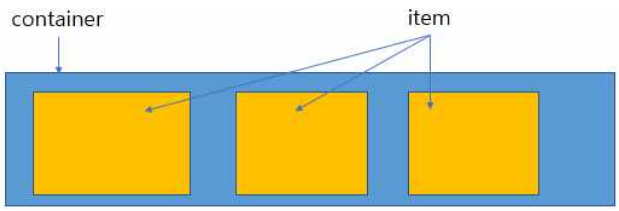
flex, flexbox Responsive Web을 위한 css layout 속성

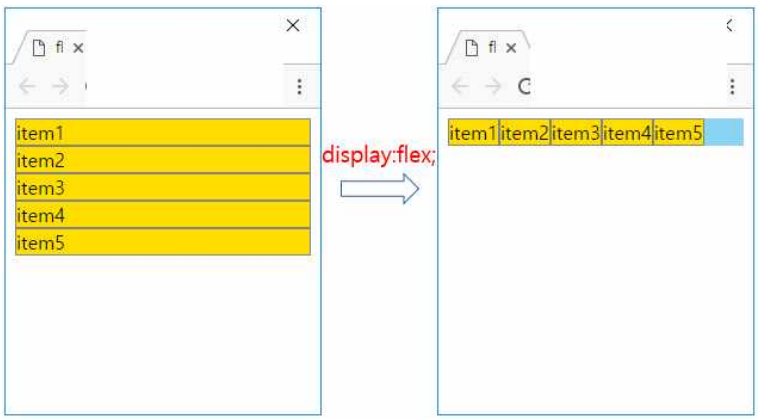
기존의 Web Page를 만들 때 사용하던 올드한 layout 제작방법을 개선하고자 flexbox가 생겨났다. flex는 flexible의 준말로 유동적인 layout을 손쉽게 만들 수 있다는 의미를 가진다.
기본 컨셉은 아주 명확하기 때문에 기존의 layout을 만드는데 사용하던 table 태그나 position 또는 float 속성을 사용하는 방법의 복잡함을 고수할 필요가 없어졌다. 다만 생각보다 다양한 속성들이 있어서 정리할 필요는 있다. 이 속성을 모두 사용할 필요는 없으므로 대략적인 이해만 해 둔 후에 필요할 때마다 참고하여 layout을 만들면 될 것이다.

☑ flexbox의 기본 개념

container와 item들이 parent와 child 관계일 때, container에 display: flex; 속성을 줌으로서 다음과 같은 모습의 flexbox layout으로 바뀐다.



다음은 display: flex; 속성을 줄 때 Browser 상에서 layout이 어떻게 바뀌는지를 보여주는 그림이다.



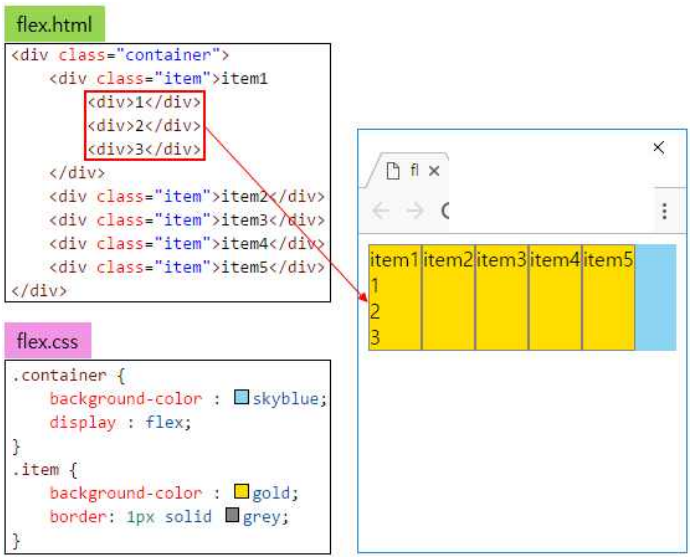
이렇게 flexbox layout으로 바뀐 후 부터는 flex container와 flex item 각각에 flex에 관련된 속성을 줄 수 있는데 각각을 구분하여 각각에 어떤 속성을 적용할 수 있고 어떤 효과가 있는지 학습하도록 한다.

☑ flex container 속성들 정리

❖ display

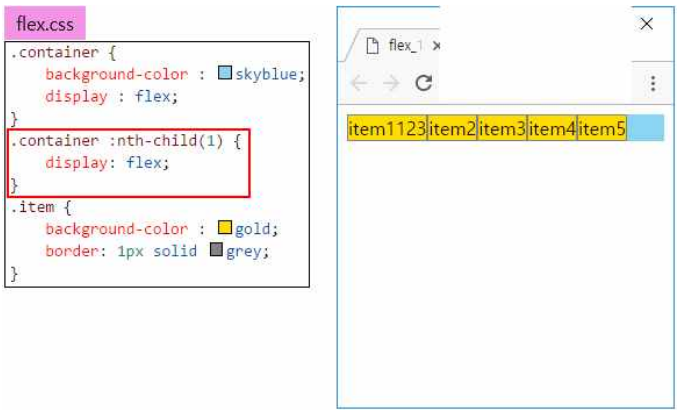
```
.container {
    display: flex;  /* inline-flex */
}
```

앞에서 설명한 속성으로 flex layout을 설정하기 위해 기본적으로 있어야 하는 속성이다. 속성 적용 후에 child element(자식 요소)들의 배치가 inline화(한줄에 배치) 되었음을 기억하자. 주의할 점은 이 속성은 직계(direct) 자손에게만 적용된다는 것이다.



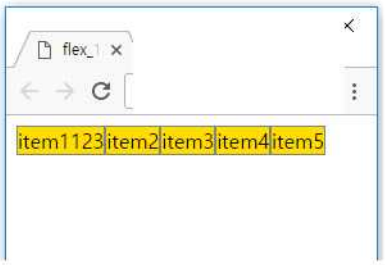
container 클래스의 직계 자손들은 flexbox layout이 적용되어 가로(inline)로 한줄 안에 배치되었으나 container 클래스의 첫 번째 직계 자손의 자손은 적용되지 않고 세로로 배치되었다. (div 태그는 display: block; 이 기본 속성이기 때문이다.)

따라서 다음 코드를 추가하여 flexbox 모든 자식에게 layout을 적용시킬 수 있다.



☞ display : inline-flex; 속성을 주면 container 영역이 item에 맞게 줄어든다.

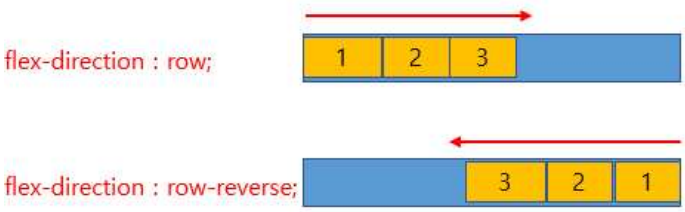
```
.container {
    background-color : skyblue;
    display: inline-flex;
}
.container :nth-child(1) {
    display: flex;
}
.item {
    background-color : gold;
    border: 1px solid grey;
}
```



❖ flex-direction

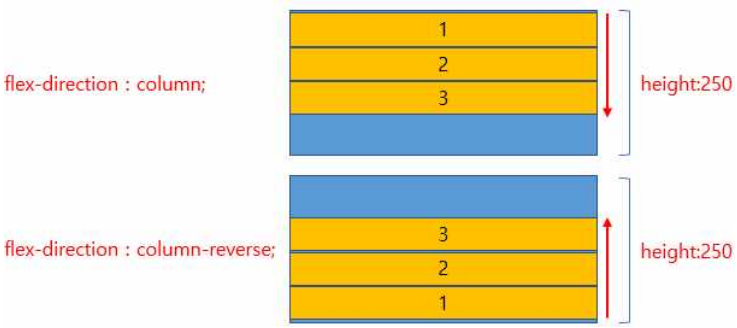
```
.container {
    display: flex;
    flex-direction: row; /* row-reverse, column, column-reverse */
}
```

container 안에서 item들의 정렬과 배치 방향을 설정하는 속성이다.



flex-direction이 row면 container 내부에서 왼쪽 정렬, 왼쪽부터 아이템이 행방향으로 순서대로 배치된다. flex-direction이 row-reverse면 container 내부에서 오른쪽 정렬, 오른쪽부터 아이템이 행방향으로 순서대로 배치된다.

container 클래스에 height 속성을 다음과 같이 넉넉히 잡고 다음 속성들을 입력해 본다.



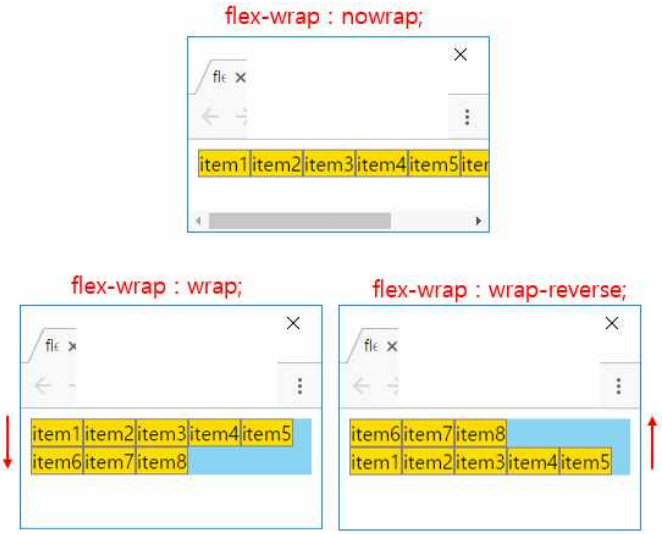
flex-direction이 column이면 아이템들은 각각 한줄을 모두 차지하며 block 속성값을 갖는 것 처럼 열 방향으로 배치된다. flex-direction이 column-reverse면 container의 아래쪽 부터 위쪽으로 채워지면서

아이템 배치순서가 아래에서 위쪽으로 바뀐다.
※ 참고 : container를 display : inline-flex; 속성으로 바꾸면 위 속성들이 어떻게 동작하는지 확인하도록 하자.

❖ flex-wrap

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

item들의 너비의 합이 container의 너비(현재 Browser 창의 너비)를 초과할 때 어떻게 처리할 지를 결정하는 속성이다. 그냥 줄바꿈 속성이라고 생각해도 된다.



☞ nowrap 은 기본값으로 Browser의 너비를 초과해도 상관없이 아이템들이 한 줄로 표시된다.
☞ wrap 은 Browser의 너비를 초과한 아이템들을 줄 바꿈을 하여 다음 줄로 넘긴다.
☞ wrap-reverse는 wrap과 같지만 아래에서 위쪽으로 배치한다.

반응형 웹에서 PC와 같은 환경에서는 메뉴가 사이드에 표시되지만 모바일 환경에서는 아래로 길게 늘어뜨려지는 경우를 본 적이 있을 것이다. 이런 것은 이 속성으로 아주 간단히 표현할 수 있다.

❖ flex-flow

```
.container {
    display: flex;
    flex-flow: row wrap;  /* flex-direction과 flex-wrap의 조합 */
}
```

flex-flow는 바로 앞의 두 속성을 같이 설정할 수 있는 약식 표현 속성이다. 이런 약식표현으로 border가 있음을 우리는 알고 있으므로 쉽게 이해할 수 있다.

❖ justify-content

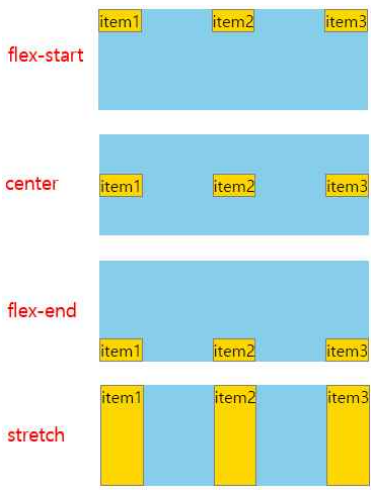
```
.container {
    display: flex;
    justify-content: flex-start;        /* flex-end, center, space-between, space-around */
}
```

item과 container 간에 수평방향으로 여백을 두는 방식을 지정한다.

이 속성은 container의 display 속성이 inline-flex 라면 소용없는 속성이다. 왜냐면 inline-flex 속성을 주면 item과 container 간에 여백이 없어지기 때문이다. 어렵지 않은 속성이므로 위 그림으로 설명을 대신한다.

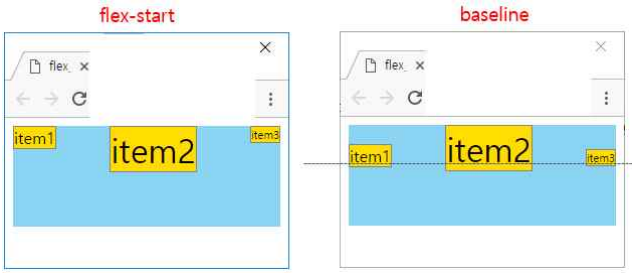
❖ align-items

```
.container {
    display: flex;
    height: 100px;
    /* flex-end, center, baseline, stretch */
    align-items: flex-start;
}
```



justify-content 속성이 수평 방향으로 여백을 주는 방식을 설정하는 속성이라면 align-items는 수직방향으로 item과 container 간에 여백을 주는 방식을 설정한다. 이 속성의 정확한 이해를 위해서 height 속성을 넉넉히 준 후 실험하도록 한다.

이 속성들이 어떻게 동작하는지는 그림만으로도 충분히 이해할 수 있을 것이다.
속성값 baseline은 아래 그림을 보자.

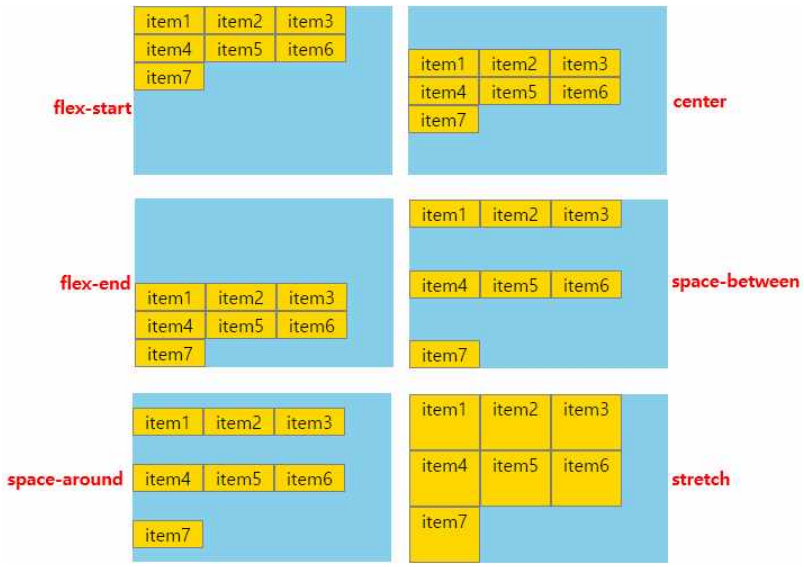


속성값이 baseline을 이해하기 위해 font-size를 달리 해 보았다. 보시다시피 폰트의 baseline을 기준으로 정렬된다. 왼쪽에 flex-start가 정렬되는 방식과 비교하면 쉽게 이해할 수 있을 것이다.

❖ align-content

```
.container {
    display : flex;
    flex-wrap : wrap;
    /* flex-end, center, space-between, space-around, stretch */
    align-content : flex-start;
}
```

아이템들을 한 줄에 다 표시할 수 없어서 다음줄로 넘김이 발생했을 때 줄 사이에 여백을 결정하는 방식을 설정한다.
따라서 이 속성은 flex-wrap: wrap; 속성이 설정되어 있어야 제대로 동작한다.



justify-content가 수평방향으로 여백을 설정한다면 align-content는 수직방향으로 여백을 설정한다. align-item이 item과 container 간에 여백을 설정하는 것이라면 align-content는 수직방향으로 item들 사이의 여백과 item과 container 사이의 여백을 설정한다. 여백이 정해지는 방향만 다르지 방식은 동일하다.
살펴본 속성들이 많기는 하지만 꽤 직관적이기 때문에 이해하기는 쉽다. 필요할 때마다 그때 그때 참조해서 사용하면 된다.

☑ flex item 속성들 정리

앞에서는 container에 설정하는 속성들을 살펴보았다. 이번에는 item에 설정하는 속성들을 살펴볼 것이다.

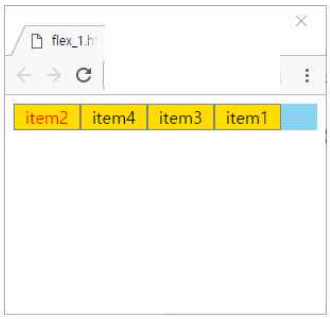
❖ order

```
.item {
    order : 1; /* 속성값은 숫자 */
}
```

order 속성은 아이템이 배치될 순서를 지정한다. 속성값은 숫자며 값이 작을수록 먼저 배치된다.

order 값이 -2로 가장 작은 item2가 제일 먼저 배치되고 order 값이 100으로 가장 큰 item1이 가장 마지막에 배치되었다.

```
<html>
<head>
    <style>
        .container {
            background-color : skyblue;
            display : flex;
        }
        .item {
            background-color : gold;
            padding: 1px 10px 1px 10px;
            border: 1px solid grey;
        }
        .container :nth-child(1) { order : 100;}
        .container :nth-child(2) { order : -2; color: red; }
        .container :nth-child(3) { order : 4; }
        .container :nth-child(4) { order : 0; }
    </style>
</head>
<body>
    <div class="container">
        <div class="item">item1</div>
        <div class="item">item2</div>
        <div class="item">item3</div>
        <div class="item">item4</div>
    </div>
</body>
</html>
```

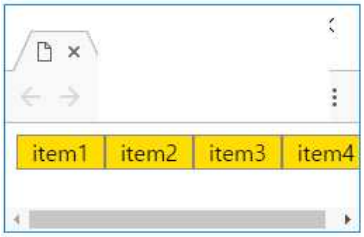


❖ flex-grow

```
.item {
    flex-grow: 1;        /* 속성값은 숫자 */
}
```

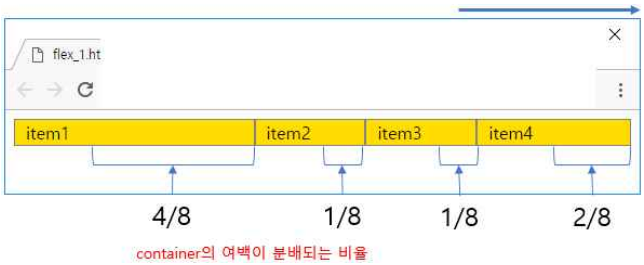
flex-grow는 container에 여분의 여백이 있을 때 동작하는 속성이다. 여분의 여백이 있다면 flex-grow에 설정된 비율만큼 분배되도록 동작된다. 따라서 Browser를 가로방향으로 늘리게 되면 flex-grow에 설정된 비율만큼씩 증가되는 것을 볼 수 있을 것이다.

```
.container :nth-child(1) { flex-grow : 4; }
.container :nth-child(2) { flex-grow : 1; }
.container :nth-child(3) { flex-grow : 1; }
.container :nth-child(4) { flex-grow : 2; }
```



4개의 item에 flex-grow를 설정해 놨다. 현재 Browser의 너비가 여분의 여유공간이 없으므로 item들에 flex-grow가 적용되지 않은 상태다. Browser 창을 가로방향으로 서서히 늘려보면 container에 여백이

생기는 순간부터 이 여백이 각각의 item에게 flex-grow에 설정한 비율만큼씩 분배되는 것을 알 수 있다.



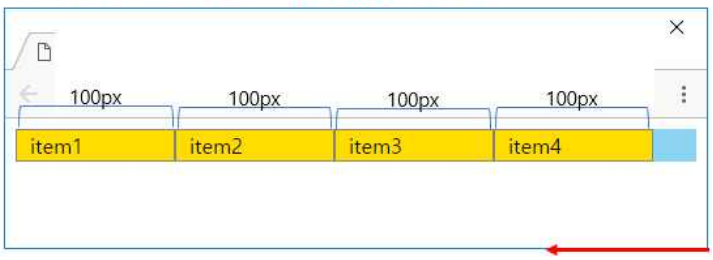
하지만 늘어난 Browser를 다시 줄여보면 줄어드는 비율도 위와 같다. 따라서 flex-grow는 늘어나는 비율 뿐 아니라 줄어드는 비율이기도 하다.

❖ flex-basis

```
.item {
    flex-basis: 100px;
}
```

flex-basis는 item의 기본 너비를 설정한다.
그런데 이 속성을 정하면 디폴트로 flex-shrink 속성이 붙는다. 그리고 flex-shrink는 item마다 줄어드는 비율을 설정하는 속성이다. flex-basis에서 설정한 너비보다 더 줄이면 flex-shrink 속성에 설정한 비율로 줄어드는 것을 볼 수 있다.

```
.container :nth-child(1) { flex-basis: 100px; }
.container :nth-child(2) {  flex-basis: 100px;}
.container :nth-child(3) {flex-basis: 100px; }
.container :nth-child(4) {  flex-basis: 100px;}
```



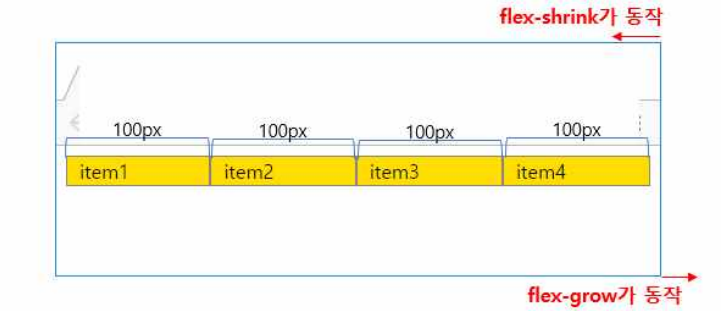
위 그림에서 Browser의 창을 줄이다가 container와 item의 여백이 없어진 이후에도 창을 더 줄이면 item들은 flex-basis에서 설정한 100px 이하로 줄어들 것이다. 이 때부터 flex-shrink에서 설정한 비율만큼 줄어드는데 현재 디폴트로 네 개의 아이템의 flex-shrink의 값은 동일하다. 즉 동일한 비율로 줄어드는 것을 볼 수 있다. (각각 1/4, 1/4, 1/4, 1/4 의 비율로)
주의할 점은 flex-basis를 설정해도 flex-grow는 디폴트로 주어지지 않는다는 것이다. 그래서 위 그림에서 각 item들이 100px 이상 늘어나지 않는 것이다. 만약 flex-grow를 주면 어떻게 되는지는 아래 flex-shrink 속성의 설명을 참고하도록 하자.

❖ flex-shrink

```
.item {
    flex-shrink: 4;
}
```

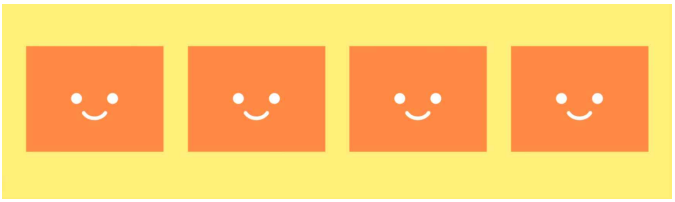
flex-shrink는 item이 줄어드는 비율을 설정한다. flex-basis를 설정했다면 이 값을 경계로 item의 너비가 flex-basis에서 설정한 값보다 클 때는 flex-grow가 동작하고 item의 너비가 flex-basis에서 설정한 값 보다 작을 때는 flex-shrink가 동작한다.

```
.container :nth-child(1) { flex-basis: 100px;flex-grow : 4; flex-shrink: 4;}
.container :nth-child(2) {  flex-basis: 100px;flex-grow : 3;flex-shrink: 3;}
.container :nth-child(3) {flex-basis: 100px; flex-grow : 2; flex-shrink: 2;}
.container :nth-child(4) {  flex-basis: 100px;flex-grow : 1;flex-shrink: 1;}
```



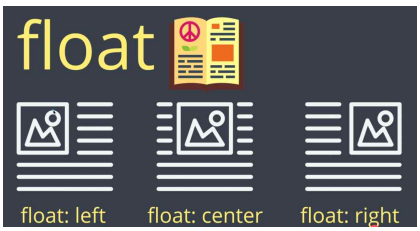
따라서 위와 같이 설정했다면 flex-grow 속성으로 인해 Browser를 아무리 늘려도 container와 item 사이에 여분의 여백은 생기지 않는다.

// Position, float, Table => 너무 복잡하고 시간도 많이 소요되고 힘들다.
// 할 수 없는 일들



1) Box 안에 item들을 수직적으로 가운데 정렬하기
2) item들의 사이즈에 상관없이 동일한 간격으로 동일한 사이즈로 box 안에 배치하는 것
3) box를 동일한 높이로 놓아 두는 것들이 조금은 까다로웠다.

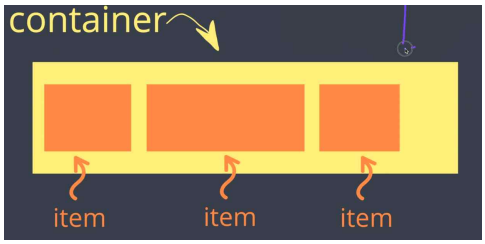
☞ Flexbox로 이러한 문제를 손쉽게 해결할 수 있다.



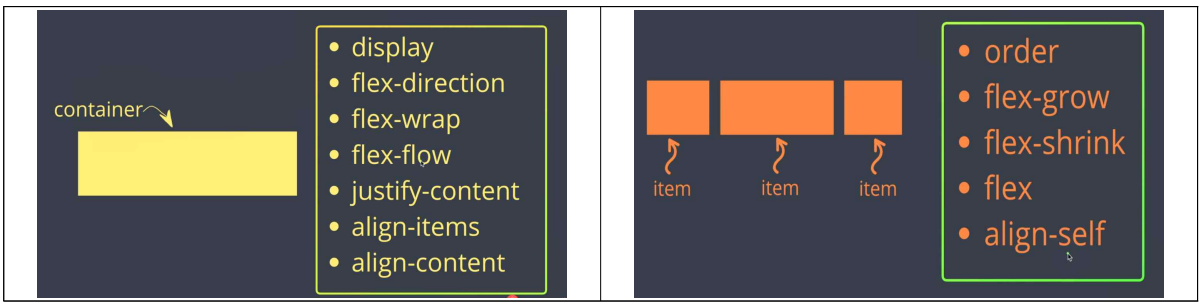
☞ float의 원래의 목적은 image와 text들을 어떻게 배치할 것인지에 대하여 정의하기 위하여 나타난 것이다.
☞ float: left는 image가 왼쪽에 배치되고 text들이 image를 감싸면서 배치될 수 있도록 해주는 것이 발 float 이다.
☞ 예전에는 CSS에 이러한 layout을 할 수 있는 기능이 없었기 때문에 float을 이용하여 box를 왼쪽이나 중앙이나 오른쪽에 배치하곤 하였다. 이것은 float의 순수 목적에는 어긋나는 것이었다.
☞ 이제는 flexbox가 있기 때문에 이 float은 원래의 목적인 text와 image들을 배치하는 용도로 사용되게 되었다.

Flexbox는 2가지만 이해하면 된다.

① Flexbox는 container의 box에 적용되는 속성값들이 존재한다. 또한 각각의 item들에 적용할 수 있는 속성값들이 존재한다.

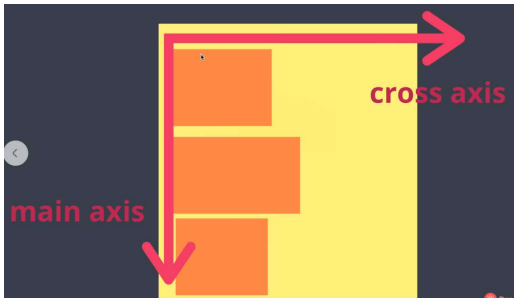


☞ container와 item 들에 꾸며줄 수 있는 속성값들은 다음과 같은 것들이 있다.



② Flexbox에는 중심축(main axis)과 반대축(cross axis)이 있다. 만약 수직축이 중심축이면, 수평축은 반대축이 된다. 아래 그림과 같은 경우에는 item들이 왼쪽에서 오른쪽으로 정렬이 되기 때문에 수평추이 중심축이 되고, 반대로 수직축이 반대축이 된다.

반대로 item들이 위에서 아래로 정렬될 경우, 수직으로 가는 축이 중심축이 되고 반대로 수평축이 반대축이 된다.



따라서 중심축을 수평에 둘지 수직에 둘지에 따라 반대축이 바뀌는 것을 볼 수가 있다.

[실습 예제]

[index01.html]

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Demo</title>
  <link rel="stylesheet" href="style01.css">
</head>
<body>
  <!-- div.container>div.item.item${$}*10 -->
  <div class="container">
    <div class="item item01">1</div>
    <div class="item item02">2</div>
    <div class="item item03">3</div>
    <div class="item item04">4</div>
    <div class="item item05">5</div>
    <div class="item item06">6</div>
    <div class="item item07">7</div>
    <div class="item item08">8</div>
    <div class="item item09">9</div>
    <div class="item item10">10</div>
  </div>
</body>
</html>
```

[style01.css]

```
/* body, html {
  height: 100%;
} */

.container {
  background: beige;
  height: 100vh;
}

.item01 {
  background: #e53935;
}
.item02 {
  background: #8e24aa;
```

```
}
.item03 {
  background: #3949ab;
}
.item04 {
  background: #039be5;
}
.item05 {
  background: #43a047;
}
.item06 {
  background: #c0ca33;
}
.item07 {
  background: #ffb300;
}
.item08 {
  background: #f4511e;
}
.item09 {
  background: #6d4c41;
}
.item10 {
  background: #546e7a;
}
```

100vh(viewport height)을 사용하지 않고 100%을 사용하면, height이 사라진다. item에 맞게 지금 높이가 지정되어 있는데, %는 container가 들어있는 parent의 높이의 100%를 채우겠다는 의미이다.
container의 parent는 body 이다.
body tag 안에 height: 100%; 를 지정하여도 나타나지 않는다. 그것은 body의 parent인 html이 100%가 아니기 때문이다. 따라서 다음과 같이 지정하면 height 이 나타난다.

```
body, html {
    height: 100%;
}
```

이제 parent와 상관없이 item을 보이는 viewport height을 100% 다 사용하겠다고 하는 것이 height: 100vh; 이다.

COLOR TOOL => https://material.io/resources/color/#!/?view.left=0&view.right=0

먼저 box를 만들어 보자.

```
.item {
    width: 40px;
    height: 40px;
}
```

Chrome Browser에서 F12로 개발 Tool을 이요하여 item을 보면, div 옆에 margin이 들어가 있기 때문에 더 이상 item들이 안 들어오는 것을 확인할 수가 있다.

Flexbox에는 container에 들어가는 속성값과 각각 item에 들어가는 속성값으로 나누어져 있다.
먼저 container에 들어가는 속성값을 학습하도록 한다.

☞ flexbox로 만들고 싶다면 먼저 container에게 display: flex; 를 입력하면 된다. 그러면 item들이 자동적으로 왼쪽에서 오른쪽으로 정렬이 된다.

☞ flex-direction: row;
기본값은 row 이다. row는 왼쪽에서 오른쪽으로 가는 방향이다. row-reverse; 는 오른쪽에서 왼쪽으로 옮겨가게 된다. 여기서 중심축은 수평축이다.

☞ flex-direction: column;
column은 중심축이 수직축이 되고 위에서 아래로 내려가게 된다. column-reverse; 는 아래에서 위로 올라가게 된다.

☞ flex-wrap: nowrap;
기본값은 nowrap 이다. 만약 html에서 item들이 더 많아지게 되면 한줄에 꽉차게 붙여있게 된다. 아무리 줄여도 한 줄에 붙어있게 된다. 이것은 wraping을 하지 않겠다고 기본값으로 지정이 되어있기 때문이다.
만약 wrap을 바꾸게 되면 item들이 한줄에 꽉차게 되면 자동적으로 다음 라인으로 넘어가게 된다.
wrap-reverse; 를 하면 반대로 waping이 이루어진다.

item 에서 border: 1px solid black; 이라고 하면, 원래는 border-width, border-style, border-color 등 각각 따로 속성값들이 있는데 이렇게 한줄에 편하게 사용할 수가 있다.

☞ flex-flow: column nowrap; 도 위의 두 가지 flex-direction, flex-wrap들을 합한 것으로 사용할 수가 있다.

지금까지는 flexbox로 보여주고, 전체적인 방향에서 수펴이 중심축인지 수평이 중심축인지를 정하고, 한줄에 가득차면 다음줄로 넘어가게 할 것인지 아닌지를 학습하였다.

다음으로는 item들을 어떻게 배치할 것인지에 학습하도록 한다.

☞ justify-content: flex-start;
이것은 item들을 어떻게 배치할 것인지를 결정해 준다. 기본값은 flex-start; 이다. 처음부터 왼쪽에서 오른쪽으로 또는 수직축이 중심축이라면 위에서 아래로 나타나게 된다.
flex-end; 는 오른쪽 축으로 item들을 배치하는 것이다. 여기서 flex-end; 는 item 들의 순서는 유지하되 오른쪽으로 배치를 하고, flex-direction: column; 이게 되면, 밑에서 아래쪽으로 배치하게 된다.

여기서 조금 다른점은 flex-direction: column-reverse; 를 사용하게 되면 item들이 1, 2, 3, 4 와 같이 아래에서 부터 쌓여졌다면 flex-end; 는 item의 열은 그대로 유지한 채로 item 만 밑으로 내려주는 것을 확인할 수가 있다. justify-content: center; 로 지정하게 되면 item들을 center로 맞추어서 놓는다.
justify-content: space-around; 는 box를 둘러싸게 space를 넣어주는 것이다. 왼쪽에 있는 item과 오른쪽 끝에 있는 item은 한 줄만 들어가기 때문에 space가 작고 가운데 있는 item 들은 조금더 큰 space가 들어가게 된다.
justify-content: space-evenly; 는 item들이 똑같은 간격으로 space를 넣어주는 것이다.
justify-content: space-between; 은 item을 왼쪽과 오른쪽이 화면에 딱 맞게 배치하고 중간에만 item들을 넣어주는 것이다.
justify-content: 는 중심축에서 item을 어떻게 배치하는가를 결정하는 것이다.

이제는 반대축에서 item 들을 배치하는 속성값이 있는데 align-items: 이다.

☞ align-items: center;
이 box들을 수직적으로 중심에 놓고 싶다고 하면 align-items: center; 를 사용하면 된다.
justify-content: 는 중시묵에서 item 들을 배치하고, align-items: 는 반대축에서 item들을 어떻게 할 것인지를 결정하게 된다.
align-items: 에는 baseline 이라는 것이 있다. item01에 padding 있어서 text의 위치가 바뀌었으면,
align-items: baseline; 이라고 하면 text가 모두 다 동일하게 균등하게 보여질 수 있도록 item들을 배치할 수 있다.

```
.item01 {
  background: #e53935;
  padding: 20px;
}
```

☞ align-content: space-between;
justify-content: 와 비슷하지만 justify-content: 는 중심축에서 item들을 배치한다면, align-content: 는 반대축의 item들을 지정하게 된다.

여기에서 item들을 flex-wrap: wrap; 으로 설정하고, 많은 itemm들을 넣어보도록 한다. 기본적으로 item들이 한줄 한줄씩 표시가 된다. 만약 align-content: space-between; 으로 설정을 하면, 위에와 아

래는 딱 붙어있으면서 중간에 space가 들어가게 된다. 여기에는 justify-content: 속성값들을 모두 사용할 수가 있다.
align-content: center; 라고 지정하게 되면 중간에 item들이 모여서 배치가 된다.

그러나 여기에서 MDN Browser 호환성을 보면 새로나온 left and right, safe and unsafe, start and end 등은 아직 많은 Browser에서 지원이 안되는 것을 볼 수가 있다.
따라서 Flexbox나 이러한 기본적인 값들은 많이 사용되고 있지만, 새로운 속성값들은 아직 지원이 안되는 경우가 있기 때문에 이러한 것들이 지원이 되는지 안되는지 확인하면서 사용하여야 한다.
그러나 Browser가 갱신되는 속도가 빠르기 때문에 이러한 제한을 두지 말고 새로운 것들을 많이 사용하면서 caniuse.com 이나 MDN Site에서 이러한 새로운 속성값들이 baseline에 지원이 되는지 혹은 지원이 되지 않는지 확인을 하면서 사용할 것을 권한다.

이러한 속성값들이 기억이 나지 않으면 찾아가면서 사용하면 된다. 여기서 중요한 점은 큰 개념을 이해하고 있느냐 없느냐 이다. 이 개념을 이해하고 있을 때 찾는 속도와 이해를 하지 못하고 찾는 속도(정보를 획득할 수 있는 속도)는 엄청난 차이가 있다.

자세한 내용들은 다음 사이트를 참고하도록 한다.

https://css-tricks.com/snippets/css/a-guide-to-flexbox/
https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
https://developer.mozilla.org/en-US/docs/Web/CSS/float

https://flexboxfroggy.com/#ko

이제는 item에 들어가는 속성값에 대하여 학습하도록 한다.

[index02.html]

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Flexbox Demo</title>
  <link rel="stylesheet" href="style02.css">
</head>
<body>
  <!-- div.container>div.item.item${$}*10 -->
  <div class="container">
    <div class="item item01">1</div>
    <div class="item item02">2</div>
    <div class="item item03">3</div>
  </div>
</body>
</html>
```

[style02.css]

```
.container {
  padding-top: 100px;
  background: beige;
  height: 100vh;
  display: flex;
}

.item {
  width: 40px;
  height: 40px;
  border: 1px solid black;
}

.item01 {
  background: #e53935;
}
.item02 {
  background: #8e24aa;
}
.item03 {
  background: #3949ab;
}
```

container 안에 총 3개의 item만 남겨 놓고, container에는 다음과 같이 작성하도록 한다.

```
.container {
  padding-top: 100px;
  background: beige;
  height: 100vh;
  display: flex;
}
```

기본적으로 flex이기 때문에 왼쪽에서 오른쪽으로 row 값으로 정렬이 된다.
item01에 order: 를 지정할 수가 있다. 기본값은 모두 다 0;으로 되어 있기 때문에 item이 html에서 나온 상태 그대로 정렬이 된다. 이것을 order: 를 이용하여 변경할 수가 있다.

```
.item01 {
  background: #e53935;
  order: 2;
}
.item02 {
  background: #8e24aa;
  order: 1;
}
.item03 {
  background: #3949ab;
  order: 3;
}
```

이렇게 item들의 order를 변경하고 싶을 때 사용할 수 있다.

☞ flex-grow: 1;
flex-grow: 를 사용하지 않으면 item은 원래 40px 씩 유지하고 있다. container가 아무리 켜져도 계속 40px 씩 유지하다가 container가 너무 작아지면 어쩔 수 없이 한 줄에 다 꽉차있어야 되기 때문에 조금씩 작아진다. 그러나 flex-grow: 1; 를 사용하면 container를 꽉 채울려고 item들이 늘어나게 된다.
기본값은 0 이지만, item들이 이 상자에 채울려는 노력을 하지 않는다.

```
.item01 {
  background: #e53935;
  flex-grow: 1;
}
.item02 {
  background: #8e24aa;
}
.item03 {
  background: #3949ab;
}
```

첫번째 item01만 flex-grow: 1; 로 지정하였기 때문에 두번째나 세번째는 자신의 고유한 사이즈를 유지

하고 있다. item02, item03 도 동일하게 flex-grow: 1; 로 지정하면 item들이 골고루 크기에 따라 늘어나거나 줄어들게 된다.
item01을 flex-grow: 2; 로 지정하게 되면, item02나 item03 보다 2배로 크기게 된다.

☞ flex-shrink: 0;
flex-shrink: 는 container가 점점 작아졌을 때 어떻게 행동하느냐를 지정하는 것이다. 기본값은 0; 이다. item 모두 flex-shrink: 1; 으로 지정하면, item들이 점점 줄어들게 되면 어떨게 줄어드는 지가 표기가 된다. 반대로 item01를 flex-shrink: 2; 로 지정하면, 줄어들 때 2배로 작게 줄어들어 item02 나 item03 보다 2배 많이 줄어들게 된다.

이렇게 flex-grow: 와 flex-shrink: 는 container 사이즈가 변경되었을 때 item들이 얼마나 어떻게 더 줄어들고 늘어나야 되는지를 정의하는 것이다.

☞ flex-basis: auto;
flex-basis: 는 item들이 공간을 얼마나 차지해야 되는지 조금 더 세부적으로 명시할 수 있게 도와주는 속성값이다. 기본값은 auto 이다. auto로 하게 되면, grow 나 shrink는 지정된 것에 맞추어서 item들이 변형되지만, grow 나 shrink를 사용하지 않고 item01에 flex-basis: 60%; 로 지정하면 다른 item들에 비하여 60%를 차지하고 item02에 flex-basis: 30%;, item03은 flex-basis: 10%; 로 지정하면 커질 때도 작아질 때도 container의 width에 따라서 60%, 30%, 10% 차지하게 된다.

☞ align-self: center;
container level에서는 justify-content:, align-items:, align-content: 를 통해서 item들을 골고루 배치할 수 있다면, align-self: 를 이용해서 item 별로 item들을 정렬할 수가 있다.
item01 하나만 맞추고 싶다면 align-self: center; 로 할 수 있고, container에 지정된 것을 벗어나서 item 하나만 특별히 배치하고 싶다면 align-self: 를 사용하면 된다.