CSS Reference

made by Rex

< 선택자 >

전체 선택자 : 문서 내에 모든 태그를 선택한다. (html, head, title, body, style 태그까지 모두 선택함.)

```
* { border: 1px solid black; }
```

태그 선택자 : 문서 내에 해당 모든 태그를 선택한다.

```
tag { color: red; }
tag1, tag2, tag3, tag4 { color: red; }
==> 여러 개의 선택자를 컴마(,)로 구분하여 동시에 적용 할 수 있다.
```

아이디(#), 클래스(.) 선택자 : id 속성값으로 선택한다.

```
#id { color: red; }
.button { display: inline-block; text-align: center; line-height: 50px; }
```

tag 가 가지고 있는 id 또는 class 를 함께 선택하기

```
a.abc { background: green; }  ==>  a태그이면서 class="abc" 인 태그를 선택  (동일: a[class=abc] )
input#pwd { color: red; }     ==>  input태그이면서 id="pwd" 인 태그를 선택  (동일: input[id=pwd] )
```

속성 선택자 : 해당 태그의 해당 속성과 값을 가진 태그를 선택한다. (속성은 1개~여러개를 쓸 수 있다.)

```
tag[key=value][key=value] { color: red; }
input[type=text][name=pwd] { background: orange; }
```

문자열 속성 선택자 : 태그에 지정한 속성의 특정 문자열을 확인함

```
tag[속성~=값] { color: red; }  ==> 속성 안의 값이 특정값을 단어로 포함하는 태그를 선택 (-을 하나로)
tag[속성|=값] { color: red; }  ==> 속성 안의 값이 특정값을 단어로 포함하는 태그를 선택 (-기준으로 각각)
tag[속성^=값] { color: red; }  ==> 속성 안의 값이 특정값으로 시작하는 태그를 선택
tag[속성$=값] { color: red; }  ==> 속성 안의 값이 특정값으로 끝나는 태그를 선택
tag[속성*=값] { color: red; }  ==> 속성 안의 값이 특정값을 포함하는 태그를 선택
img[src^=../img2/] { border: 1px solid black; }
img[src$=png] { width: 200px; }
[class*=abc]  ==>  ★ 태그이름 같은 것을 쓰지 않아도, 사용가능! (class속성에 abc가 포함되는 모든 태그)
```

후손 선택자 : 선택자1 의 후손에 위치하는 선택자2 를 선택한다.

```
Selector1 Selector2 { color: red; }
#header .aaa { color: red; }
```

자손 선택자 : 선택자1 의 직계자식인 선택자2 를 선택한다.

```
Selector1 > Selector2 { color: red; }
```

동위 선택자 : 동위 관계(같은 레벨)에서 뒤에 위치한 태그를 선택할 때 사용하는 선택자 (radio버튼을 이용한 탭메뉴)

```
Selector1 ~ Selector2 { color: red; }            ==> Selector1 뒤에 위치하는 Selector2를 선택 (다수)
Selector1 + Selector2 { background-color: red; }  ==> Selector1 바로 뒤에 위치하는 Selector2를 선택 (1개)
```

반응 선택자 : 사용자의 반응으로 생성되는 특정한 상태를 선택하는 선택자

```
Selector:hover { color: red; }   ==> Selector 위에 마우스를 올렸을 때를 선택
Selector:active { color: red; }  ==> Selector 를 클릭했을 때를 선택
```

상태 선택자 : input 태그의 상태를 선택할 때 사용하는 선택자

inputTag:checked { display: none; } ==> type="checkbox, radio" 가 선택되었을 때
inputTag:focus { display: none; } ==> input태그가 포커스 되었을 때
inputTag:enabled { display: none; } ==> 사용 가능한 input 태그를 선택
inputTag:disabled { display: none; } ==> 사용 불가능한 input 태그를 선택

★ 속성 선택자 + 상태 선택자 + 동위 선택자 응용

input[type=checkbox]:checked + div { height: 0px; } ==> checkbox가 선택되면 div의 height를 0으로 만듦
input[type=radio][id=tab1]:checked ~ div.tabContent { display: block; }
==> radio 버튼이 선택되는 것에 따라, 탭메뉴의 내용이 display: none -> block 으로 변경됨. (label태그 사용)

일반 구조 선택자 : 특정한 위치에 있는 태그를 선택하는 선택자 (형제들 중, 다른 태그가 섞여있다면 주의가 필요함)

Selector:first-child { border-radius: 10px 0 0 10px; } ==> 선택자의 형제들 중에서 첫 번째 태그를 선택
Selector:last-child { border-radius: 0 10px 10px 0; } ==> 선택자의 형제들 중에서 마지막 태그를 선택
Selector:nth-child(2n) { color: hotpink; } ==> 선택자의 형제들 중에서 **앞에서 수열 번째** 태그를 선택
Selector:nth-last-child(2n+1) { color: red; } ==> 선택자의 형제들 중에서 **뒤에서 수열 번째** 태그를 선택

형태 구조 선택자 : 일반 구조 선택자와 비슷하지만, 태그 형태를 구분함. (상황에 따라, 조금 더 편할 수도 있음)

Selector:first-of-type { border-radius: 10px 0 0 10px; } ==> 선택자의 형제들 중에서 첫 번째 태그를 선택
Selector:last-of-type { border-radius: 0 10px 10px 0; } ==> 선택자의 형제들 중에서 마지막 태그를 선택
Selector:nth-of-type(2n) { color: hotpink; } ==> 선택자의 형제들 중에서 **앞에서 수열 번째** 태그를 선택
Selector:nth-last-of-type(2n+1) { color: red; } ==> 선택자의 형제들 중에서 **뒤에서 수열 번째** 태그를 선택
body > *:first-of-type { color: red; } ==> 이걸 써보면 어떤 건지 알 수 있다.

문자 가상 요소 선택자 : 태그 내부 특정 조건의 문자를 선택하는 선택자. :: 기호를 사용하는 것이 표준이다.

(1) 시작 문자 선택자 : 태그 내부의 첫 번째 글자와 첫 번째 줄을 선택할 때 사용하는 선택자

Selector::first-letter { color: red; } ==> 첫 번째 글자를 선택
Selector::first-line { color: red; } ==> 첫 번째 줄을 선택

(2) 전후 문자 선택자 : 택정 태그의 전후에 위치하는 공간을 선택하는 선택자 (정말 거의 사용할 일이 없음...)

Selector { counter-increment: aaa; } ==> 선택자에게 aaa 라는 시퀀스 넘버를 부여함. (1부터시작)
Selector::before { content: counter(aaa) "."; }
==> 문자의 앞에 시퀀스넘버+" ." 을 삽입함. (예: 1.)
Selector::after { content: " - " attr(data-page) " page"; }
==> 문자의 뒤에 " - "+data-page속성의 값+" page"를 삽입함. (예: - 52 page)
==> 속성 앞에 문자열 data- 를 붙이면, 사용자 지정 속성으로 인정해준다. (웹표준에 따르기 위해)

(3) 반응 문자 선택자 : 사용자가 문자와 반응해서 생기는 영역을 선택하는 선택자

Selector::selection { background-color: black; color: white; } ==> 사용자가 드래그한 글자를 선택

링크 선택자 : href 속성을 가지고 있는 a 태그에 적용되는 선택자, 한번 다녀온 링크의 색깔 등을 조정할수 있다.

aTag:link { color: red; } ==> href 속성을 가지고 있는 a 태그를 선택
aTag:visited { color: blue; } ==> 방문했던 링크를 가지고 있는 a 태그를 선택
aTag:link::after { content: " - " attr(href); }
==> href 속성을 가지고 있는 a 태그의 맨 뒤에 ' - '+주소 를 추가한다.

부정 선택자 : 지금까지 배운 선택자를 모두 반대로 적용할 수 있게 만드는 선택자

Selector:not([type=password]) { background-color: red; }
==> Selector로 선택된 태그들 중, type속성이 password인 태그를 제외한, 모든 태그를 선택

< CSS3 스타일 속성 기본 >

1. CSS3 의 단위

- 주로 px, %, em를 사용. (cm, mm, inch 도 있음)
- 100% === 1.0em, 150% === 1.5em
- 디폴트 font-size 는 16px임

(1) 색상 단위

- 색상단위를 키워드로 사용하면 간편함 (red, green, blue, orange, hotpink, lightgreen, skyblue 등)
- 색상단위 http://www.w3schools.com/cssref/css_colornames.asp

```
#000000 ~ #FFFFFF  ==>  Hex 코드 단위
rgb(255, 255, 255)            ==>  RGB 색상 단위
rgba(red, green, blue, alpha)  ==>  RGBA 색상 단위
hsl(hue, saturation, lightness)          ==>  HSL 색상 단위 (색상, 채도, 명도)
hsla(hue, saturation, lightness, alpha)  ==>  HSLA 색상 단위
```

(2) URL 단위

- CSS3에서 이미지 파일이나, 폰트 파일을 불러올 때 사용.

```
background-image: url("../chocolate.jpg");
```

2. 가시 속성

- 태그가 화면에 보이는 방식을 지정하는 속성

```
display: none | block | inline | inline-block;
visibility: visible | hidden | collapse(table태그에서만 사용);
opacity: 0.0 ~ 1.0;  (0은 완전투명, 1은 완전불투명)
```

- display: none 속성과 visibility: hidden 속성의 차이는?

```
display: none       ==>  태그가 화면에서 완전히 제거됨
visibility: hidden  ==>  태그의 자리는 차지한 상태로, 화면에 보이지만 않을 뿐
```

3. 박스 속성

- 웹 페이지의 레이아웃을 구성할 때 가장 중요한 스타일 속성
- margin, border, padding, width, height 속성을 모두 박스속성 이라고 한다.
- 태그 전체의 크기는 아래의 공식으로 나타낼 수 있다.

```
전체 너비 = width + (2 * (margin + border + padding))
전체 높이 = height + (2 * (margin + border + padding))
```

(1) width 속성과 height 속성 ==> 글자, 이미지 등의 컨텐츠를 감싸는 영역의 크기를 지정하는 스타일 속성

(2) margin 속성과 padding 속성 ==> margin(외부여백), padding(내부여백)을 조정

```
margin: [margin-top] [margin-right] [margin-bottom] [margin-left] | auto;
margin: [margin-상하] [margin-좌우] | auto;
padding: [padding-top] [padding-right] [padding-bottom] [padding-left] | auto;
padding: [padding-상하] [padding-좌우] | auto;
```

(3) box-sizing 속성 ==> 컨텐츠를 감싸는 영역의 크기를 지정하는 공식을 변경할 수 있는 CSS3 속성

- width 속성과 height 속성이 차지하는 범위를 지정한다.

```
box-sizing: content-box(default) | border-box
==> content-box --> width 와 height 가 padding과 border를 포함하지 않음
==> border-box  --> width 와 height 가 padding과 border를 포함함! (좀 더 쉬운 레이아웃 픽셀계산이 가능함)
        전체 너비 = width + (2 * margin)
        전체 높이 = height + (2 * margin)
```

(4) border (테두리) 속성

border: 1px solid black;
border-width: 1px | thick | thin ...; (테두리의 두께)
border-style: solid | dashed | dotted | double | groove ...; (테두리의 형태)
border-color: black | red | green | blue | #000000 ...; (테두리의 색깔)
border-top, right, bottom, left 를 따로도 사용 가능

border-radius: 10px; (테두리의 모서리를 둥글게 깎음, 원형으로 만들수도 있다.)
border-radius: 5px 10px 20px 30px; (10시, 2시, 5시, 7시방향)

5. 배경 속성

- 특정 태그의 배경 이미지 또는 색상을 지정하는 스타일 속성

background-image: url("../chocolate.jpg");
background-image: url("../chocolate.jpg"), url("../milk.png"); ==> 여러 개의 배경이미지를 적용가능!

background-position: top | right | bottom | left; ==> ★ sprite 이미지를 사용할 때 중요하다!
background-position: x축위치; (width 와 height 속성으로 잘라줘야 한다)
background-position: x축위치 y축위치;

background-size: 100% | contain | cover;
==> 100% --> 가로세로를 화면을 꽉 채운 상태에서, 원본비율 유지함
==> contain --> 너비를 100% 적용한 것과 같음
==> cover --> 높이를 100% 적용한 것과 같음
background-size: 100%; 250px; ==> 가로화면을 꽉 채운 상태에서, 원본비율을 유지하며, 세로 250px 로 자름
background-size: 100%, 50%; ==> 컴마로 구분하면 url() 2개의 이미지에 대해서 각각 적용시킴

background-repeat: repeat(default) | no-repeat | ...;
background-attachment: scroll(default) | fixed | ...; ==> ★ 패럴렉스 스크롤링 기법에 사용함.
background-color: red | green | blue | ...;

background: color image position/size repeat origin clip attachment; ==> 한 번에 쓸 수 있다.

6. 폰트 속성

(1) 글자와 관련된 스타일 속성

- 원래 HTML 페이지는 대학에서 원격으로 논문을 제출하고 확인하는 용도로 만들어졌기 때문에, 글자관련 속성이 많음.

크기) font-size: 32px | 2em | large | small;
글꼴) font-family: "없을수도 있는폰트", "Times New Roman", Arial;
==> font-family 의 가장 마지막폰트는 Serif(명조체), Sans-serif(고딕체), Mono space(고정폭 글꼴)을 적용함.
형태) font-style: italic;
두께) font-weight: 400(default) | 700(bold);

(2) line-height 속성

- 글자의 높이를 지정. 실제로는 글자의 높이를 지정하기보단, **글자를 수직 중앙 정렬**할 때 많이 사용한다.

display: block; text-align: center; line-height: 50px;

(3) text-decoration 속성 -> 링크에 밑줄을 조절하는 속성

a { text-decoration: none; }

7. 위치 속성

(1) ★★★ position 속성의 키워드 ★★★

static(default) ==> HTML요소는 기본적으로 정적배치 됨. top, bottom, left, right의 영향을 받지 않는다.
페이지의 기본적인 흐름에 따라 자동으로 배치된다.

relative ==> 기본(normal)적인 static 상태의 위치를 기준으로,
top, bottom, left, right 속성을 사용해서, 새로운 위치를 잡을 수 있게 된다.

fixed ==> 내가 보고 있는 화면의 좌표를 기준으로 위치를 잡는다.
viewport에 상대적으로 자리잡게 된다. 페이지를 스크롤해도 항상 동일한 위치에 머물게 된다.
위로 붕 뜨게 되어, 다른 Element의 위치에 영향을 미치지 않는다. (옆에 따라다니는 광고 등)

absolute ==> 가장 가까운 position 속성(static제외)이 적용된 부모태그로부터, 상대적으로 위치를 잡는다.
position속성이 적용된 부모태그가 없을 경우, body태그(웹페이지 전체)를 기준으로
위치를 잡는다. (모달 광고창 등)

(2) z-index 속성 ==> 숫자가 클수록 앞에(위에?) 위치하게 된다.

```
div { z-index: -9999 ~ 9999; }
```

(3) overflow 속성 ==> 내부의 요소가 부모의 범위를 벗어날 때, 어떻게 처리할지 지정하는 속성

- 프로그래밍 요소를 사용해 애니메이션을 구현 할 때 많이 사용함
- 사실 float 속성과 함께, 다른 용도로 쓰이는 경우가 더 많음.
- 그것을 감싸는 태그보다 큰 Element가 있고, 그것이 float 되어있을 경우,
  그것을 그것을 감싸는 container의 밖으로 오버플로우된다.
  우리는 overflow: auto; 를 추가함으로써, containing Element의 문제를 해결할 수 있다.
  --> 이를 “The clearfix Hack” 이라고 부른다.

```
overflow: hidden;  ==>  영역을 벗어나는 부분을 보이지 않게 만든다.
overflow: scroll;  ==>  영역을 벗어나는 부분을 스크롤로 만든다.
overflow: auto;
overflow-x: hidden | scroll;
overflow-y: hidden | scroll;
```

(4) text-indent 속성 ==> 사실은 들여쓰기 속성이지만, ScreenReader에는 잡히되, 눈에는 안보이고 싶은 경우 사용

```
p { text-indent: -99999px; }
```

8. float 속성과 clear 속성

(1) float 속성

- 부유하는 대상을 만들 때 사용하는 스타일 속성
- 간단하게는, 이미지를 글자로 감싸기 위해 사용될 수 있다.
- 원래 float 속성은, 이미지를 글자 위에 띄우기 위해 만들어졌다.
- 보통 레이아웃에서 aside 메뉴를 사용할 때 사용한다.

```
float: left | right;
* float: right 의 경우, 좌우가 거울모드로 뒤집힘에 주의해야한다.
```

(2) clear 속성

- clear 속성은 floating Element의 동작을 제어하기 위해 사용된다.
- clear 속성이 적용된 요소 옆에서 floating Element는 float을 허용하지 않음.

```
clear: none(default) | left | right | both | initial | inherit;
```

* 첨언... position: absolute 속성과 float: left 속성의 차이점

position: absolute ==> 위로 붕 뜨고, 영역을 차지하지 않는다. top, left 등으로 절대좌표를 설정한다.
float: left ==> 영역을 차지한다. (태그를 왼쪽 또는 오른쪽으로 붙인다.) 순서대로 차곡차곡 붙는다.

9. 그림자 속성

- 글자 또는 박스에 그림자를 부여하는 속성 (CSS3 Generator 를 사용하면 간편하다.)
- 여러 개의 그림자를 연달아서 사용할 수 있다.

```
text-shadow: 5px 5px 5px black;  (오른쪽, 아래, 흐림도, 색상 순서)
box-shadow: 5px 5px 5px black, 10px 10px 10px orange, 20px 20px 20px skyblue;  (상동)
```

10. Vendor Prefix (벤더 프리픽스)

- 웹 브라우저 공급 업체에서 제공하며, 실험적인 기능이 필요할 때 사용
- Prefix Free 플러그인으로 손쉽게 해결할 수도 있다. http://leaverou.github.io/prefixfree/
  (style sheet 파일이 모두 로딩되고, 맨 마지막에 로딩되어야한다.)

```
-ms-transition-duration: 1s;
-webkit-transition-duration: 1s;  (크롬 & 사파리)
-moz-transition-duration: 1s;
-o-transition-duration: 1s;
transition-duration: 1s;
```

11. Gradient (그레이디언트)

- CSS3 Gradient Generator 를 사용해서 만드는 것이 편리하다. (이걸 직접 타이핑하는사람은 거의 없다.)

```
background: linear-gradient(top bottom, #f85032 1%, #f16f5c 50%, #f6290c 51%, #e73827 100%);
==> linear-gradient(45deg, #f85032 0%, #e73827 100%)  -->  각도, 색상, 위치순서
```

12. 변형 속성 --> transition

- CSS3 에서 움직임을 구현할 수 있는 기능은 '변형' 속성과 '애니메이션' 속성으로 나뉜다.
- 주의! ★ **transition 속성은 반드시 기본형태의 선택자에 넣는다!** (:hover :focus 등에 넣지 않는다! 동작이 끊긴다.)

```
.box { transition-duration: 2s; }
transition-duration: 2s;  ==>  몇 초 동안 변형할지 지정
transition-delay: 1s;     ==>  이벤트 발생 후, 몇 초 후에 변형하기 시작할지 지정
transition-property: width, height, background-color;  ==>  어떤 속성을 변형할지 지정 (쓰지 않으면 모두)
transition-timing-function: ease(default) | linear | ease-in-out | cubic-bezier(n,n,n,n);  ==>  변형함수
                                        (베지어 곡선을 만들어주는 웹사이트 --> http://cubic-bezier.com/)
transition: background-color 1s ease, width 5s linear 1s;  (property duration function delay 순서로 기입)
```

*** transition, animation 으로 변형 가능한 속성** ==> http://www.w3schools.com/cssref/css_animatable.asp

```
배경 속성 - background(-color, -position, -size) (background-image 는 불가능)
테두리 속성 - border / border-width, border-color, border-radius
위치 속성 - top, right, bottom, left, z-index
크기 속성 - width, height, line-height, min-height ~ max-width
박스 속성 - margin, padding, outline
색상 속성 - color, background-color
투명도 속성 - opacity, visibility (display 속성은 불가능)
변환 속성 - transform(-origin), perspective(-origin)
글자 속성 - font(size, weight, stretch, size-adjust) / text-decoration-color / text-indent / text-shadow
기타 - box-shadow / clip / flex / column-... / filter 등등등
```

13. 애니메이션 --> Animation (@keyframes)

- CSS3 에서는 애니메이션을 활용해서 플래시를 대체할 수 있다.
- 주의! ★ **background-image 속성은 애니메이션 가능한 속성이 아니다.** 대신, 구현하려는 위치에 position: absolute로 모든 이미지를 배치시켜놓고, 원하는 것을 제외한 나머지를 opacity 또는 visibility 을 주는 방식으로 해야 한다.
- 주의! **animation 중인 속성은 hover 같은 것으로 다시 건드릴 수 없는 모양**이다. (확실하진 않음, 경험상.)

```
animation-name: aaaaa;   ==>  애니메이션 이름을 지정
animation-duration: 2s;  ==>  몇 초 동안 재생할지 지정
animation-delay: 1s;     ==>  이벤트 발생 후, 몇 초 후에 재생할지 지정
animation-iteration-count: 숫자 | infinite;  ==>  반복 횟수를 지정
animation-direction: normal(from-to) | alternate(from-to-from)  ==>  애니메이션 진행 방향을 설정
animation-timing-function: ease(default) | linear | ease-in-out | cubic-bezier(n,n,n,n);  ==>  변형함수
animation-play-state: running | paused | initial;  ==>  재생 상태를 지정
                                        (자바스크립트로는 --> object.style.animationPlayState="paused")
animation-fill-mode: none(default) | forwards(100%정지) | backwards(?) | both(alternate일 때 from값);
animation: aaaaa 2s linear none infinite alternate;  (name duration function delay count direction 순서)

@keyframes tabChange {     <-- 마치 함수처럼 사용함.
    0% { background-color: red; }  (from키워드로 사용 가능)
   50% { background-color: green; }
  100% { background-color: blue; }  (to키워드로 사용 가능)
}
.box { animation: tabChange 3s infinite; }
```

14. 변환 속성 --> transform

- CSS3 에서는 transform 속성을 사용해서 3차원을 구현할 수 있다. (impress.js)
- 자바스크립트로는 WebGL 로 구현할 수도 있다.
- 화면좌표를 사용한다. (왼쪽 위가 영점. position: absolute 의 기준과 동일하다.) (데카르트 좌표 아님)
- translate의 단위는 **px**을 사용 / scale의 단위는 **숫자**를 사용(1이 기본 값, 100%) / 각도(angle)의 단위는 **deg**를 사용

(1) 2차원 변환 함수

```
translate(translateX, translateY)  ==>  특정 크기만큼 이동
translateX(translateX)     ==>  X축으로 특정 크기만큼 이동
translateY(translateY)     ==>  Y축으로 특정 크기만큼 이동

scale(scaleX, scaleY)   ==>  특정 크기만큼 확대 및 축소
scaleX(scaleX)  ==>  X축으로 특정 크기만큼 확대 및 축소
scaleY(scaleY)  ==>  Y축으로 특정 크기만큼 확대 및 축소

skew(angleX, angleY)   ==>  특정 각도만큼 기울임
skewX(angleX)  ==>  X축으로 특정 각도만큼 기울임
skewY(angleY)  ==>  Y축으로 특정 각도만큼 기울임

rotate(angleZ)  ==>  Z축으로 특정 각도만큼 회전
```

(2) 3차원 변환 함수
- 3차원 변환은, 아래 8개 함수가 추가됨.
translate3d(translateX, translateY, translateZ) ==> 특정 크기만큼 이동
translateZ(translateZ) ==> Z축으로 특정 크기만큼 이동
scale3d(scaleX, scaleY, scaleZ) ==> 특정 크기만큼 확대 및 축소
scaleZ(scaleZ) ==> Z축으로 특정 크기만큼 확대 및 축소

rotate3d(angleX, angleY, angleZ) ==> 특정 각도만큼 회전
rotateX(angleX) ==> X축으로 특정 각도만큼 회전
rotateY(angleY) ==> Y축으로 특정 각도만큼 회전
rotateZ(angleZ) ==> Z축으로 특정 각도만큼 회전

transform: rotate(60deg) scale(1.2) skewY(10deg); ==> transform 속성 하나 안에 모두 때려박는다.
--> ★ 입력하는 순서에 따라 실행 결과가 바뀔 수 있으므로 주의!!!
(간 후에 돌리는 것과, 돌린 후에 가는 것의 방향에는 차이가 존재한다.)
transform-origin: 0~100% 0~100% | right bottom; ==> 변환 중심을 설정하는 스타일 속성
transform-style: flat(default, 후손의 3차원속성을 무시) | preserve-3d(후손의 3차원속성을 유지)
==> 변환을 적용할 때, 그 영향력이 자신에게만 적용될지 자손에게도 적용될지 정하는 속성

(3) backface-visibility 속성
- 3차원 공간에서, 평면의 후면을 보이거나, 보이지 않게 만드는 스타일 속성
backface-visibility: visible(default) | hidden;

(4) 원근법 (perspective 속성)
- 화면에 얼마나 많은 3차원 픽셀을 놓을 것인지 정의하는 속성
- 0픽셀에서 1000픽셀로 이동할수록 밀집도가 증가함.
- perspective 속성이 클수록, 픽셀을 밀집해서 보여줌. (일반적으로 400px에서 2000px 사이의 값을 입력함)
perspective: 400px;

15. @-rule 규칙
(1) @keyframes
- 애니메이션을 정의할 때 사용하는 규칙 (함수와 비슷함)

(3) @import url("../styleSheetA.css");
- .css파일에서 다른 .css파일을 추가하는 방법 (비권장, 그냥 link를 많이 쓰는 것이 직관적이고 좋다.)

(2) @font-face
- 웹 폰트 업체에서 지원하지 않는 폰트는 자체적으로 지원해야 하는데, 이러한 웹 폰트를 생성할 때 사용하는 규칙
- local()함수로 엔드유저의 폰트를 확인하고, 해당 폰트가 존재하지 않으면 url() 함수로 폰트를 다운받는다.
- 웹 브라우저별로 지원하는 폰트의 확장자명이 다르므로, 여러 폰트를 넣으면 된다. (woff가 가장 범용적인 듯)

```
@font-face {
  font-family: "font name";    /* 추가하고자 하는 폰트의 이름을 적용 (마음대로 하면 됨) */
  src: local("NanumGothic")    /* EndUser의 컴퓨터 내부에 있는 폰트를 선택하는 함수 */
      , url("/content/NanumGothic.eot")
      , url("/content/NanumGothic.ttf")    /* EndUser의 컴퓨터에 존재하지 않는 폰트를 지정하는 함수 */
      , url("/content/NanumGothic.woff");
}
* { font-family: "font name"; }
```

(4) @media

- 다양한 장치에서 HTML 문서가 적절한 형태를 갖추게 만들어주는 규칙 (★반응형 웹!)
- HTML 4.01 버전에서는 link태그의 media 속성에 적절한 키워드(print, screen 등)를 사용해서,
  스타일시트가 특정한 디바이스에서만 작동하게 함.
- 장치에 따라 stylesheet를 다르게 적용하는 방법 3가지
  1. @import 규칙에 장치 종류를 입력합니다. (사용하는 것 자체를 비권장)
  2. link 태그에 media 속성에 장치 종류를 입력합니다. <link rel="stylesheet" href="print.css" media="print" />
  3. @media 규칙에 장치 종류를 입력합니다.

```
@media screen {
  html { height: 100%; background: black; }
  body { color: white; font-family: serif; }
}
@media print {
  h1 { text-align: center; color: red; font-family: sans-serif; }
}
```

- "미디어 쿼리"를 함께 사용 (orientation 속성을 제외한 모든 속성은 min, max 접두사를 사용할 수 있다)

```
width, height  ==>  화면의 너비와 높이
device-width, device-height  ==>  장치의 너비와 높이
device-aspect-ratio  ==>  화면의 비율
orientation  ==>  장치의 방향 (portrait | landscape)
color  ==>  장치의 색상 비트
color-index  ==>  장치에서 표현 가능한 최대 색상 개수
monochrome  ==>  흑백 장치의 픽셀당 비트 수
resolution  ==>  장치의 해상도
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
==>  ★꼭 있어야함!!! <meta>태그가 없으면 웹페이지에게 화면 너비와 관련된 정보를 전달할 수 없다!
```

```
<style>
@media screen and (max-width: 767px) {                          /* 화면 너비   1px ~ 767px */
  html { background: red; color: white; font-weight: bold; }
}
@media screen and (min-width: 768px) and (max-width: 959px) {    /* 화면 너비 768px ~ 959px */
  html { background: green; color: white; font-weight: bold; }
}
@media screen and (min-width: 960px) {                          /* 화면 너비 960px ~ 무한px */
  html { background: blue; color: white; font-weight: bold; }
}
@media screen and (orientation: portrait) {    /* 가로세로 픽셀중, 세로픽셀이 1px이라도 더 길면 */
  html { background: orange; color: white; font-weight: bold; }
}
@media screen and (orientation: landscape) {   /* 가로세로 픽셀중, 가로픽셀이 1px이라도 더 길면 */
  html { background: pink; color: white; font-weight: bold; }
}
</style>
```

16. 반응형 웹

- 간단한 반응형 웹 미디어쿼리 소스

```
<style>
    @media screen and (max-width: 767px) {    /* 화면너비 0px ~ 767px : 그리드를 해제 */
      .container { text-align: center; }
    }
    @media screen and (min-width: 768px) and (max-width: 959px) {    /* 화면너비 768px ~ 959px : 동적 그리드 */
      .container { margin: 0; padding: 0; }
      .row { overflow: hidden; }
      .row [class*=span_] { float: left; }
      .span_1  { width: 8.33%; }      .span_2  { width: 16.66%; }      .span_3  { width: 24.99%; }
      .span_4  { width: 33.32%; }     .span_5  { width: 41.65%; }      .span_6  { width: 49.98%; }
      .span_7  { width: 58.31%; }     .span_8  { width: 66.64%; }      .span_9  { width: 74.97%; }
      .span_10 { width: 83.3%; }      .span_11 { width: 91.63%; }      .span_12 { width: 99.96%; }
    }
    @media screen and (min-width: 960px) {    /* 화면너비 960px ~ 무한px : 정적 그리드 시스템 */
      .container { width: 960px; margin: 0 auto; padding: 0; }
      .row { overflow: hidden; }
      .row [class*=span_] { float: left; }  /* 고수들의 방법 (클래스속성에 span_가 포함되는 모든 태그를 선택) */
      .span_1  { width: 80px; }      .span_2  { width: 160px; }      .span_3  { width: 240px; }
      .span_4  { width: 320px; }     .span_5  { width: 400px; }      .span_6  { width: 480px; }
      .span_7  { width: 560px; }     .span_8  { width: 640px; }      .span_9  { width: 720px; }
      .span_10 { width: 800px; }     .span_11 { width: 880px; }      .span_12 { width: 960px; }
    }
</style>
```

< 간단한 공식 >

*** 초간단 Reset CSS**

- 브라우저마다 태그에 설정되어 있는 기본 값이 조금씩 다르다.
  일관된 홈페이지 및, 유지-보수를 위해 그것들을 모두 초기화해줄 필요가 있다.
- 아래의 Reset CSS 는 모든 스타일시트가 로딩 되기 이전, 최상단에 삽입한다. (common.css 의 최상단에 삽입)

```
* { margin: 0; padding: 0; font: 12px "Dotum"; }
a { text-decoration: none; color: #000000; }
li { list-style: none; }    /* ul 에 적용해도 무관하며, ol의 숫자를 좋아한다면 ul에 넣자. */
```

*** 수평(가로) 가운데 정렬을 하는 4가지 방법**

규칙 1. block속성을 margin: 0 auto; 로 가운데정렬 해주려면, 반드시 width 속성을 지정해야한다.
(margin의 auto로 인해, computed width 문제가 발생한다, width: auto; 또한 불가능.)

규칙 2. body 태그에는 사용하지 않을 것을 권장한다. 이유는 aside(광고) 등을 원하는 위치에 배치할 수 없기 때문이다.

*** width의 너비가 정해졌을 경우**

```
header, nav, main, footer { width: 960px; margin: 0 auto; }
header > div { width: 300px; margin: 0 auto; }
```

*** 정확히 가운데는 아닌데, 어느정도 가운데일 경우**

```
div { position: absolute; left: 50%; margin-left: -(width/2)px; }
```

*** 자식의 width가 정해지지 않았을 경우 (동적으로 변할 경우, 반응형)**

```
div.parent { display: flex; justify-content: flex-end; margin-right: 19%; }
div.child { width: 25%; min-width: 200px; max-width: 700px; }
```

*** inline 속성일 경우**

inline 속성은 부모태그의 **text-align: center** 속성으로 가운데정렬 가능.

*** 수직(세로) 가운데 정렬을 하는 2가지 방법**

*** Element 안에서 특정 Element를 수직 가운데정렬 하고싶을 때**

```
div { position: absolute; top: 50%; height: 100px; margin-top: -(height/2)px; }
```

*** 버튼이나 셀 안에 글자가 가로세로 중앙정렬 되도록 하기**

```
.button { display: block; text-align: center; line-height: 30px; }
```

==> inline속성을 먼저 block으로 변경(너비, 높이조절이 가능해짐)하고, 좌우정렬, 세로정렬

*** 지정된 영역을 벗어나는 글자를 "..." 처리하기**

```
p { width: 200px; white-space: nowrap; overflow: hidden; text-overflow: ellipsis; }
```

==> 영역설정 / 줄바꿈 없음 / 영역초과 숨기기 / 글자가 넘어가면 "..." 으로 대체

*** 위치 속성과 관련된 공식**

- position 속성을 사용함에 있어서, absolute 키워드를 사용하면 2가지 문제점이 발생한다.

1) 부모를 기준으로 위치를 잡지 않는다.

2) position: absolute; 를 적용한 태그는 위로 뜬 상태로, 영역을 차지하지 않는다.

==> 자손의 position 속성에 absolute 키워드를 적용하면,
부모의 **position 속성에 relative 키워드**(부모기준 위치설정)와, **height 속성**(영역차지)을 사용한다.

==> 부모의 영역 안을 넘지 못하도록 하고 싶으면 **overflow: hidden;** 을 추가한다.

*** 글자가 ScreenReader 에는 잡히지만, 시각적으로는 보이지 않게 만드는 2가지 방법 (부트스트랩의 sr-only)**

```
.sr-only { position:absolute; width:1px; height:1px; margin:-1px; overflow:hidden; clip:rect(0,0,0,0); }
p { background: url("sprites.png"); text-indent: -99999px; }  ==> 그림은 보이되, 글자만 안보이게 함(버튼)
```

*** float 속성을 사용한 레이아웃 구성 (One True Layout 방식)**

자손에 float 속성을 적용하면,
부모의 overflow 속성에 hidden 키워드를 적용한다.
그러나 여기서 overflow: hidden 속성과 함께 width, 또는 height 속성을 주면, 해당 부분만큼 잘리게 된다.

```
  <article>
    <aside>
      <h1>Aside</h1>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit. <p>
    </aside>

    <section>
      <h1>Section</h1>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.</p>
    </section>
  </article>
aside { width: 200px; float: left; }
section { width: 780px; float: left; }
article { overflow: hidden; }  <-- ★ overflow: hidden 으로 인해, float으로 부유되어있는 자식들까지 하나로 묶어주게 된다.
```

단순하게 float속성을 자식태그(aside, section)들에게 부여한다면, 그들은 부모태그로부터 가출한 자식태그들이 된다.
부모가 자식을 놓치게 되는 것이다. 그래서 부모태그(article)의 height는 0px 상태가 된다.
그러나 overflow: hidden; 속성을 부여하게 되면, 부모태그(article)는 자식태그(aside, section)를 붙잡을 수 있게된다.
그래서 그들은 하나가 된다.(응?) (정정 : 하나의 block 으로서 자리 잡게 된다.)

*** 패럴렉스 스크롤링**

```
  <div id="parel1"> </div>
  <div id="parel2"> </div>
  <div id="parel3"> </div>
  #parel1 {
    height: 1000px;
    background-image: url("aaa.jpg");
    background-size: 100%;
    background-attachment: fixed;
    position: relative;  /* relative는 나중에 하위 태그가 absolute 일 경우, 초토화됨을 예방함. */
  }
  #parel2 {
    height: 500px;
    background-image: url("bbb.jpg");
    background-size: 100%;
    position: relative;
  }
  #parel3 {
    height: 1000px;
    background-image: url("ccc.jpg");
    background-size: 100%;
    background-attachment: fixed;
    position: relative;
  }
```

* CSS 만으로 보이거나 사라지는 “탭” 만들기 (radio버튼 활용, 탭 3개 + 네비게이션(choiceBox version)

```
    <section class="newProduct">
      <input type="radio" name="np" id="newProduct1" checked />
      <input type="radio" name="np" id="newProduct2" />
      <input type="radio" name="np" id="newProduct3" />
      <div class="choiceBox">
        <label for="newProduct1">1</label>
        <label for="newProduct2">2</label>
        <label for="newProduct3">3</label>
        <span>/3</span>
        <label for="newProduct1">1</label>
        <label for="newProduct2">2</label>
        <label for="newProduct3">3</label>
      </div>
      <div class="newProduct1">탭1 내용</div>
      <div class="newProduct2">탭2 내용</div>
      <div class="newProduct3">탭3 내용</div>
    </section>

.newProduct input[type*=radio] { display: none; }
.newProduct .choiceBox { position: absolute; top: 10px; right: 0px; }
.newProduct .choiceBox label { display: none; }
.newProduct .choiceBox label::selection { background-color: white; color: initial; }

.newProduct [id=newProduct1]:checked ~ .choiceBox span::before { content: "1"; }
.newProduct [id=newProduct1]:checked ~ .choiceBox label:nth-of-type(3) { display: inline-block; color: green; }
.newProduct [id=newProduct1]:checked ~ .choiceBox label:nth-of-type(5) { display: inline-block; color: green; }
.newProduct [id=newProduct2]:checked ~ .choiceBox span::before { content: "2"; }
.newProduct [id=newProduct2]:checked ~ .choiceBox label:nth-of-type(1) { display: inline-block; color: green; }
.newProduct [id=newProduct2]:checked ~ .choiceBox label:nth-of-type(6) { display: inline-block; color: green; }
.newProduct [id=newProduct3]:checked ~ .choiceBox span::before { content: "3"; }
.newProduct [id=newProduct3]:checked ~ .choiceBox label:nth-of-type(2) { display: inline-block; color: green; }
.newProduct [id=newProduct3]:checked ~ .choiceBox label:nth-of-type(4) { display: inline-block; color: green; }

.newProduct div { display: none; }
.newProduct [id=newProduct1]:checked ~ div.newProduct1 { display: block; }
.newProduct [id=newProduct2]:checked ~ div.newProduct2 { display: block; }
.newProduct [id=newProduct3]:checked ~ div.newProduct3 { display: block; }
```

made by Rex

< CSS 고급 >

*** Flex (ie 11버전부터 되는 CSS 3 최신기능)**

==> 어떤 한 공간에 대해서, 각 Element들이 width 또는 height 속성에 대해서 특정 비율을 갖도록 한다.
==> 반응형 UI, 디스플레이 장치나 모든 크기의 디바이스를 수용하는 웹페이지를 만들 때, 아주 유용하다.
==> 배치 정렬에 대해서, 그 크기를 알 수 없는 동적인 상황인 경우에도, 컨테이너의 항목 사이의 공간을 보다 효율적인 방법으로 정렬하는 것을 목적으로 제공됨.

참고 웹사이트.
https://philipwalton.github.io/solved-by-flexbox/
https://css-tricks.com/snippets/css/a-guide-to-flexbox/

1. 부모태그에는 **display: flex; 속성**을 부여한다.
    * **display: flex 속성**이란, display: flex 속성을 가진 태그의 자식태그들이,
      특정하게 (화면크기에 따라 일관성 있는) 비율을 가지며, 순차적으로 배치될 수 있도록 지시하는 속성이다.
      만약 자식들의 width가 부모의 width를 넘어설 경우에는, 부모의 height를 균등하게 분배하여 줄 바꿈이 된다.
  추가적으로 아래의 속성을 부여할 수 있다.
  * flex-direction ==> 비율을 나눠가질 방향을 지정
    flex-direction: row(default) | row-reverse | column | column-reverse;
  * flex-wrap ==> flex 속성을 포장(유연하지 않도록)할지 안할지(유연하도록, 기본값) 지정함.
    flex-wrap: nowrap(default) | wrap | wrap-reverse;
  ==> 기본적으로 Flex 항목들은 모두 1줄에 맞추도록(nowrap) 되어있는데,
    상황에 따라 wrap(포장, 유연하지 않도록)하여 새로운 라인으로 적층할 수 있다.
    wrap을 사용하려면 Flex items중 하나 이상의 item에 width 속성 또는 height 속성이 반드시 존재해야한다.
  축약형 --> **flex-flow**: flex-direction flex-wrap; (디폴트는, row nowrap)

2. 자식태그에는 width 또는 height 속성을 부여하지 않고, 대신에 flex 속성을 부여해서,
   width 또는 height 속성을 특정 비율(할당량)로 나눠 갖도록 지정한다. 아래의 속성을 사용한다.

    * 자식 Element의 width 또는 height 속성이 존재하지 않을 경우
    flex-grow: 0; (default)
    ==> flex-grow: 0; 을 주게되면 자식태그는 자신의 컨텐츠 크기만큼 width가 정해진다.(늘어남없음, AutoWidth)
    ==> flex-grow: 1; 을 주게되면 자식태그는 display: flex된 부모의 width 크기에서 1의 비율만큼 늘어난다.
                만약 flex-grow속성이 없는 다른 자식태그들이 있다면,
                해당 자식태그의 width만큼은 빼고 나머지를 차지하게 된다.
    ==> flex-grow: 2; 을 주게되면 자식태그는 display: flex된 부모의 width 크기에서 2의 비율만큼 늘어난다.

    * 자식 Element의 width 또는 height 속성이 존재할 경우
    flex-shrink: 1; (default)
    ==> flex-shrink: 0; 을 주게되면 자식태그의 원본 width 또는 height 를 그대로 유지한다. (줄어들지 않는다)
    ==> flex-shrink: 1; 을 주게되면 자식태그는 display: flex된 부모의 width 크기에서 1의 비율만큼 줄어든다.
    ==> flex-shrink: 2; 을 주게되면 자식태그는 display: flex된 부모의 width 크기에서 2의 비율만큼 줄어든다.

    * 자식의 기본적인 width를 정해주고 싶을 때 (0을 넣으면 content-box크기만 가지게 된다.)
    flex-basis: 75px | 100% | 3em | 15vw | auto(default);
    ==> display: flex된 부모의 width 크기를 기준으로 기본적인 width크기를 가지도록 한다.

    위 3개 속성의 축약형 --> **flex**: flex-grow flex-shrink flex-basis | auto; (디폴트는, 0 1 auto)

*** Flex 에 대한 정렬**

- **부모태그에 "display: flex" 속성과 함께 사용**하여, **자식태그들을 정렬**한다. (flex-direction: row 기준 설명)
- 이 설명에서 말하는 margin 값을 가진다는 것은, 실제로 margin값을 갖는 것이 아니라, 형태가 그리 보인다는 것이다.

1. **align-items** ==> **수직 정렬**, flex-items 를 정렬
   - align-items: stretch; (default) ==> 자식태그들의 height가 없을 경우, 부모태그의 height 만큼 가득 찬다.
   - align-items: center; ==> 부모태그의 height를 기준으로, 자식태그의 content-box의 위치가 가운데 정렬된다.
   - align-items: flex-start; ==> 부모태그의 height에서 맨 위에 딱 붙는다.
   - align-items: flex-end; ==> 부모태그의 height에서 맨 아래에 딱 붙는다.

2. **align-content** ==> **수직 정렬**, flex-lines 를 정렬
   flex-wrap: wrap; 일 경우와, 반드시 여러 줄이 있어야 효과를 발휘한다.
   - align-content: stretch; (default) ==> 부모태그의 height를 기준으로 좌상단(0px, 0px)부터 균등한 margin-bottom 값을 가지며, 세로로 분할하여 배치된다.
   - align-content: center; ==> 부모태그의 height를 기준으로 자식태그의 margin(top, bottom) 없이 가운데로 배치된다.
   - align-content: flex-start; ==> 자식태그의 margin(top, bottom) 없이 좌상단(0px, 0px)부터 순차적으로 세로로 배치된다.
   - align-content: flex-end; ==> 자식태그의 margin(top, bottom) 없이 좌하단에 맞게, 순차적으로 세로로 배치된다. (거꾸로 배치되는 것이 아니다.)
   - align-content: space-between; ==> 자식태그의 컨텐츠가 4개라면, 부모태그의 상단과 하단 끝에 붙고, 2개는 동일한 margin(top, bottom)을 가지며, 동일한 간격으로 배치된다.
   - align-content: space-around; ==> 자식태그의 컨텐츠가 4개라면, 모두 동일한 margin(top, bottom)을 가진다.

3. **justify-content** ==> **수평 정렬** (default = flex-start)
   - justify-content: center; ==> 부모태그의 width를 기준으로, 자식태그의 content-box의 위치가 가운데 정렬된다. (margin 없이 딱 붙어서 가운데로 온다.)
   - justify-content: flex-start; ==> 부모태그의 좌상단(0px, 0px)부터 margin없이 순차적으로 위치하게 된다.
   - justify-content: flex-end; ==> 부모태그의 우상단에 맞게, margin없이 순차적으로 위치하게 된다.
   - justify-content: space-between; ==> 자식태그의 컨텐츠가 4개라면, 부모태그의 좌측과 우측 끝에 붙고, 2개는 동일한 margin(left, right)을 가지며, 동일한 간격으로 배치된다.
   - justify-content: space-around; ==> 자식태그의 컨텐츠가 4개라면, 모두 동일한 margin(left, right)를 가진다.

4. **align-self** ==> 자식태그가 본인 스스로를 부모의 범위 내에서 개별적으로 정렬, 부모태그의 align-items 속성을 override(무시) 한다.
   - align-self: auto; (default) ==> 부모태그의 정렬(align-items)를 따른다.
   - align-self: stretch; ==> height가 없을 경우, 부모태그의 height 만큼 가득 찬다.
   - align-self: center; ==> 부모태그의 height를 기준으로, 자신의 content-box의 위치가 가운데 정렬된다.
   - align-self: flex-start; ==> 부모태그의 height에서 맨 위에 딱 붙는다.
   - align-self: flex-end ==> 부모태그의 height에서 맨 아래에 딱 붙는다.
   - align-self: baseline; ==> 자신의 content-box의 크기만큼 영역을 차지하며, 기본적인 static position을 차지하는 것처럼, 위에 붙는다.

5. **order** ==> Flex Container의 자식태그(items)들 중에, 표시되는 순서에 대한 정렬속성이다.
   (**z-index**와 같은개념... **x-index** 라고나 할까...?!)
   - order: 0; (default)
   - order: -1;
   - order: 9999;

*** Flex 에 대한 정리**

- 부모가 되는 태그에 사용할 수 있는 속성

```
div.parent {
        display: flex;  <---  ★
        flex-direction: row;
        flex-wrap: nowrap;  (flex-flow 속성으로 축약해서 사용가능)
        align-content: center;  (flex-wrap 속성의 값이 wrap 일 경우)
        align-items: center;
        justify-content: space-around;
}
```

- 자식이 되는 태그에 사용할 수 있는 속성

```
div.parent > div.child {
        flex-grow: 0;
        flex-shrink: 1;
        flex-basis: 25%;  (flex 속성으로 축약해서 사용가능)
        align-self: flex-end;  (부모태그의 align-items 속성을 오버라이드(무시)한다. 자식마음대로 행동!)
        order: -1;
}
```

< 간단한 공식 >

1. Better, Simpler Grid Systems ==> 반응형, 동적레이아웃에 좀 더 적합한 그리드시스템

https://philipwalton.github.io/solved-by-flexbox/demos/grids/

```
/* Basic Grid Styles (기본 그리드 스타일) */
.Grid { display: flex; }
.Grid-cell { flex: 1; }

/* Grid Style Modifiers (그리드 수정, gutter, padding등) */
.Grid--gutters { margin: -1em 0 0 -1em; }  /* With gutters */
.Grid--gutters > .Grid-cell { padding: 1em 0 0 1em; }

/* Alignment per row (flex-box 전체적인 정렬) */
.Grid--top { align-items: flex-start; }
.Grid--bottom { align-items: flex-end; }
.Grid--center { align-items: center; }

/* Alignment per cell (각각의 cell 정렬) */
.Grid-cell--top { align-self: flex-start; }
.Grid-cell--bottom { align-self: flex-end; }
.Grid-cell--center { align-self: center; }

/* 기본적인 그리드시스템 */
.Grid--fit > .Grid-cell { flex: 1; }
.Grid--full > .Grid-cell { flex: 0 0 100%; }
.Grid--1of2 > .Grid-cell { flex: 0 0 50%; }
.Grid--1of3 > .Grid-cell { flex: 0 0 33.3333%; }
.Grid--1of4 > .Grid-cell { flex: 0 0 25%; }
```

2. Holy Grail Layout ==> 일반적인 헤더, 메인(네비, 내용, 광고), 푸터 가 존재하는 레이아웃

https://philipwalton.github.io/solved-by-flexbox/demos/holy-grail/

모바일 버전에서는 아래의 순서대로 order속성이 부여되어야 한다.

1.header 2.nav 3.main 4.aside 5.footer

3. input Add-ons ==> input태그와 버튼 또는 다른 item을 동적, 반응형 레이아웃에서 딱 붙여서 Add-on 하는 방법

```
<div style="display: flex;">
    <span class="item"></span>
    <input style="flex: 1;" />    <!--item, 버튼의 크기를 제외한 나머지영역을 모두 차지함-->
    <button>전송</button>    <!--심지어 높이까지 동일하다-->
</div>
```

4. Media Object ==> 이미지와 글자, 또는 아이콘과 글자 등을 조합해서 사용할 때 나타나는 문제들을 해결

```
<div style="display: flex; align-items: flex-start;">
    <img style="margin-right: 1em;" src="" />    <!--이미지 or 아이콘을 왼쪽 위에 두고-->
    <p style="flex: 1;">...</p>                  <!--나머지 컨텐츠를 오른쪽에 붙이며, 남은 공간을 차지-->
</div>
```

5. Sticky Footer ==> 중앙 컨텐츠가 많이 없더라도 footer가 브라우저의 맨 아래부분에 위치하도록 하는 방법

```
<body style="display: flex; flex-direction: column; min-height: 100vh;">    <!--flex를 세로방향으로-->
    <header>...</header>
    <main style="flex: 1;">...</main>
    <!--자동으로 header와 footer의 height를 제외한 나머지를 main이 차지한다.-->
    <footer>...</footer>
</body>
```

6. Vertical Centering ==> 세로 가운데정렬

```
<div style="display: flex; align-items: center; justify-content: center;">
    <div style="max-width: 50%; align-self: flex-start;">상단정렬(자식단독)</div>
    <div style="max-width: 50%;">가운데정렬(부모영향)</div>
    <div style="max-width: 50%; align-self: flex-end;">하단정렬(자식단독)</div>
</div>
```

7. 박스 내에서 완벽한 가로세로 중앙정렬

```
<div style="display: flex; height: 300px; width: 300px;">
    <div style="margin: auto">중앙정렬</div>    <!-- Magic! -->
</div>
```

8. flex를 활용한 간단한 Navigation bar

http://codepen.io/HugoGiraudel/pen/pkwqH

9. Prefixing Flexbox (최대한 많은 브라우저에서 지원하게 하려면 아래의 코딩을 따르자.)

```
@mixin flexbox() {
  display: -webkit-box;
```

```
  display: -moz-box;
  display: -ms-flexbox;
  display: -webkit-flex;
  display: flex;
}

@mixin flex($values) {
  -webkit-box-flex: $values;
  -moz-box-flex:  $values;
  -webkit-flex:  $values;
  -ms-flex:  $values;
  flex:  $values;
}

@mixin order($val) {
  -webkit-box-ordinal-group: $val;
  -moz-box-ordinal-group: $val;
  -ms-flex-order: $val;
  -webkit-order: $val;
  order: $val;
}

.wrapper {
  @include flexbox();
}

.item {
  @include flex(1 200px);
  @include order(2);
}
```

Note...

- Flex item 에서는 float, clear, vertical-align 속성은 동작하지 않는다. (no effect)
- Flex-Box를 활용함에 있어서 버그가 발생한다면, 아래의 링크를 참조해 보자.

https://github.com/philipwalton/flexbugs

made by Rex