다. 이렇게 만들어진 행은 잠재적 그리드 안에서 생성됩니다. 명시적 그리드는 [grid-template-columns](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-columns) 및 [grid-template-rows](https://developer.mozilla.org/ko/docs/Web/CSS/grid-template-rows)로 직접 정의한 행과 열로 이루어진 그리드를 말합니다. 만약에 이렇게 정의된 그리드 밖에 무언가를 배치할 땐, 늘어난 콘텐츠 양 때문에 더 많은 그리드 트랙이 필요하고, 그리드는 잠재적 그리드에 새로운 행과 열을 만들게 됩니다. 이렇게 해서 생긴 트랙은 기본적으로 크기가 자동으로 정해지는데, 트랙 내부의 내용물에 따라 크기가 조정됩니다.

물론 잠재적 그리드에서 생성된 트랙의 크기는 [grid-auto-rows](https://developer.mozilla.org/ko/docs/Web/CSS/grid-auto-rows) 및 [grid-auto-columns](https://developer.mozilla.org/ko/docs/Web/CSS/grid-auto-columns) 프로퍼티를 써서 지정해줄 수도 있습니다.

예제2)에서

.wrapper {

display: grid;

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

grid-auto-rows: 200px;

}

### **트랙 크기 조정과 minmax()**

명시적 그리드를 설정하거나 자동으로 생성된 행이나 열의 크기를 정의할 때, 트랙의 최소 크기를 정해도 나중에 추가되는 콘텐츠에 맞게 늘어나도록 하고 싶을 때가 있을 겁니다. 예를 들면, 가로 행의 높이가 100픽셀 밑으로 줄어드는 것은 막고 싶지만, 콘텐츠가 300픽셀 높이까지 늘어나면 그에 따라 행의 높이도 같이 늘어나길 원할 때가 있습니다.

이런 경우를 고려해서 그리드는 [minmax()](https://developer.mozilla.org/ko/docs/Web/CSS/minmax) 함수를 제공하고 있습니다. 다음 예제에서는 [grid-auto-rows](https://developer.mozilla.org/ko/docs/Web/CSS/grid-auto-rows) 값에 minmax() 를 사용하였습니다. 자동으로 생성된 행의 높이는 최소 100픽셀이고 최댓값은 auto입니다. auto를 지정하면 크기는 콘텐츠의 크기를 살피게 되는데, 가로 행에 있는 가장 높은 셀의 크기만큼 자동으로 늘어나서 부족한 공간을 메꿔줍니다.

<div class="wrapper">

<div>One</div>

<div>Two

<p>I have some more content in.</p>

<p>This makes me taller than 100 pixels.</p>

</div>

<div>Three</div>

<div>Four</div>

<div>Five</div>

</div>

Css :

.wrapper {

display: grid;

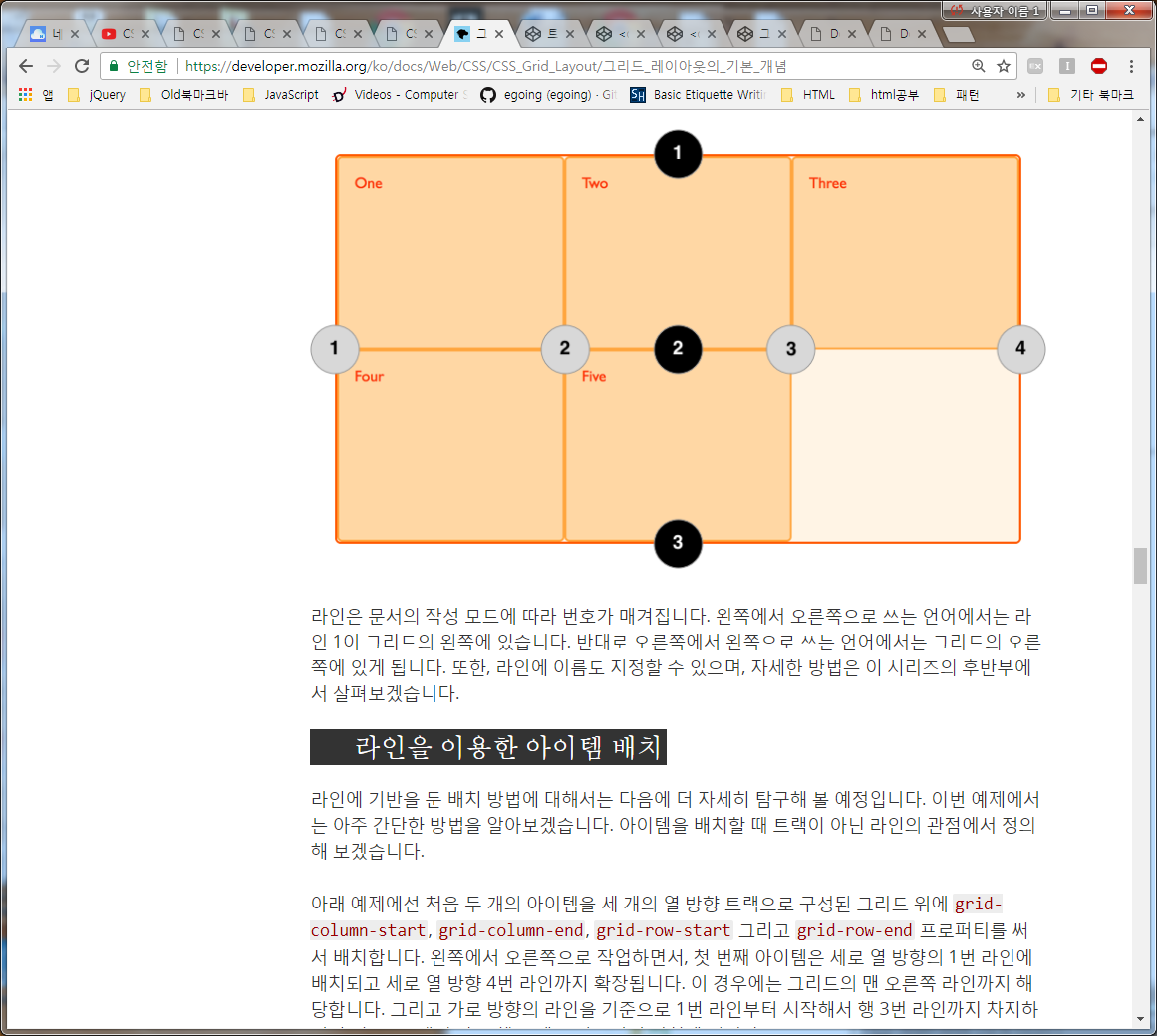
grid-template-columns: repeat(3, 1fr);

grid-auto-rows: minmax(100px, auto);

}

**그리드 라인**

그리드를 정의할 때 라인이 아닌 그리드 트랙을 정의한다는 점을 명심해야 합니다. 그러면 그리드는 아이템을 배치할 때 쓸 수 있게 번호가 매겨진 라인을 자동으로 제공합니다. 아래 그림처럼, 그리드가 세 개의 세로 열과 두 개의 가로 행으로 이루어져 있다면, 네 개의 세로 행 라인이 자리 잡게 됩니다.



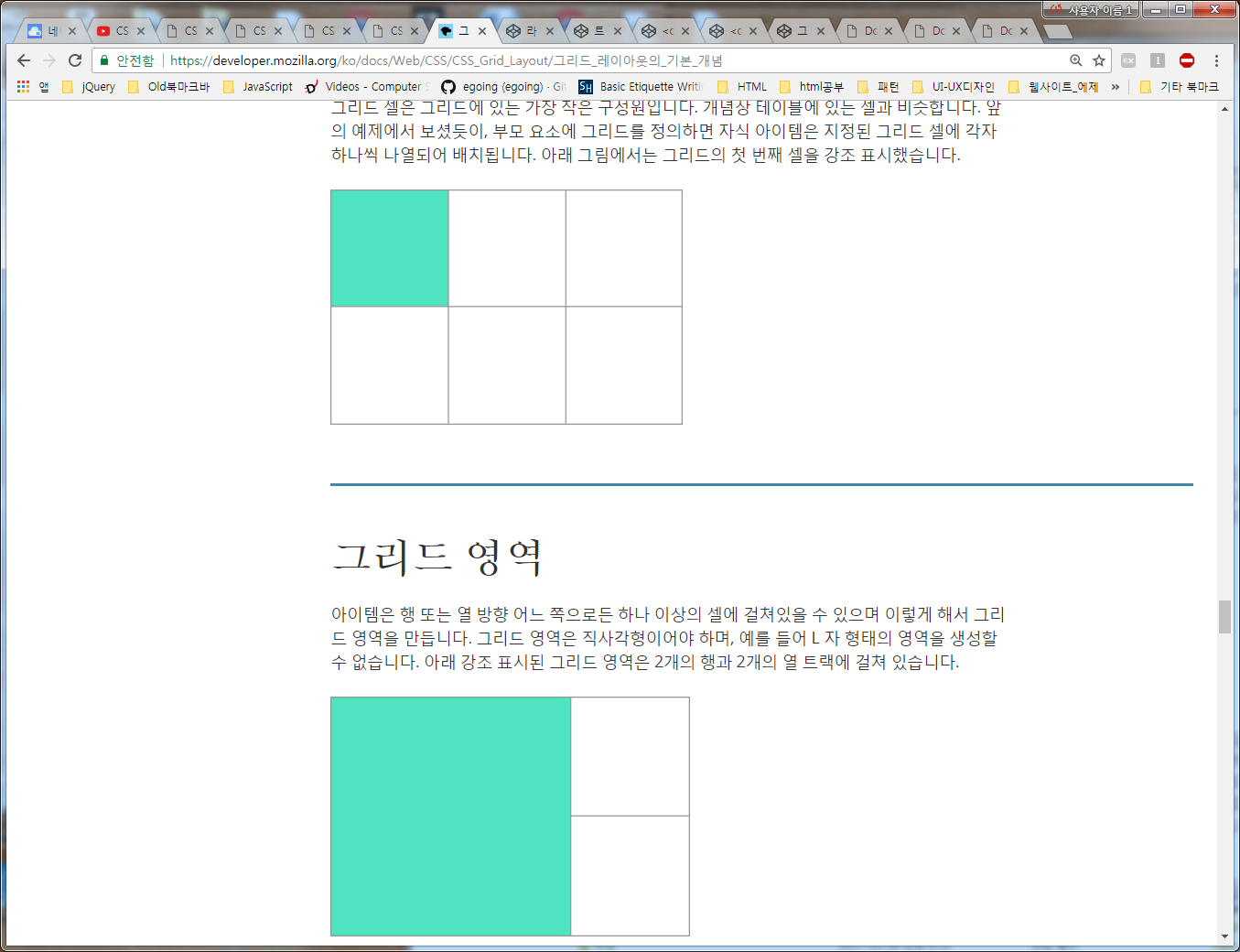
라인은 문서의 작성 모드에 따라 번호가 매겨집니다. 왼쪽에서 오른쪽으로 쓰는 언어에서는 라인 1이 그리드의 왼쪽에 있습니다. 반대로 오른쪽에서 왼쪽으로 쓰는 언어에서는 그리드의 오른쪽에 있게 됩니다. 또한, 라인에 이름도 지정할 수 있으며, 자세한 방법은 이 시리즈의 후반부에서 살펴보겠습니다

아래 예제에선 처음 두 개의 아이템을 세 개의 열 방향 트랙으로 구성된 그리드 위에 [grid-column-start](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column-start), [grid-column-end](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column-end), [grid-row-start](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row-start) 그리고 [grid-row-end](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row-end) 프로퍼티를 써서 배치합니다. 왼쪽에서 오른쪽으로 작업하면서, 첫 번째 아이템은 세로 열 방향의 1번 라인에 배치되고 세로 열 방향 4번 라인까지 확장됩니다. 이 경우에는 그리드의 맨 오른쪽 라인까지 해당합니다. 그리고 가로 방향의 라인을 기준으로 1번 라인부터 시작해서 행 3번 라인까지 차지하면서 결국, 두 개의 가로 행 트랙을 가로질러 걸치게 됩니다.

두 번째 아이템은 그리드 세로 열의 1번 라인에서 시작해서 트랙 하나에 걸쳐 있습니다. 여기엔 기본값을 사용해서 마지막 라인은 지정할 필요가 없습니다. 가로 행 방향 라인을 기준으로 3번의 라인부터 5번 라인까지 두 개의 가로 행 트랙에 걸쳐 있습니다. 나머지 아이템은 그리드 빈 자라에 자동으로 배치됩니다.

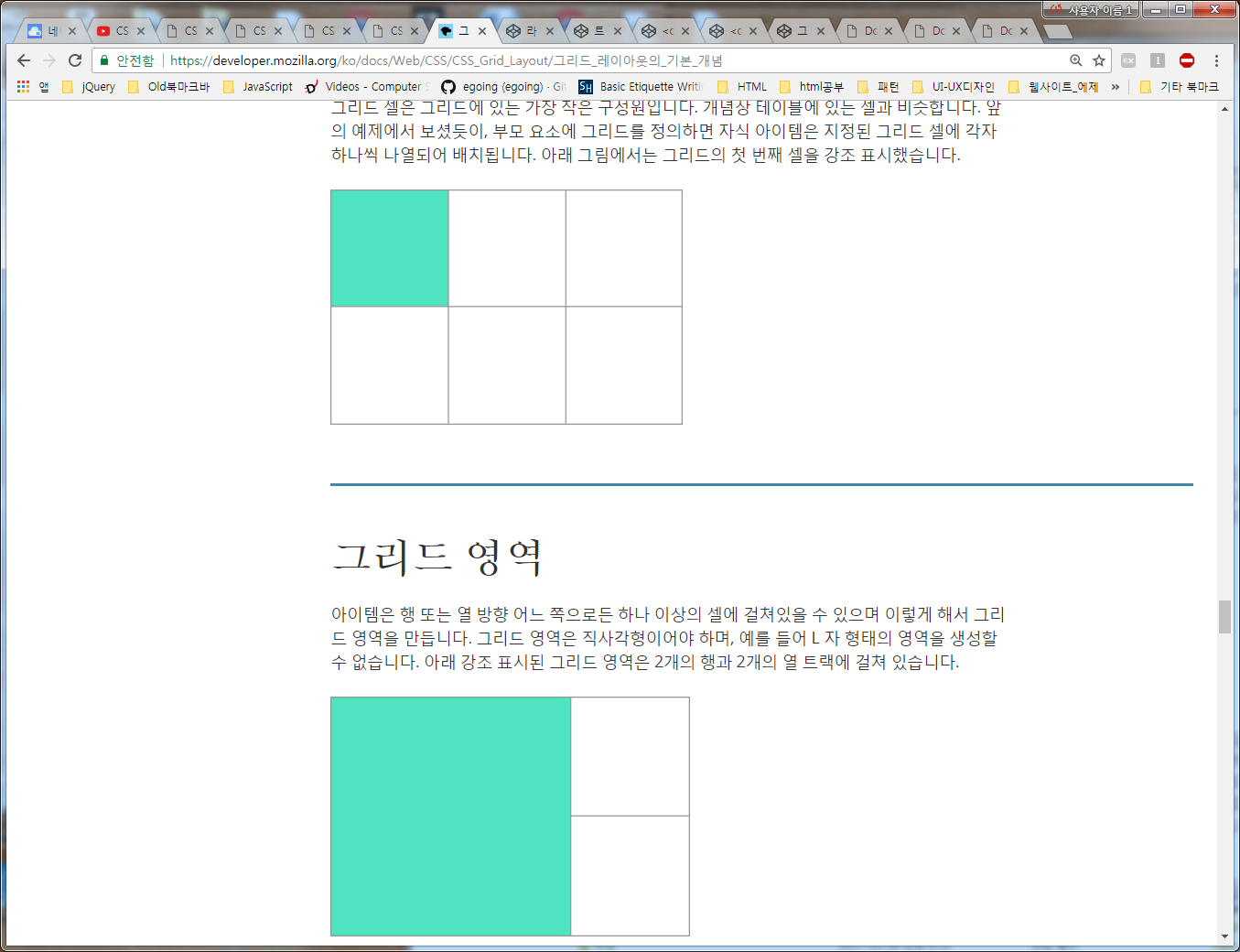
그리드 셀

그리드 셀은 그리드에 있는 가장 작은 구성원입니다. 개념상 테이블에 있는 셀과 비슷합니다. 앞의 예제에서 보셨듯이, 부모 요소에 그리드를 정의하면 자식 아이템은 지정된 그리드 셀에



그리드 영역

아이템은 행 또는 열 방향 어느 쪽으로든 하나 이상의 셀에 걸쳐있을 수 있으며 이렇게 해서 그리드 영역을 만듭니다. 그리드 영역은 직사각형이어야 하며, 예를 들어 L 자 형태의 영역을 생성할 수 없습니다. 아래 강조 표시된 그리드 영역은 2개의 행과 2개의 열 트랙에 걸쳐 있습니다.



## 경계 여백

그리드 셀 사이의 경계 여백 혹은 간격은 [grid-column-gap](https://developer.mozilla.org/ko/docs/Web/CSS/grid-column-gap) 및 [grid-row-gap](https://developer.mozilla.org/ko/docs/Web/CSS/grid-row-gap) 프로퍼티를 지정해서 생성할 수 있으며, 짧게 [grid-gap](https://developer.mozilla.org/ko/docs/Web/CSS/grid-gap)으로 지정할 수도 있습니다. 아래의 예제에서는 세로 열 사이의 간격은 10픽셀이고 가로 행 사이의 간격은 1em입니다

예제3)

<div class="wrapper">

<div>One</div>

<div>Two</div>

<div>Three</div>

<div>Four</div>

<div>Five</div>

</div>

Css:

.wrapper {

display: grid;

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

grid-column-gap: 10px;

grid-row-gap: 1em;

}

\* {box-sizing: border-box;}

.wrapper {

border: 2px solid #f76707;

border-radius: 5px;

background-color: #fff4e6;

}

.wrapper > div {

border: 2px solid #ffa94d;

border-radius: 5px;

background-color: #ffd8a8;

padding: 1em;

color: #d9480f;

}

.nested {

border: 2px solid #ffec99;

border-radius: 5px;

background-color: #fff9db;

padding: 1em;

}

**중첩 그리드**

그리드 아이템은 자기 자신이 그리드 컨테이너가 될 수도 있습니다. 다음 예제는 이전에 생성했던 3개의 세로 열로 구성된 그리드가 있으며, 여기에 따로 위치가 지정된 두 개의 아이템을 배치해 놓은 모습입니다. 이번 경우엔 첫 번째 아이템이 몇 개의 하위 아이템을 포함하고 있습니다. 이 아이템들은 그리드 바로 밑에 있는 요소가 아니므로 그리드 레이아웃에 관여하지 않고 정상적인 문서 흐름에 따라 표시됩니다.

<div class="wrapper">

<div class="box box1">

<div class="nested">a</div>

<div class="nested">b</div>

<div class="nested">c</div>

  </div>

<div class="box box2">Two</div>

<div class="box box3">Three</div>

<div class="box box4">Four</div>

<div class="box box5">Five</div>

</div>

Css:

.box1 {

grid-column-start: 1;

grid-column-end: 4;

grid-row-start: 1;

grid-row-end: 3;

display: grid;

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

}

\* {box-sizing: border-box;}

.wrapper {

border: 2px solid #f76707;

border-radius: 5px;

background-color: #fff4e6;

}

.box {

border: 2px solid #ffa94d;

border-radius: 5px;

background-color: #ffd8a8;

padding: 1em;

color: #d9480f;

}

.nested {

border: 2px solid #ffec99;

border-radius: 5px;

background-color: #fff9db;

padding: 1em;

}

.box1 {

grid-column-start: 1;

grid-column-end: 4;

grid-row-start: 1;

grid-row-end: 3;

display: subgrid;

}

## z-index를 이용한 아이템 중첩도 조정

그리드 아이템은 서로 같은 셀을 차지할 수 있습니다. 앞서 라인 번호로 아이템의 위치를 지정했던 예제로 돌아가서, 여기에 다음과 같이 두 아이템이 서로 자리가 겹치도록 수정합니다.

<div class="wrapper">

<div class="box box1">One</div>

<div class="box box2">Two</div>

<div class="box box3">Three</div>

<div class="box box4">Four</div>

<div class="box box5">Five</div>

</div>

Css :

.wrapper {

display: grid;

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

grid-auto-rows: 100px;

}

.box1 {

grid-column-start: 1;

grid-column-end: 4;

grid-row-start: 1;

grid-row-end: 3;

}

.box2 {

grid-column-start: 1;

grid-row-start: 2;

grid-row-end: 4;

}

\* {box-sizing: border-box;}

.wrapper {

border: 2px solid #f76707;

border-radius: 5px;

background-color: #fff4e6;

}

.box {

border: 2px solid #ffa94d;

border-radius: 5px;

background-color: #ffd8a8;

padding: 1em;

color: #d9480f;

}

.nested {

border: 2px solid #ffec99;

border-radius: 5px;

background-color: #fff9db;

padding: 1em;

}

### **중첩의 순서 조정**

위치를 따로 지정한 아이템과 마찬가지로, 아이템이 위아래로 중첩되는 순서는 z-index 프로퍼티를 써서 제어할 수 있습니다. 만약에 box2에 box1 보다 더 낮은 z-index를 지정하면 box1아래로 중첩되어 표시됩니다.

위의예제에서

.wrapper {

display: grid;

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

grid-auto-rows: 100px;

}

.box1 {

grid-column-start: 1;

grid-column-end: 4;

grid-row-start: 1;

grid-row-end: 3;

z-index: 2;

}

.box2 {

grid-column-start: 1;

grid-row-start: 2;

grid-row-end: 4;

z-index: 1;

}

\* {box-sizing: border-box;}

.wrapper {

border: 2px solid #f76707;

border-radius: 5px;

background-color: #fff4e6;

}

.box {

border: 2px solid #ffa94d;

border-radius: 5px;

background-color: #ffd8a8;

padding: 1em;

color: #d9480f;

}

.nested {

border: 2px solid #ffec99;

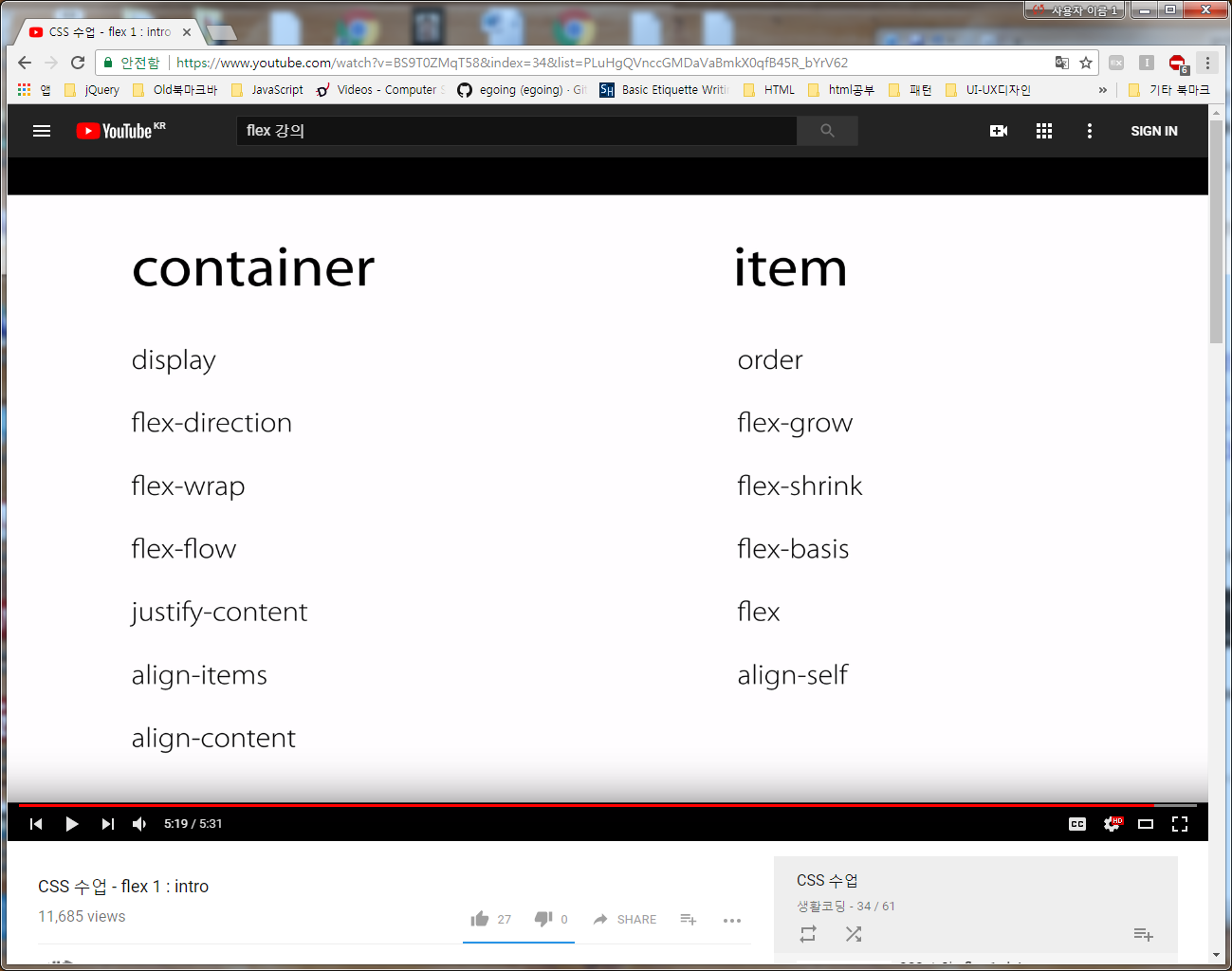
border-radius: 5px;

background-color: #fff9db;

padding: 1em;

}

기존에 우리가 사용하던 레이아웃 기법은 display, float, position 으로써 컬럼 레이아웃을 표현하는데 한계가 있고 구현 방법이 복잡한 문제가 있었는데요. flexible(신축성 있는, 유연한) 박스 모델의 장점을 한 마디로 표현하면 “**복잡한 계산 없이 박스의 크기와 순서를 유연하게 배치할 수 있다.**” 라고 정리할 수 있습니다. 쉬운 예를 들면 컬럼의 한 쪽은 고정하고 다른 한 쪽을 가변폭으로 처리하고 싶을 때 유용하지만 그것 이상의 편의를 제공합니다. 어떤 속성과 값을 통해 무엇을 할 수 있는지 설명해 보겠습니다.



Container, item 이라는 태그가 있는 것이 아니라

여기서 container 는 바깥쪽에 부여해야 하는 속성이고

Item 는 container 의 내부에 있는 아이템들에게 부여해야 하는 속성을 말한다

[플렉스 컨테이너와 플렉스 아이템(flex container & flex item)의 개념.](http://naradesign.net/wp/2017/04/20/2363/#flex-container-item)

[flex container](http://naradesign.net/wp/2017/04/20/2363/#flex-container)

[flex item](http://naradesign.net/wp/2017/04/20/2363/#flex-item)

[신축성(flexibility): flex item의 팽창 및 수축.](http://naradesign.net/wp/2017/04/20/2363/#flexibility)

[flex item의 팽창을 제어하는 ‘flex-grow’ 속성.](http://naradesign.net/wp/2017/04/20/2363/#flex-grow)

[flex item의 수축을 제어하는 ‘flex-shrink’ 속성.](http://naradesign.net/wp/2017/04/20/2363/#flex-shrink)

[flex item의 기준 사이즈를 제어하는 ‘flex-basis’ 속성.](http://naradesign.net/wp/2017/04/20/2363/#flex-basis)

[flex item의 단축 속성 ‘flex’.](http://naradesign.net/wp/2017/04/20/2363/#flex-shorthand)

### 플렉스 컨테이너와 플렉스 아이템(flex container & flex item)의 개념.

http://naradesign.net/wp/2017/04/20/2363/#flex-container

flex 박스 모델은 tr/td 개념와 유사합니다.  flex 박스는 **flex container**(부모, 마치 tr)와 **flex item**(자식, 마치 td)으로 이루어 집니다. **flex container** 요소에 display 속성의 값을 flex 또는 inline-flex 라고 선언하면 해당 요소는 flex container가 되고 자식 요소는 자동으로 flex item이 되어 flex 박스로 렌더링하기 시작합니다.

#### flex container

flex container는 flex item의 면적, 방향, 정렬을 결정하는 컨테이너 입니다. flex container 요소에 display 속성의 값으로 inline-flex를 선언하면 인라인 수준의 flex container를 생성하고, flex 값을 선언하면 블럭 수준의 flex container를 생성합니다. inline-flex상태의 컨테이너는 inline-block 박스와 같은 형태로 표시합니다.

flex item

flex item은 컨테이너 내부에 형성된 free space(남거나 모자라는 공간, margin과 유사한 개념이지만 margin은 아님)를 팽창지수 또는 수축지수 값에 따라 형제들이 서로 나누어 갖습니다. flex container에 inline-flex 또는 flex 값을 선언하면 자식 요소들은 자동으로 **flex item**이 됩니다. flex item의 기본적인 스타일(User Agent 기본값)은 다음과 같습니다. 기본 스타일(flex-grow:0; flex-shrink:1; flex-basis:auto; flex-direction:row; flex-wrap:nowrap;)은 개발자가 변경할 수 있으며, 단축 속성(flex)을 사용할 때 값이 자동으로 재설정([flex item의 단축 속성 ‘flex’](http://naradesign.net/wp/2017/04/20/2363/#flex-shorthand) 참고)되기도 합니다.

여분의 free space가 있어도 폭이 저절로 늘어나지 않습니다.(flex-grow:0)

부모 박스 초과 시(이 때 flex container 내부에 음수 free space 발생) 자동으로 균등 수축(음수 free space 크기 ÷ flex item 개수)합니다.(flex-shrink:1)

콘텐츠 너비만큼 수축합니다.(flex-basis:auto)

행으로 배치 합니다.(flex-derection:row)

개행하지 않습니다.(flex-wrap:nowrap)

텍스트 노드도 익명 flex item이 됩니다. 공백은 flex item이 되지 않습니다.

그 밖에 이런 특징도 있습니다.

float 속성은 무시합니다.

position:absolute|fixed 속성이 부여되면 flex item에서 빠지게 됩니다.

flex item은 형제 또는 부모와 수직 또는 수평 margin을 중첩하지 않습니다.

브라우저마다 구현이 다를 수 있기 때문에 margin/padding의 값으로 상대 단위(%)를 사용하지 않는 것이 좋습니다.

flex container의 자손은 flex item이 되지 않습니다.

### 신축성(flexibility): flex item의 팽창 및 수축.

flex item은 flex-grow(팽창지수), flex-shrink(수축지수), flex-basis(기준 사이즈) 속성과 이것들의 단축 속성인 flex 속성으로 박스의 팽창, 수축, 기준 사이즈를 제어할 수 있습니다. W3C 명세는 flex 단축 속성(flex : none | [<flex-grow> <flex-shrink>? || <flex-basis>])으로 제어하는 것을 권장합니다. 단축 속성은 명시하지 않은 값을 일반적인 용도에 맞게 재설정하기 때문입니다. 단축 속성 선언 시 재설정되는 값에 대한 설명은 [flex item의 단축 속성 ‘flex’](http://naradesign.net/wp/2017/04/20/2363/#flex-shorthand)를 참고하세요. 참고: CSS 명세에서 바 ‘|‘는 분리된 값들 중 반드시 ‘하나’를 선언해야 한다는 의미입니다. 더블 바 ‘||‘는 분리된 값들 중 ‘하나 또는 그 이상’을 선언할 수 있다는 의미입니다. 물음표 ‘?‘는 ‘생략하거나 또는 한 번만 선언’해야 한다는 의미입니다.

flex item의 팽창을 제어하는 ‘**flex-grow**‘ 속성.

**Name**: flex-grow

**Value**: <number> // 음수 값은 유효하지 않음. 보통 ‘0‘ 또는 ‘1‘ 값을 선언.

**Initial**: 0 // 단축 속성 flex 사용 시 flex-grow 값을 생략하면 값은 초기 값은 ‘1‘으로 다시 설정 됨.

**Applies to**: flex items

flex-grow 속성의 기본값은 ‘0‘ 이기 때문에 flex item은 기본적으로 팽창하지 않습니다. flex-grow의 값이 ‘0‘이 아닌 경우에는 컨테이너 내부에 형성된 빈 공간(free space)을 팽창지수(flex-grow)에 따라 flex item에 반영하여 남은 공간을 채우게 됩니다. 공간이 남는 경우에는 flex-grow 속성에 의해 팽창하지만, 공간이 남지 않는 경우에는 아무리 값을 올려도 컨테이너의 너비 이상으로 팽창하지 않습니다. max-width 속성을 선언했다면 이 값을 초과하여 팽창하지 않습니다.

#### flex item의 수축을 제어하는 ‘flex-shrink‘ 속성.

**Name**: flex-shrink ( 기본적으로 flex-basis 값을 갖고 있어야 사용할 수 있다)

**Value**: <number> // 음수 값은 유효하지 않음. 보통 ‘0‘ 또는 ‘1‘ 값을 선언.

**Initial**: 1 // 단축 속성 flex 사용 시 flex-shrink 값을 생략해도 초기 값은 여전히 ‘1‘이 된다.

**Applies to**: flex items

flex-shrink 속성의 기본값은 ‘1‘ 이기 때문에 flex item은 기본적으로 수축합니다. flex-shrink 값이 ‘0‘인 경우 flex item의 너비가 컨테이너를 초과 해도 수축하지 않습니다. 한편 flex-shrink 값이 ‘0‘이 아닌 경우에는 flex item이 컨테이너를 초과했을 때 넘치는 공간(이 때 음수 free space 발생)의 크기를 기준으로 수축지수(flex-shrink)에 따라 수축하게 됩니다. 컨테이너를 초과해서 공간이 모자라는 경우에는 flex-shrink 속성에 의해 수축하지만, 공간이 모자라지 않는 경우에는 아무리 값을 올려도 수축하지 않습니다. min-width 속성을 선언했다면 이 값 미만으로 수축하지 않습니다.

#### flex item의 기준 사이즈를 제어하는 ‘flex-basis‘ 속성.

**Name**: flex-basis

**Value**: content | <width>

**Initial**: auto

**Applies to**: flex items

flex-basis 속성은 flex item 요소가 flex-grow 또는 flex-shrink 속성에 의해 팽창/수축하기 이전의 기준 크기를 명시하는 속성입니다. flex item에 width값을 명시하는 것은 flex-basis 값을 선언하는 것과 결과적으로 동일합니다. flex item의 팽창 또는 수축은 width값 또는 flex-basis에 선언한 값을 기준으로 free space(빈 공간) 값을 구하게 되고, free space 값은 팽창/수축 지수를 통해 flex item의 크기에 영향을 미치게 되므로 flex-basis(또는 width) 값을 가능하다면 명시적으로 선언하는 것이 좋습니다. 동일한 flex item에 width 값과 flex-basis 값을 동시 선언하는 경우 flex-basis 값은 width 값을 덮어 쓰기 때문에 코드를 간결하게 작성하려면 width 값을 선언하는 것보다 flex-basis 값을 선언하는 것을 권장합니다.

**flex item의 단축 속성 ‘flex‘.**

flex 속성은 flex-grow, flex-shrink, flex-basis 속성의 값을 하나의 속성값으로 작성할 수 있는 단축 속성입니다. flex item 요소에 flex와 관련 있는 아무런 속성도 선언하지 않은 경우 flex-grow:0; flex-shrink:1; flex-basis:auto; 다시 말하면 flex:0 1 auto; 상태가 됩니다. 세 가지 속성을 모두 선언할 수도 있지만 하나만 선언하고 다른 값을 생략할 수도 있는데요. 생략한 속성의 값은 자동으로 재설정 됩니다.

flex**단축 속성**은 아래와 같이 다양한 형태로 선언할 수 있습니다. 생략한 일부 속성의 값은 flex item에 아무런 속성도 선언하지 않았을 때의 값 flex:0 1 auto; 와도 다르고 flex:none; 을 선언했을 때의 값 flex:0 0 auto; 와도 다르다는 점에 유의하세요.

flex: none;  
// flex-grow:0; flex-shrink:0; flex-basis:auto; 상태가 된다.

flex: <flex-grow>  
// flex-shrink:1; flex-basis:0; 상태가 된다.

flex: <flex-basis>  
// flex-grow:1; flex-shrink:1; 상태가 된다.

flex: <flex-grow> <flex-shrink>  
// flex-basis:0; 상태가 된다.

flex: <flex-grow> <flex-basis>  
// flex-shrink:1; 상태가 된다.

flex: <flex-grow> <flex-shrink> <flex-basis>  
// 생략한 속성 없음.

[flex item의 ‘방향’을 제어하는 ‘flex-direction’.](http://naradesign.net/wp/2017/04/24/2440/#flex-direction)

[flex item의 ‘줄 바꿈’을 제어하는 ‘flex-wrap’.](http://naradesign.net/wp/2017/04/24/2440/#flex-wrap)

[flex item의 ‘뱡향과 줄 바꿈’을 제어하는 단축 속성 ‘flex-flow’.](http://naradesign.net/wp/2017/04/24/2440/#flex-flow)

[flex item의 ‘배치 순서’를 제어하는 ‘order’.](http://naradesign.net/wp/2017/04/24/2440/#order)

flex item의 ‘방향’을 제어하는 ‘flex-direction’.

flex-direction 속성은 flex item이 흐르는 방향(상하좌우)을 제어합니다.

* Name: flex-direction
* Value: row | row-reverse | column | column-reverse
* Initial: row
* Applies to: **flex container**

<h2>flex-direction: row | row-reverse</h2>  
<div class="a z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
<div class="b z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
  
<h2>flex-direction: column | column-reverse</h2>  
<div class="c z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
<div class="d z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>

Css :

.z{display:inline-flex;width:100px;height:100px;background:silver;border:5px solid black;color:white;vertical-align:top;}  
.z>\*{flex:1 1 0;}  
.z>:nth-child(1){background:red;}  
.z>:nth-child(2){background:green;}  
.z>:nth-child(3){background:blue;}  
.a{flex-direction:row;}  
.b{flex-direction:row-reverse;}  
.c{flex-direction:column;}  
.d{flex-direction:column-reverse;}

### flex item의 ‘줄 바꿈’을 제어하는 ‘flex-wrap’.

flex-wrap 속성은 flex item의 줄 바꿈 성질을 제어합니다.

* Name: flex-wrap
* Value: nowrap | wrap | wrap-reverse
* Initial: nowrap
* Applies to: **flex container**

<h2>flex-wrap: nowrap(default)</h2>  
<div class="a z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
  
<h2>flex-wrap: wrap | wrap-reverse</h2>  
<div class="b z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
<div class="c z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>

Css:

.z{display:inline-flex;width:100px;height:100px;border:5px solid black;color:white;vertical-align:top;}  
.z>\*{flex:0 0 50px;}  
.z>:nth-child(1){background:red;}  
.z>:nth-child(2){background:green;}  
.z>:nth-child(3){background:blue;}  
.a{flex-wrap:nowrap;}  
.b{flex-wrap:wrap;}  
.c{flex-wrap:wrap-reverse;}

### flex item의 ‘뱡향과 줄 바꿈’을 제어하는 단축 속성 ‘flex-flow’.

flex-flow 속성은 flex item의 flex-direction 속성과 flex-wrap 속성의 값을 한꺼번에 작성할 수 있는 단축 속성입니다.

* Name: flex-flow
* Value: <flex-direction> || <flex-wrap> // 둘 중 하나 또는 둘을 선언해야 한다.
* Initial: row nowrap
* Applies to: **flex container**

<h1>flex-flow:</h1>  
<p>'<em>회색 처리한 값</em>'은 기본값으로써 생략 가능하다는 것을 의미함.</p>  
<main>  
 <section>  
 <h2><em>row nowrap</em> <br> <em>row</em> <br> <em>nowrap</em></h2>  
 <div class="a z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2><em>row</em> wrap <br> wrap</h2>  
 <div class="b z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2><em>row</em> wrap-reverse <br> wrap-reverse</h2>  
 <div class="c z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>row-reverse <em>nowrap</em> <br> row-reverse</h2>  
 <div class="d z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>row-reverse wrap</h2>  
 <div class="e z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>row-revers wrap-reverse</h2>  
 <div class="f z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>column <em>nowrap</em> <br> column</h2>  
 <div class="g z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>column wrap</h2>  
 <div class="h z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>column wrap-reverse</h2>  
 <div class="i z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>column-reverse <em>nowrap</em> <br> column-reverse</h2>  
 <div class="j z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>column-reverse wrap</h2>  
 <div class="k z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
 <section>  
 <h2>column-reverse wrap-reverse</h2>  
 <div class="l z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
 <div>4</div>  
 </div>  
 </section>  
</main>

Css:

.z{display:inline-flex;width:99px;height:99px;outline:5px solid rgba(0,0,0,.5);color:white;vertical-align:top;}  
.z>\*{flex:0 0 33px;}  
.z>:nth-child(1){background:orange;}  
.z>:nth-child(2){background:red;}  
.z>:nth-child(3){background:green;}  
.z>:nth-child(4){background:blue;}  
  
.a{flex-flow:row nowrap;}  
.b{flex-flow:row wrap;}  
.c{flex-flow:row wrap-reverse;}  
  
.d{flex-flow:row-reverse nowrap;}  
.e{flex-flow:row-reverse wrap;}  
.f{flex-flow:row-reverse wrap-reverse;}  
  
.g{flex-flow:column nowrap;}  
.h{flex-flow:column wrap;}  
.i{flex-flow:column wrap-reverse;}  
  
.j{flex-flow:column-reverse nowrap;}  
.k{flex-flow:column-reverse wrap;}  
.l{flex-flow:column-reverse wrap-reverse;}

### flex item의 ‘배치 순서’를 제어하는 ‘order’.

order 속성은 flex item의 배치 순서를 제어하는 속성입니다. 기본값은 ‘0‘이며 flex-direction 속성의 방향값(row, row-reverse, column, column-reverse)을 기준으로 낮은 숫자를 먼저 배치하고 높은 숫자를 나중에 배치합니다.

* Name: order
* Value: <integer> // ‘0, 양의 정수, 음의 정수’를 사용할 수 있음.
* Initial: 0
* Applies to: **flex items** and absolutely-positioned children of flex containers // 절대값으로 처리된 flex-item에도 적용 가능하다고 하는데 어떤 상황에서 쓰이는지 정확히 알 수 없음.

<h1>order:</h1>  
<p>검정색 배경으로 처리한 두 번째 flex-item의 순서를 변경합니다.</p>  
<h2>.a>:nth-child(2){ order:0 } // default</h2>  
<div class="a z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
  
<h2>.b>:nth-child(2){ order:1 }</h2>  
<div class="b z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
  
<h2>.c>:nth-child(2){ order:2 } // 다른 아이템의 order가 모두 '0'이므로 결과는 '1'과 같음.</h2>  
<div class="c z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
  
<h2>.d>:nth-child(2){ order:-1 }</h2>  
<div class="d z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>  
  
<h2>.e>:nth-child(2){ order:-2 } // 다른 아이템의 order가 모두 '0'이므로 결과는 '-1'과 같음.</h2>  
<div class="e z">  
 <div>1</div>  
 <div>2</div>  
 <div>3</div>  
</div>

Css:

.z{display:flex;outline:5px solid rgba(0,0,0,.5);color:white;margin-bottom:2em;}  
.z>\*{flex:1;}  
.z>:nth-child(1){background:rgba(255,0,0,.66);}  
.z>:nth-child(2){background:black;}  
.z>:nth-child(3){background:rgba(0,0,255,.66);}  
  
.a>:nth-child(2){order:0;}  
.b>:nth-child(2){order:1;}  
.c>:nth-child(2){order:2;}  
.d>:nth-child(2){order:-1;}  
.e>:nth-child(2){order:-2;}