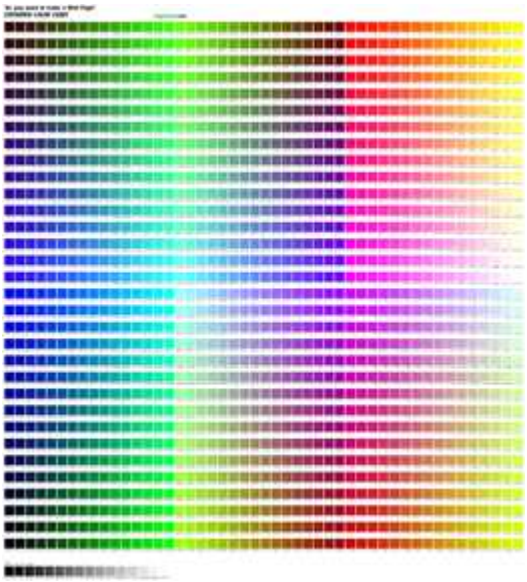
Hex Code



#361999 #78FFF1 #FF6495

RGB(Red Green Blue) 각각의 값을 16진수로 만들어 이어붙인것이다.
CSS 에서 색 이름 대신 Hex Code 사용 가능하다.
추천 검색어: web color trend

```
<div class="font-test">hex code 연습</div>
```

```
/*index.css*/
.font-test {
  color: #FF6495;
}
```

해당 hex code 의 색깔은 다음과 같다.

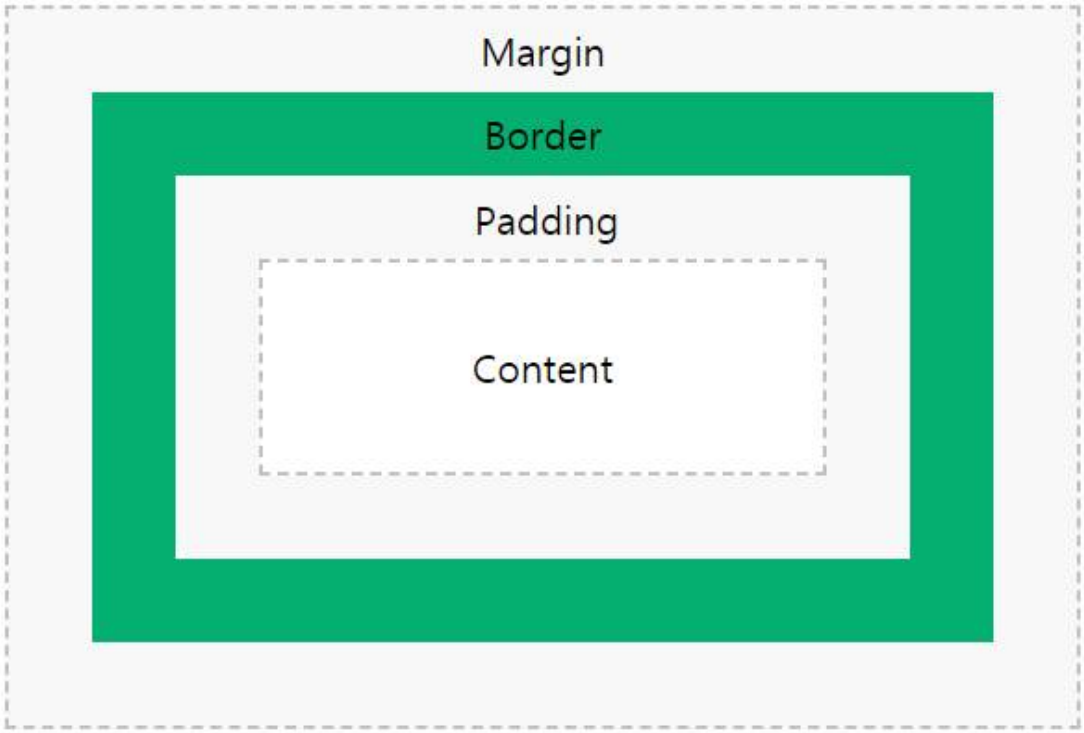
#FF6495

결과:

hex code 연습

박스모델

모든 태그는 박스(box) 로 이루어져있다.
구글 개발자도구 F12로 열어서, element 부분 찍어보면 다음과 같이 나온다.



- border　　태그를 둘러싸는 경계선
- padding　　태그의 컨텐츠와 보더간의 간격
- margin　　태그와 태그간의 간격
  또는, 태그와 화면간의 간격

```html
<body>
    <div class="margin-test">box1</div>
    <div class="margin-test">box2</div>
</body>
```

```css
/*index.css*/
/*box model 연습*/
.margin-test {
  border: 1px solid black;
  /* margin-top: 10px;
  margin-right: 10px;
  margin-bottom: 10px;
  margin-left: 10px; */
  margin: 10px 10px 10px 10px;
}
```

border 는 경계선을 말한다. 뒤엔 순서대로,

- 1px 경계선 굵기
- solid 경계선 형태. solid 외엔 거의 쓰지 않는다
- black 선의 색깔

순서는 꼭 지키지 않아도 상관은 없다. 즉, black 1px solid 라고 써도 잘 동작한다.

border 는 실전에선 주로 테스트할때 사용한다.

- 내 태그가 위치에 맞게 잘 들어갔는지 보는 용도다.

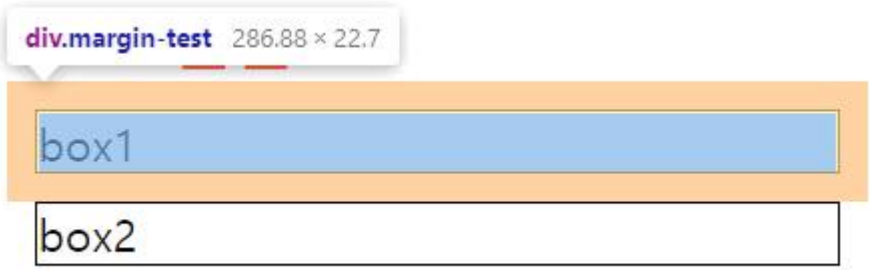
margin의 순서는

- top

right
bottom
left

즉, 시계방향이며, 분리해서(margin-top 등) 쓸수도 있다.

결과:



이 경우, 마진 겹침 현상이 발생하며, 마진이 큰 쪽의 태그를 따른다.

```
<body>
    <div class="padding-test">padding-test</div>
</body>
```

```
/*index.css*/
.padding-test {
  border: 1px solid black;
  padding: 10px 10px 10px 10px;
}
```

결과:

즉, 패딩은 태그의 보더와 컨텐츠간의 간격임을 알 수 있다.

추가로, 태그를 채우는 색을 넣을수도 있다.
이 경우, background-color 를 사용한다.

```
.padding-test {
  border: 1px solid black;
  padding: 10px 10px 10px 10px;
  background-color: tomato;
}
```

결과:

display

해당 태그가 화면에 어떻게 나올지 정의하는 property
모든 태그는 기본적으로 다음 세 가지 중 하나를 가진다

block	가로 전체 차지. 크기 지정 불가능.	, <h1>, …
inline	쓰인 영역만 차지. 크기 지정 불가능	, <a>, …
inline-block	쓰인 영역만 차지하나, 크기 변경 가능	<button>

추가적으로, 다음이 쓰인다.

none	화면에서 아예 안 보이게 만들 때 보통 JavaScript 와 같이 쓰인다
grid	레이아웃 짤 때

block 가로 전체 차지. 크기 지정 가능

```
<div class="divTag">div tag</div>
```

```
/*index.css*/
.divTag {
  border: 1px solid black;
  /* 기본값은 block인데 display는 아래와 같이 변경 가능 */
  /* display: inline; */
  /* block은 크기 변경 가능 */
  /* height: 500px; */
  /* width: 100px; */
}
```

결과:

대표적인 태그는 , <h1>
웹에서 “정해진 영역의” 가로 전체를 차지한다.
디스플레이는 변경 가능하지만, 를 inline 으로 바꾸지는 마라!
왜냐면, 아무 의미없는 태그가 필요한데 inline 으로 쓰고 싶으면
 이 제공되기 때문
block 은 기본적으로 크기 변경이 가능하다.

inline 쓰인 영역만 차지. 크기 지정 불가능

```
<span>span tag</span>
```

```css
/*index.css*/
span {
  border: 1px solid black;
  /* 기본값 */
  /* display: inline; */
  /* inline은 크기 변경 불가능 */
}
```

결과:

span tag

대표적인 태그는 , <a>
쓰인 영역만 차지하며, 크기 지정은 불가능하다.
즉, inline 요소가 필요한, 의미없는 태그가 필요하다면
 대신 을 써야한다!
inline 은 크기를 변경할 수 없다

inline-block 쓰인 영역만 차지하나, 크기 변경 가능

```html
<button>button tag</button>
```

```css
/*index.css*/
button {
  /* 기본값 */
  /* display: inline-block; */
  /* inline-block은 크기 변경 가능 */
  /*width: 100%;*/
  height: 50px;
}
```

결과:

대표적인 태그는 <button>
쓰인 영역만 차지하나, 크기 변경이 가능하다.
width 를 100% 로 지정하면 블록 버튼을 만들 수 있다.

none

```
<div>안보임!</div>
```

```
/*index.css*/
div {
  display: none;
}
```

화면에서 아예 안 보이게 하는 것이다.
보통은 JavaScript 와 결합해서 사용한다.

Position

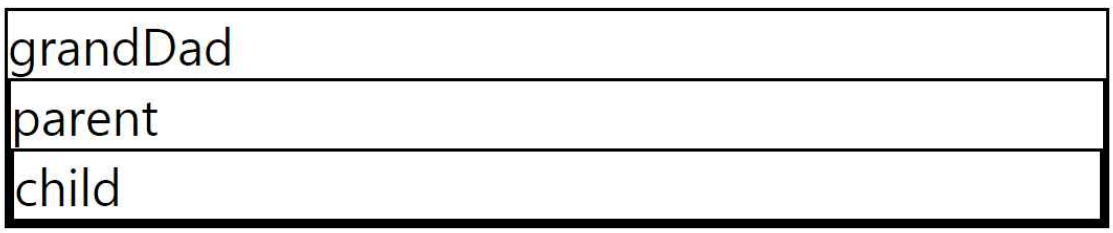
태그의 위치를 옮길 때 사용한다.
단순히 옮기면 되는 간단한 문제가 아니다.
무엇을 기준으로 옮길지 알아야 한다.
그 기준은 부모와 관련있다.

먼저 조부모, 부모, 자식태그가 필요하다.

```
<div class="grandParent">
    grandDad
    <div class="parent">
        parent
        <div class="child">child</div>
    </div>
</div>
```

```
.grandParent,
.parent,
.child {
    border: 1px solid black;
    /*position: static;*/
}
```

결과:



position 의 기본값은 static 이다.
static 일 경우, 움직이는것은 불가능하다.

1. relative

child 부분만 relative 로 바꿔보자.

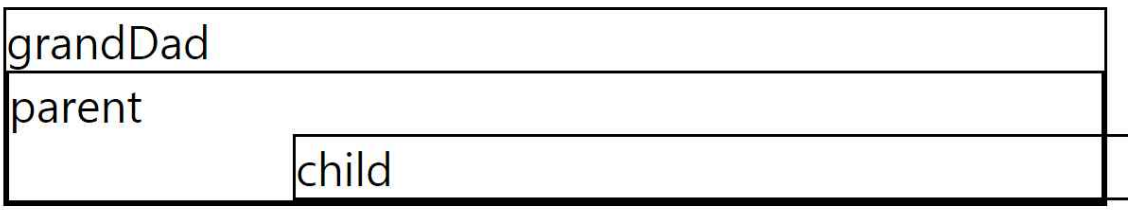
```
.child {
    position: relative;
}
```

이 경우, 당장은 아무런 변화가 없다.

그러나, 여기서 움직임을 주면 변화가 생긴다.
움직임은 top, right, bottom, left 로 지정한다.

```
.child {
    position: relative;
    left: 100px;
}
```

결과:



오른쪽에 튀어나온 걸 보면 알겠지만, 오른쪽으로 이동한 것을 볼 수 있다.
헷갈리지 말자. left 면 오른쪽으로 갈 것이다.
left: 100px 은 왼쪽에 100px의 여백을 둔다는 뜻이다.
중요한 건 기준이다. relative는 부모 태그로부터 자신의 위치를 결정한다.
원래 자기가 있었던 위치에서 이동하는 것이다.

2. absolute

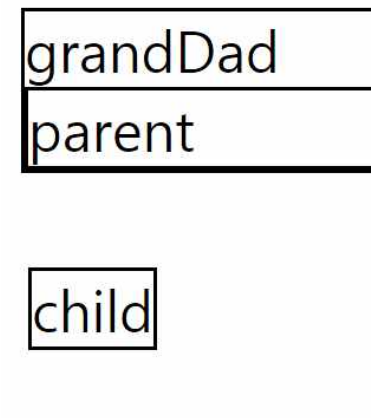
static이 아닌 부모를 기준으로 삼는다.
즉, 부모와 조부모에게 position 을 지정해주지 않았다면
화면 왼쪽 위 모서리가 기준이다.

case1. 부모를 relative 로 지정할 경우
이 경우, 부모가 static이 아니기 때문에 부모가 기준이다.

```css
.parent {
    position: relative;
}

.child {
    position: absolute;
    top: 50px;
}
```

결과:



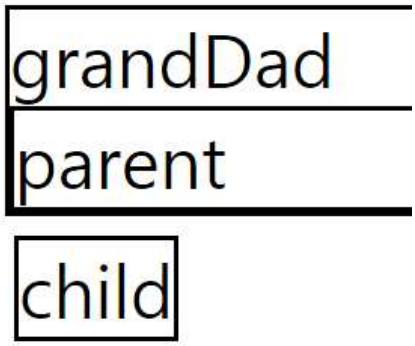
parent 를 기준으로, 아래로 50px 내려왔다.

case2: 부모가 static이면서 조부모가 static이 아닌 경우
이 경우, 부모는 static이고 조부모가 static이 아니기 때문에 조부모가 기준이다.

```css
.grandParent {
    position: relative
}

.child {
    position: absolute;
    top: 50px;
}
```

결과:

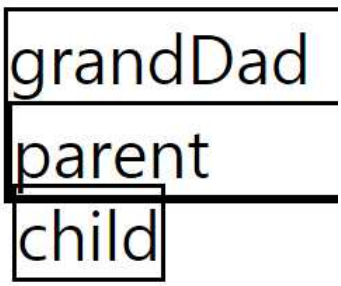


조부모를 기준으로 50px 내려왔다.

case3: 부모에게 아무것도 지정하지 않았을 경우
이 경우, 부모도 static 이고 조부모도 static 이기 때문에 화면 왼쪽 위가 기준이다.

```
.child {
    position: absolute;
    top: 50px;
}
```

결과:



화면 왼쪽 끝을 기준으로 50px 내려왔다.

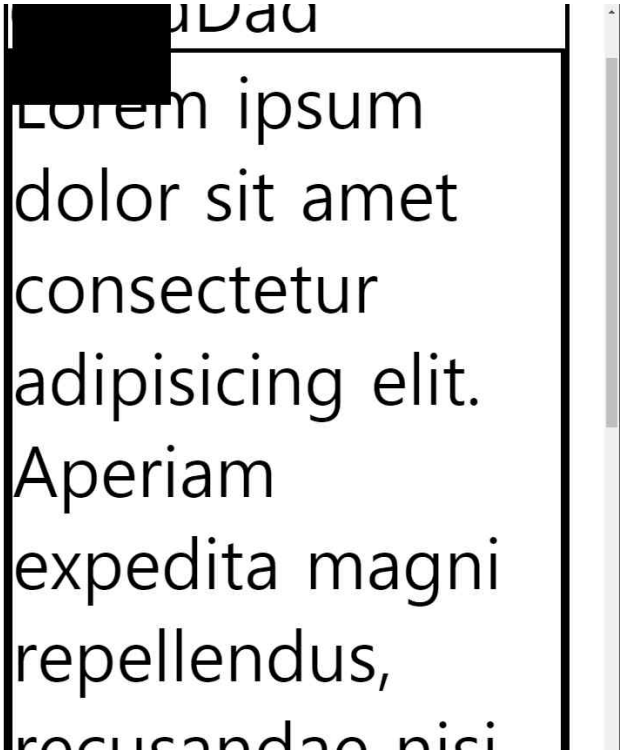
3. fixed

시조 태그의 위치가 기준이며, 스크롤도 무시하고, display는 더이상 block이 아니다.

```
<div class="grandParent">
    grandDad
    <div class="parent">
        Lorem ipsum dolor sit amet consectetur
        adipisicing elit. Aperiam expedita magni
        repellendus, recusandae nisi
        ad perspiciatis dolor nulla? Et distinctio
        corrupti architecto debitis perferendis
        voluptatibus a, dolorum
        saepe rem nisi?
        <div class="child">child</div>
    </div>
</div>
```

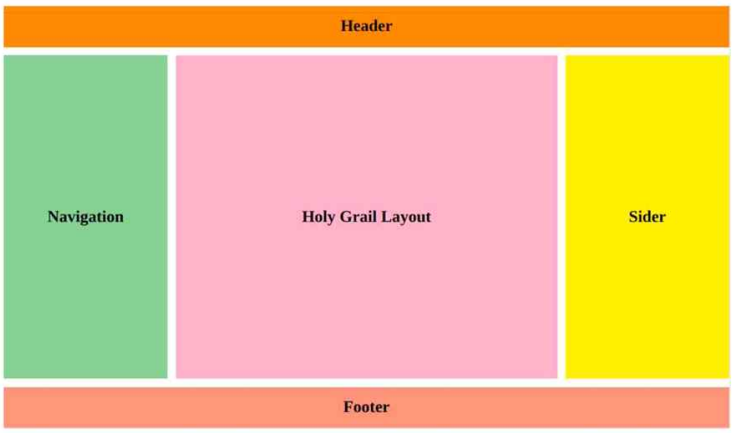
```
.child {
    position: fixed;
    top: 0px;
    background-color: black;
}
```

결과:



구분을 위해 background-color 를 black 으로 두었다.
시조 태그를 기준으로 0px로 두었다. 스크롤을 내려도 움직이지 않는다.

레이아웃



다음과 같은 레이아웃을 마치 성배와 같다고 해서, Holy Grail Layout 이라고 한다.
성배는 예수님이 마셨던 잔을 의미하는데, 이렇게 레이아웃 만들기가
마치 성배 찾는것과 같이 굉장히 어려웠기 때문.

왜? 일단 비율이 문제다.
웹은 해상도가 정해져있는 상태에서 제공되는 프로그램이 아니다.
그러면 각 요소들을 일정한 비율로 나누거나,
특정한 크기로 고정시킬 필요가 있는데
CSS 에선 그것이 상당히 어려웠다.
그러나, CSS 에서 grid 라는 기술이 정식 채택된 이후 매우 쉬워졌다.

우선, 다음과 같은 코드를 입력하자.

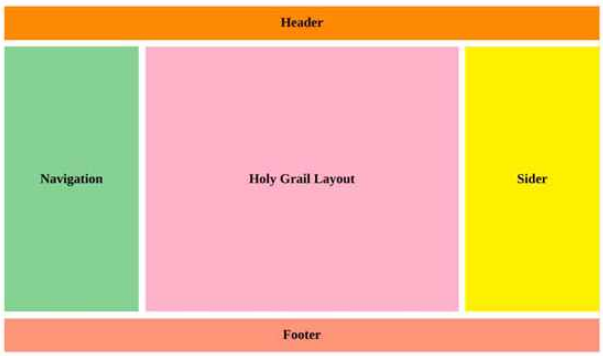
```
<div class="header">header</div>
<div class="nav">nav</div>
<div class="main">main</div>
<div class="aside">aside</div>
<div class="footer">footer</div>
```

```
.header,
.nav,
.main,
.aside,
.footer {
    border: 1px solid black;
}
```

결과:

header
nav
main
aside
footer

여기서, 우리가 만들고 싶은 형태는 앞서 언급했던 Holy grail layout 이다.



그렇다면, nav, main, aside 부분만 옆으로 늘어뜨리면 될 것이다.

grid 는 부모 태그에서 사용하기 때문에, 이 셋을 감싸는 부모태그를 만들어주겠다.

```
<div class="header">header</div>
<div class="container">
    <div class="nav">nav</div>
    <div class="main">main</div>
    <div class="aside">aside</div>
</div>
<div class="footer">footer</div>
```

그리고, 다음과 같은 css 를 입력하자.

```
.container {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
}
```

우선, 결과는 다음과 같다.

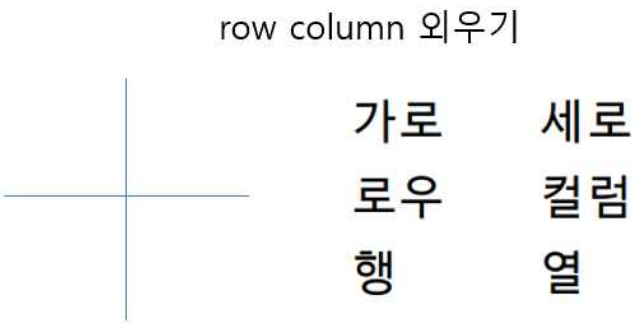
header		
nav	main	aside
footer		

우리가 원하는 형태와 얼추 비슷해진것을 확인가능하다.
브라우저 화면 크기를 조절해도, 정확하게 1대 1대 1의 비율로 떨어짐을 알 수 있다.

즉, grid는 부모태그에서 자식에게 지정하는 크기이다.
fr은 각 자식의 비율을 의미한다.
여기선 1:1:1 의 비율을 주었다.
fr대신 px을 사용하면 해당 크기로 고정이 될 것이다.

그리고, grid-template-columns 라고 쓴 것은, 각 요소를 컬럼 형식으로 늘어뜨리는것을 의미한다.

row 와 column 은 은근히 헷갈리는 개념일 수 있는데, 다음과 같이 손으로 십자가를 그리면서 외우자.

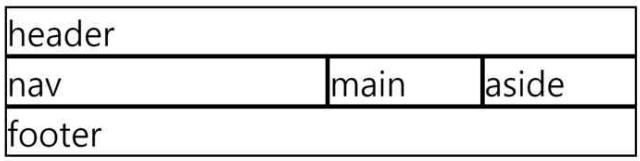


여섯번 말하면 된다.
가로 세로 로우 컬럼 행 열
다시 반복
가로 세로 로우 컬럼 행 열

만약, fr 대신 px을 쓰면 어떤 현상이 일어날까?

```
.container {
    display: grid;
    grid-template-columns: 150px 1fr 1fr;
}
```

결과:



즉, nav 의 너비가 150px로 고정되고
main과 aside 는 남은 비율에서 1:1 을 각각 차지한다.
브라우저 화면 크기를 조절해도 이 결과는 동일하게 유지된다.

다음과 같이 해보면 어떨까?

```css
.container {
    display: grid;
    grid-template-columns: 1fr 2fr 1fr;
}
```

이 경우, 셋의 비율은 1:2:1 이다.

결과:

header		
nav	main	aside
footer		

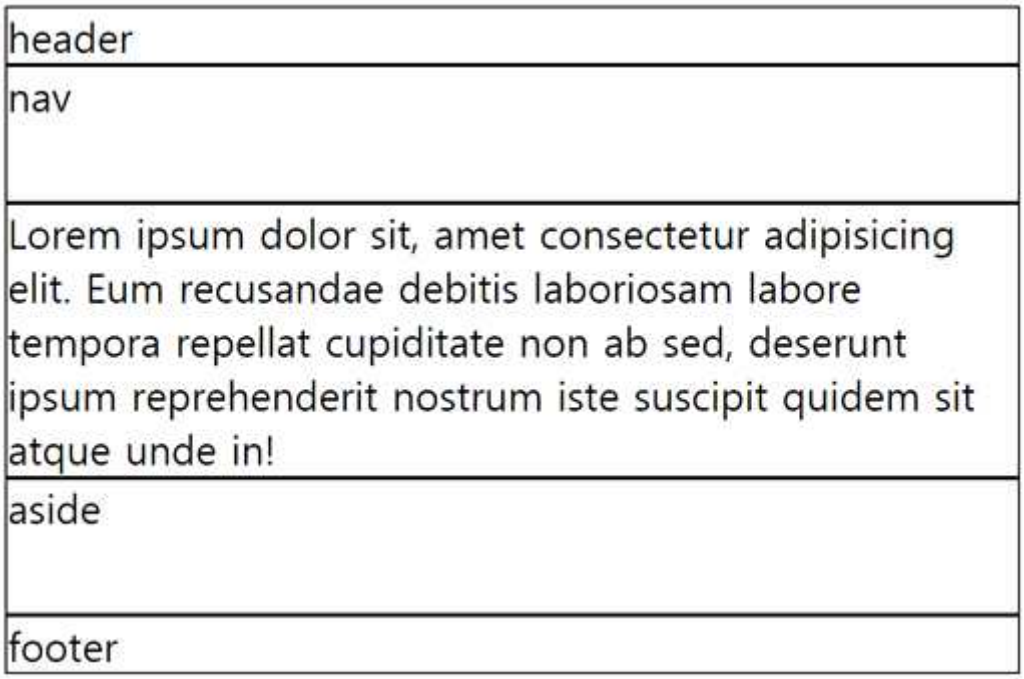
브라우저 화면 크기를 조절해도 이 결과는 동일하게 유지된다.

column이 있다면 row 도 있을 것이다. 다음과 같이 써보자.

```html
<div class="header">header</div>
<div class="container">
    <div class="nav">nav</div>
    <div class="main">
        Lorem ipsum dolor sit, amet consectetur
        adipisicing elit. Eum recusandae debitis
        laboriosam labore tempora repellat cupiditate
        non ab sed, deserunt ipsum reprehenderit
        nostrum iste suscipit quidem sit
        atque unde in!
    </div>
    <div class="aside">aside</div>
</div>
<div class="footer">footer</div>
```

```
.container {
    display: grid;
    grid-template-rows: 1fr 2fr 1fr;
}
```

이 경우, 1:2:1 의 비율을 가지는데, rows 로 썼으니 열이 아니라 행으로 나뉠 것이다.



화면 크기 조절해 확인해보자.