구문과 기본 서식

- 탭 대신 스페이스 두 칸 (2) 들여쓰기
- 이상적인 행의 너비는 80 글자
- 적절하게 쓰인 여러 행의 CSS 규칙
- 공백의 의미 있는 사용

```
// Yep(올바른 예)
.yep {
  display: block;
  overflow: hidden;
  padding: 0 1em;
}

// Nope(잘못된 예)
.nope {
    display: block; overflow: hidden;

    padding: 0 1em;
}
```

인코딩

- 문자 인코딩과 관련한 잠재적인 문제를 피하기 위해서는, 메인 스타일시트에서 @charset 지시어를 사용해 UTF-8 인코딩을 강제하는 것이 강력하게 권장된다.
- @charset 지시어가 스타일시트의 가장 첫 번째 요소이고 어떤 종류의 문자도 앞에 오지 않도록 해야 한다.

```
@charset 'utf-8';
```

문자열

- Sass에서 문자열은 언제나 작은 따옴표(')로 감싸져야 한다.
  - 색 이름은 따옴표가 없으면 색으로 취급되는데, 이는 심각한 문제로 이어질 수 있다.
  - 대부분의 구문 강조기는 따옴표 없는 문자열을 지원하지 못할 것이다.
  - 전반적인 가독성에 도움이 된다.
  - 문자열을 따옴표로 감싸지 않을 적절한 이유가 없다.

```
// Yep
$direction: 'left';

// Nope
$direction: left;
```

CSS 값인 문자열

- initial이나 sans-serif 같은 특정 CSS 값은 따옴표로 싸여서는 안된다.
- Font-family: ' sans-serif ' 같은 선언의 경우 CSS는 인용부호가 붙은 문자열이 아니라 식별자를 기대하고 있기 때문에 아무런 경고도 없이 작동하지 않으므로, 그런 값들은 따옴표로 감싸지 않는다.

```
// Yep
$font-type: sans-serif;

// Nope
$font-type: 'sans-serif';

// Okay I guess
$font-type: unquote('sans-serif');
```

- 위의 예처럼 CSS 값(CSS 식별자)으로 사용될 문자열과 맵 키와 같은 Sass 자료 유형에 쓰일 문자열을 구별할 수 있다.

따옴표를 포함한 문자열

- 만약 문자열이 하나 혹은 여러 개의 작은 따옴표를 포함하고 있다면?
- 문자열 안에서 과도한 문자 이스케이프를 피하기 위해 대신 큰 따옴표(")로 문자열을 감싸는 것을 고려할 수 있다.

```
// Okay
@warn 'You can\'t do that.';

// Okay
@warn "You can't do that.";
```

URL

- 따옴표로 감싸여야 한다.

```
// Yep
.foo {
  background-image: url('/images/kittens.jpg');
}

// Nope
.foo {
  background-image: url(/images/kittens.jpg);
}
```

참고

- All You Ever Need to Know About Sass Interpolation
- SassyStrings

숫자 - 영

- 숫자는 1보다 작은 소수 앞에 앞장서는 영을 표기해야 한다.
- 뒤따르는 영은 절대 표기하지 말 것!

```
// Yep
.foo {
  padding: 2em;
  opacity: 0.5;
}

// Nope
.foo {
  padding: 2.0em;
  opacity: .5;
}
```

단위

- 길이를 다룰 때, 0 값은 절대로 단위를 가져선 안 됨!

```
// Yep
$length: 0;

// Nope
$length: 0em;
```

- 단위를 숫자에 붙이기 위해서는, 이 숫자에 *1 단위*를 곱해야 한다.

```
$value: 42;

// Yep
$length: $value * 1px;

// Nope
$length: $value + px;
```

단위 제거

- 값의 단위를 제거하기 위해서는, *그 종류의 한 단위*로 나누어야 한다.

```
$length: 42px;

// Yep
$value: $length / 1px;

// Nope
$value: str-slice($length + unquote(''), 1, 2);
```

- 단위를 문자열로서 숫자에 덧붙이면 결과물은 문자열이 되며, 그 값으로 더이상 연산을 할 수 없다.
- 숫자의 숫자 부분을 단위에서 잘라내면 그 결과 역시 문자열이 된다.

연산

- **최상위 숫자 계산은 언제나 괄호로 감싸져야 한다.**
- 이 요건은 가독성을 향상시킬 뿐만 아니라, Sass가 괄호 안의 수치를 계산하도록 강제함으로써 일부 예외적인 상황을 방지할 수 있다.

```
// Yep
.foo {
  width: (100% / 3);
}

// Nope
.foo {
  width: 100% / 3;
}
```

매직 넘버

- "매직 넘버"는 *익명의 숫자 상수*를 일컫는 전통적인 프로그래밍 용어로, 이 숫자는 어쩌다보니 *맞아떨어지지만* 어떤 논리적인 설명과도 관련되지 않은 임의의 숫자이다.
- **매직 넘버는 역병 같은 존재이며 무슨 수를 써서라도 피해야 합니다**.
  - 왜 매직넘버가 효과를 내는지에 대한 합리적인 설명을 찾을 수 없을 때는, 어떻게 거기에 도달했고 왜 효과를 낸다고 생각하는지를 설명하는 충분한 주석을 기술해야 다음 개발자에게 더 도움이 된다.

```
/**
 * 1. 매직 넘버. `.foo`의 상단을 부모에 맞춰 정렬시키기 위해 찾은 값 중 가장
 * 낮은 값이다. 가능하다면, 적절하게 고쳐야 할 것.
 */
.foo {
  top: 0.327em; /* 1 */
}
```

참고

- Use Lengths, Not Strings
- Correctly Adding Unit to Number
- Magic Numbers in CSS
- Sassy-Math

색

- 색은 CSS 에서 매우 중요하므로 Sass는 몇 가지의 강력한 함수를 제공한다.
- 색을 가능한 한 간단하게 만들기 위한 우선순위

1. CSS 색 키워드
2. HSL 표기법
3. RGB 표기법
4. 16진법 표기법.
   가급적 소문자로,
   가능한 경우 단축형으로 표기

```
// Nope
.foo {
  color: #FF0000;
}

// Nope
.foo {
  color: red;
}
```

```
// Yep
.foo {
  color: hsl(0, 100%, 50%);
}

// Also yep
.foo {
  color: rgb(255, 0, 0);
}

// Meh
.foo {
  color: #f00;
}
```

색

- HSL이나 RGB 표기를 사용할 때, 쉼표(,) 뒤에는 언제나 스페이스 한 칸을 더하고 괄호((,))와 내용 사이에는 스페이스를 넣지 않는다.

```
// Yep
.foo {
  color: rgba(0, 0, 0, 0.1);
  background: hsl(300, 100%, 100%);
}

// Nope
.foo {
  color: rgba(0,0,0,0.1);
  background: hsl( 300, 100%, 100% );
}
```

색과 변수

- 색을 한 번 이상 사용할 때는 색을 대변하는 의미 있는 이름을 붙여 변수에 저장하세요.

```
$sass-pink: hsl(330, 50%, 60%);
```

- 만약 변수의 용도가 테마와 깊은 관련이 있다면, 그 변수가 어떻게 사용되어야 하는지 설명하는 이름을 붙여 다른 변수에 저장하여 사용한다.

```
$main-theme-color: $sass-pink;
```

색 명암 조절

- lighten과 darken 두 함수는 HSL 공간에서 색의 명도를 증감하여 조정한다.
- 기본적으로, 이들은 adjust-color 함수의 $lightness 매개 변수의 가명일 뿐이다.

- 문제
  - 이들 함수가 가끔 기대되는 결과를 제공하지 않는다

- mix 함수
  - 색을 white나 black과 혼합함으로써 명암을 조절하는 좋은 방법이다.

색 명암 조절 – mix 함수

- 색의 비율을 감소시킴에 따라 점진적으로 검은 색(혹은 흰 색)으로 나아간다.
- darken과 lighten은 색을 순식간에 완전한 검은 색이나 흰 색으로 보내버린다.

- lighten/darken 과 mix 사이의 차이

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
lighten()	#d379a6	#dfa0bf	#ecc6d9	#f9eaf2						
mix() w/ white	#cb6497	#d175a3	#d786ae	#dc97ba	#e2a9c5	#e8bad1	#edcbdc	#f3dce8	#f6edf2	
darken()	#ad3972	#862d59	#602040	#3a1326	#140606					
mix() w/ dark	#b24a7e	#9e4270	#8a3a62	#763154	#632946	#4f2138	#3b182a	#27101b	#130f80d	

	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
lighten()	#ff7f33	#ff9f66	#ffbf99	#ffdfcc						
mix() w/ white	#ff6f19	#ff7f33	#ff8f4c	#ff9f66	#ffaf7f	#ffbf99	#ffcfb2	#ffdfcc	#ffefe5	
darken()	#cc4c00	#993900	#662600	#331300						
mix() w/ dark	#e55500	#cc4c00	#b24200	#993900	#7f2f00	#662600	#4c1c00	#321200	#190900	

tint 와 shade 함수

- 매번 mix 함수를 쓰지 않는다면, tint와 shade 함수를 만들어 쓸 수 있다.

```
/// 색을 약간 밝게 한다
/// @access public
/// @param {Color} $color - 밝게 만들려는 색
/// @param {Number} $percentage - 반환될 색 내 `$color`의 백분율
/// @return {Color}
@function tint($color, $percentage) {
  @return mix(white, $color, $percentage);
}

/// 색을 약간 어둡게 한다
/// @access public
/// @param {Color} $color - 어둡게 만들려는 색
/// @param {Number} $percentage - 반환될 색 내 `$color`의 백분율
/// @return {Color}
@function shade($color, $percentage) {
  @return mix(black, $color, $percentage);
```

참고

- A Visual Guide to Sass & Compass Color Functions
- How to Programmatically Go From One Color to Another
- Sass Color Variables That Don't Suck
- Using Sass to Build Color Palettes
- Dealing with Color Schemes in Sass

리스트

- 리스트는 Sass에서 배열에 상당하는 개념이다.
- 리스트는 어떤 타입의 값이든 저장할 수 있게 의도된 평면적인 데이터 구조이다.
  - 맵과는 다르다.
- 리스트 가이드라인
  1. 한 줄 혹은 여러 줄로 표기한다.
  2. 80자 줄에 안 들어갈 정도로 길면 반드시 여러 줄에 표기한다.
  3. CSS 상에서 그대로 사용되지 않는 한, 언제나 쉼표로 분리한다.
  4. 언제나 괄호로 감싼다.
  5. 여러 줄인 경우 뒤따르는 쉼표를 붙이고, 한 줄인 경우 제외한다.

리스트

```
// Yep
$font-stack: ('Helvetica', 'Arial', sans-serif);

// Yep
$font-stack: (
  'Helvetica',
  'Arial',
  sans-serif,
);

// Nope
$font-stack: 'Helvetica' 'Arial' sans-serif;

// Nope
$font-stack: 'Helvetica', 'Arial', sans-serif;

// Nope
$font-stack: ('Helvetica', 'Arial', sans-serif,);
```

리스트

- 리스트에 새로운 아이템을 추가할 때는, 제공된 API를 이용한다.
- 수동으로 새로운 아이템을 추가하지 않는다.

```
$shadows: (0 42px 13.37px hotpink);

// Yep
$shadows: append($shadows, $shadow, comma);

// Nope
$shadows: $shadows, $shadow;
```

참고

- Understanding Sass lists
- SassyLists

맵

- Sass 3.3부터 스타일시트 작성자는 맵을 정의할 수 있다.
- 이는 연관 배열, 해시 혹은 JavaScript 오브젝트에 해당하는 Sass 용어이다.
- 맵은 키를 모든 유형의 값과 연결하는 자료 구조이다.
- 키는 어떤 자료 유형도 될 수 있다.
  - 추천하지는 않지만 맵도 포함

맵

- 맵 작성방법
  1. 콜론(:) 다음에는 빈 칸(스페이스)을 입력한다.
  2. 여는 괄호 (()는 콜론(:)과 같은 줄에 표기한다.
  3. (99%의 경우에 해당하는) 문자열인 키는 따옴표로 감싼다.
  4. 각각의 키/값 쌍은 새 줄을 차지한다.
  5. 각 키/값 뒤에는 쉼표(,)를 표기한다.
  6. 추가, 제거 혹은 순서를 바꾸기 쉽도록 마지막 아이템 뒤에 따라오는 쉼표(,)를 표기한다.
  7. 닫는 괄호())는 새 줄에 표기한다.
  8. 닫는 괄호())와 세미콜론(;) 사이에는 스페이스나 새 줄을 넣지 않는다.

맵

```
// Yep
$breakpoints: (
  'small': 767px,
  'medium': 992px,
  'large': 1200px,
);

// Nope
$breakpoints: ( small: 767px, medium: 992px, large: 1200px );
```

SASS 맵 디버그

- Sass 맵에서 일어나고 있는 건지 알 수 없어 헤매는 스스로를 발견하게 된다면?

```
@mixin debug-map($map) {
  @at-root {
    @debug-map {
      __toString__: inspect($map);
      __length__: length($map);
      __depth__: if(function-exists('map-depth'), map-depth($map), null);
      __keys__: map-keys($map);
      __properties__ {
        @each $key, $value in $map {
          #{'(' + type-of($value) + ') ' + $key}: inspect($value);
        }
      }
    }
  }
}
```

SASS 맵 디버그

- 맵의 깊이를 알고 싶으면 아래 함수를 추가하면, 복잡한 믹스인이 자동으로 값을 표시해 준다.

```
/// 맵의 최대 깊이를 계산한다
/// @param {Map} $map
/// @return {Number} `$map`의 최대 깊이
@function map-depth($map) {
  $level: 1;

  @each $key, $value in $map {
    @if type-of($value) == 'map' {
      $level: max(map-depth($value) + 1, $level);
    }
  }

  @return $level;
}
```

참고

- Using Sass Maps
- Debugging Sass Maps
- Extra Map functions in Sass
- Real Sass, Real Maps
- Sass Maps are Awesome
- Sass list-maps
- Sass Maps Plus
- Sassy-Maps
- Introduction to Sass Maps Usage and Examples

CSS 기본 규칙

① 관련된 선택자는 같은 줄에 표기한다.
② 관련 없는 선택자는 새 줄에 표기한다.
③ 여는 중괄호({)는 마지막 선택자와 하나의 빈 칸을 둔다.
④ 각각의 선언은 저마다 새 줄을 차지한다.
⑤ 콜론(:) 뒤에는 스페이스 한 칸을 둔다.
⑥ 모든 선언의 끝은 세미콜론(;)으로 마무리한다.
⑦ 닫는 중괄호(})는 새 줄을 차지한다.
⑧ 닫는 중괄호(}) 뒤에 비어 있는 새 줄을 만든다.

```
// Yep
.foo, .foo-bar,
.baz {
  display: block;
  overflow: hidden;
  margin: 0 auto;
}

// Nope
.foo,
.foo-bar, .baz {
    display: block;
    overflow: hidden;
    margin: 0 auto }
```

CSS 추가 규칙

① 지역 변수는 어떤 선언보다 먼저 선언되어야 하며, 새 줄 하나로 다른 선언들과 간격을 둔다;

② @content가 없는 믹스인 호출은 다른 선언보다 앞에 위치한다.

③ 내포된 선택자는 언제나 새 줄 뒤에 온다.

④ @content를 가진 믹스인 호출은 내포된 선택자보다 뒤에 위치한다.

⑤ 닫는 중괄호(}) 앞에는 새 줄이 없어야 한다.

```
.foo, .foo-bar,
.baz {
  $length: 42em;

  @include ellipsis;
  @include size($length);
  display: block;
  overflow: hidden;
  margin: 0 auto;

  &:hover {
    color: red;
  }

  @include respond-to('small') {
    overflow: visible;
  }
}
```

참고

- Anatomy of a Ruleset

선언 정렬 – 알파벳 정렬

- 장점
  - 보편적이다.
- 단점
  - bottom과 top 같은 속성들이 서로 붙어있지 않아 이상하다.
    - 왜 애니메이션이 디스플레이 유형보다 먼저 나와야 하는가?

```
.foo {
  background: black;
  bottom: 0;
  color: white;
  font-weight: bold;
  font-size: 1.5em;
  height: 100px;
  overflow: hidden;
  position: absolute;
  right: 0;
  width: 100px;
}
```

선언 정렬 – 유형별 정렬

- 장점
  - 아주 타당하다.
    - 모든 폰트 관련 선언들이 한데 모이고, top과 bottom은 재결합하며 규칙들을 보면 마치 짧은 이야기를 읽는 느낌이 들 수 있다.
- 단점
  - 다양하게 해석될 여지가 있다.
    - white-space는 어디로 가야 할까: 폰트 혹은 디스플레이?
    - overflow는 정확히 어디에 속할까?
    - 그룹 내에서 속성들의 순서는 어떻게 되어야 할까?

```
.foo {
  height: 100px;
  width: 100px;
  overflow: hidden;
  position: absolute;
  bottom: 0;
  right: 0;
  background: black;
  color: white;
  font-weight: bold;
  font-size: 1.5em;
}
```

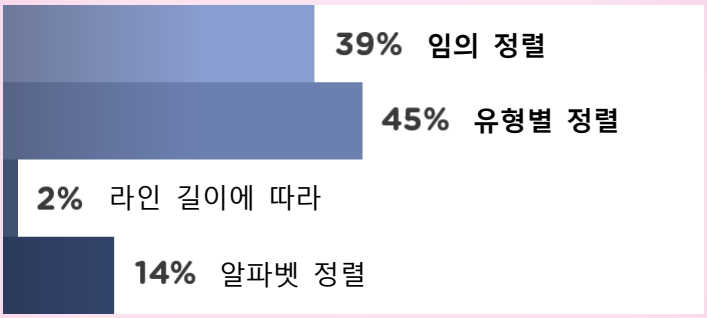
선언 정렬 – 유형별 정렬(Concentric CSS)

- Concentric CSS
  - 유형별 정렬의 하위 갈래로, 순서를 정의하기 위해 박스 모델에 의존한다.
    - 바깥쪽에서 출발해서, 안쪽으로 들어오는 정렬 구조

```
.foo {
  width: 100px;
  height: 100px;
  position: absolute;
  right: 0;
  bottom: 0;
  background: black;
  overflow: hidden;
  color: white;
  font-weight: bold;
  font-size: 1.5em;
}
```

선언 정렬 – 결론

- CSS Tricks에서의 설문조사



- 2014년 프론트엔드 개발자인 Pete Schuster의 조사에 따르면 CSS Comb(유형별 정렬)를 사용한 CSS 선언 정렬이 Gzip 압축 시 평균 파일 크기를 2.7% 줄인다는 결과를 보여준다.
  - 알파벳순으로 정렬했을 때에는 1.3%가 줄어들었다.

참고

- CSS Comb
- Concentric CSS
- Idiomatic CSS
- On Declaration Sorting
- Reduce File Size With CSS Sorting
- Poll Results: How Do You Order Your CSS Properties?

선택자 내포(Nesting; 중첩) - 정의

- Sass가 제공하는 기능 중 많은 개발자들에 의해 **심하게 남용**되고 있는 것 중 하나이다.
- 선택자 내포는 짧은 선택자들을 서로 포개어 넣음으로써 긴 선택자를 산출해 내는 방식을 제안한다.

```
.foo {
  .bar {
    &:hover {
      color: red;
    }
  }
}
```

→

```
.foo .bar:hover {
  color: red;
}
```

```
.foo {
  &-bar {
    color: red;
  }
}
```

→

```
.foo-bar {
  color: red;
}
```

선택자 내포(Nesting; 중첩) - 주의

- BEM 작명 관례와 함께 .block__element와 .block__modifier 선택자를 원래 선택자(.block)에 기반하여 생성하는 데 사용된다.

- 문제점
  - 선택자가 길어지고 현재 선택자(&)를 더 자주 인용할수록 코드를 읽기 어렵게 만든다.
  - 몇 가지 예외를 두고 가능한 한 선택자 내포는 피해야 한다.

선택자 내포(Nesting; 중첩) - 추천

① 가상 클래스와 가상 요소를 내포할 때

```
.foo {
  color: red;

  &:hover {
    color: green;
  }

  &::before {
    content: 'pseudo-element';
  }
}
```

선택자 내포(Nesting; 중첩) - 추천

② .is-active 같은 컴퍼넌트에 독립적인 상태 클래스를 사용할 때

```
.foo {
  // …

  &.is-active {
    font-weight: bold;
  }
}
```

선택자 내포(Nesting; 중첩) - 추천

▪ 요소를 꾸밀 때

```
.foo {
  // …

  .no-opacity & {
    display: none;
  }
}
```

참고

- Beware of Selector Nesting
- The Inception Rule
- Avoid nested selectors for more modular CSS

작명 관례

- CSS Guidelines가 추천하는 방법을 참고하되, 코드베이스 전체가 일관되며 읽기 쉽도록 이름을 잘 짓는 것이 중요하다.
  - 변수
  - 함수
  - 믹스인
- **하이픈으로 구분된 소문자**, 그리고 무엇보다도 의미 있는 이름이어야 한다.

```
$vertical-rhythm-baseline: 1.5rem;

@mixin size($width, $height: $width) {
  // …
}

@function opposite-direction($direction) {
  // …
}
```

참고

- CSS Guidelines' Naming Conventions

상수

- 많은 언어들의 경우처럼, 상수에 대해 모두 대문자로 된 스네이크 케이스 변수를 권장한다.

```
// Yep
$CSS_POSITIONS: (top, right, bottom, left, center);

// Nope
$css-positions: (top, right, bottom, left, center);
```

참고

- Dealing With Constants in Sass

네임스페이스

- 다른 사람의 코드와 충돌하지 않도록 구분하는 이름 (예: gt-)

```
$gt-configuration: ( … );

@function gt-rainbow($unicorn) {
  // …
}
```

참고

- Please Respect the Global CSS Namespace

주석

- 파일의 구조와 역할
- 규칙의 목적
- 매직 넘버의 배후에 있는 생각
- CSS 선언에 대한 이유
- CSS 선언의 정렬
- 진행 방식의 배후에 있는 사고 과정

주석

```
/**
 * 너무 길어서 한 줄에 안 들어가는 문자열을 자르고 말줄임표를 붙이는 헬퍼 클래스.
 * 1. 줄바꿈을 방지하고, 한 줄로 유지되도록 강제한다.
 * 2. 줄 끝에 말줄임표를 붙인다.
 */
.ellipsis {
  white-space: nowrap; /* 1 */
  text-overflow: ellipsis; /* 2 */
  overflow: hidden;
}
```

```
// 현재 모듈을 불러온 모듈 리스트에 더한다.
// 전역 변수를 업데이트하도록 하기 위해 `!global` 플래그가 필요함.
$imported-modules: append($imported-modules, $module) !global;
```

참고

- CSS Guidelines' Commenting section

문서화

- 코드베이스 전역에서 사용되도록 만들어진 모든 변수, 함수, 믹스인, 플레이스홀더는 SassDoc을 이용하여 전역 API의 일부로서 문서화되어야 한다.

```
/// 코드 베이스 전역에서 사용되는 버티컬 리듬 베이스라인.
/// @type Length
$vertical-rhythm-baseline: 1.5rem;
```

- SassDoc 의 역할
  - 공개 혹은 비공개 API의 일부인 모든 것에 대해 주석 기반의 시스템을 이용하는 표준화된 주석을 강제한다.
  - SassDoc의 엔드포인트(CLI tool, Grunt, Gulp, Broccoli, Node...)를 이용하여 API 문서의 HTML 버전을 생성할 수 있다.

문서화

```
/// `width`와 `height`를 동시에 정의하도록 돕는 믹스인.
///
/// @author Hugo Giraudel
///
/// @access public
///
/// @param {Length} $width - 요소의 `width`
/// @param {Length} $height [$width] - 요소의 `height`
///
/// @example scss - 사용법
///   .foo {
///     @include size(10em);
///   }
///
///   .bar {
///     @include size(100%, 10em);
///   }
///
```

```
/// @example css - CSS 아웃풋
///   .foo {
///     width: 10em;
///     height: 10em;
///   }
///
///   .bar {
///     width: 100%;
///     height: 10em;
///   }
@mixin size($width, $height: $width) {
  width: $width;
  height: $height;
}
```

SassDoc으로 문서화된 믹스인 예

참고

- SassDoc
- SassDoc: a Documentation Tool for Sass

설계 패턴

- 7-1 패턴
  - base/
  - components/
  - layout/
  - pages/
  - themes/
  - utils/
  - vendors/
  - main.scss

설계 패턴

- BASE 폴더
  - 리셋 파일
  - 타이포그래피 규칙
  - 자주 사용되는 HTML 요소에 대한 표준 스타일을 정의하는 스타일시트 예: base.scss
- COMPONENTS 폴더
  - 슬라이더
  - 로더
  - 위젯
  - 기본적으로 이들과 비슷한 구체적인 모듈들

```
sass/
|
|- base/
|   |- _reset.scss       # Reset/normalize
|   |- _typography.scss  # 타이포그래피 규칙
|   …                    # 기타.
|
|- components/
|   |- _buttons.scss     # 버튼
|   |- _carousel.scss    # 캐러셀
|   |- _cover.scss       # 커버
|   |- _dropdown.scss    # 드롭다운
|   …                    # 기타.
|
```

설계 패턴

- LAYOUT 폴더
  - 사이트의 주요 부분 (header, footer, navigation, sidebar...)
  - 그리드 시스템
  - 모든 폼의 스타일
- PAGES 폴더
  - 페이지에 한정된 스타일
- THEMES 폴더
  - 테마 스타일 관리

```
|- layout/
|   |- _navigation.scss  # 네비게이션
|   |- _grid.scss        # 그리드 시스템
|   |- _header.scss      # 헤더
|   |- _footer.scss      # 푸터
|   |- _sidebar.scss     # 사이드바
|   |- _forms.scss       # 폼
|   …                    # 기타.
|
|- pages/
|   |- _home.scss        # 홈 한정 스타일
|   |- _contact.scss     # 연락처 한정 스타일
|   …                    # 기타.
|
|- themes/
|   |- _theme.scss       # 디폴트 테마
|   |- _admin.scss       # 관리자 테마
|   …                    # 기타.
|
```

설계 패턴

- UTILS 폴더
  - 모든 Sass 도구와 헬퍼 모음
    - 전역 변수
    - 함수
    - 믹스인
    - 플레이스홀더
  - 컴파일되었을 때 한 줄의 CSS도 산출하지 않아야 한다.
- VENDORS 폴더
  - 외부 라이브러리와 프레임워크에서 나오는 모든 CSS 파일
- MAIN 파일
  - 언더스코어로 시작하지 않는 유일한 Sass 파일
  - **@import와 주석 외에는 아무 것도 포함하지 않아야 한다.**

```
├─ utils/
│   ├─ _variables.scss   # Sass 변수
│   ├─ _functions.scss   # Sass 함수
│   ├─ _mixins.scss      # Sass 믹스인
│   ├─ _helpers.scss     # 클래스 & 플레이스홀더 헬퍼
│
├─ vendors/
│   ├─ _bootstrap.scss   # Bootstrap
│   ├─ _jquery-ui.scss   # jQuery UI
│   …                    # 기타.
│
│
└─ main.scss             # 메인 Sass 파일
```

설계 패턴 – main.scss

- @import 당 파일 하나
- 한 줄에 하나의 @import
- 같은 폴더로부터의 두 import 사이는 새 줄로 띄우지 않는다.
- 한 폴더로부터의 마지막 import 다음에는 새 줄 하나로 간격을 둔다.
- 파일 확장자와 앞에 붙는 언더스코어는 생략한다.

```
@import 'base/reset';
@import 'base/typography';

@import 'layout/navigation';
@import 'layout/grid';
@import 'layout/header';
@import 'layout/footer';
@import 'layout/sidebar';
@import 'layout/forms';

@import 'components/buttons';
@import 'components/carousel';
@import 'components/cover';
@import 'components/dropdown';

@import 'pages/home';
@import 'pages/contact';

@import 'themes/theme';
```

반응형 웹 디자인과 브레이크포인트

```
// Yep
$breakpoints: (
  'medium': (min-width: 800px),
  'large': (min-width: 1000px),
  'huge': (min-width: 1200px),
);

// Nope
$breakpoints: (
  'tablet': (min-width: 800px),
  'computer': (min-width: 1000px),
  'tv': (min-width: 1200px),
);
```

```
$breakpoints: (
  'seed': (min-width: 800px),
  'sprout': (min-width: 1000px),
  'plant': (min-width: 1200px),
);
```

참고

- Naming Media Queries

브레이크 포인트 매니저

```
/// 반응형 매니저
/// @access public
/// @param {String} $breakpoint - 브레이크포인트
/// @requires $breakpoints
@mixin respond-to($breakpoint) {
  $raw-query: map-get($breakpoints, $breakpoint);

  @if $raw-query {
    $query: if(
      type-of($raw-query) == 'string',
      unquote($raw-query),
      inspect($raw-query)
    );

    @media #{$query} {
      @content;
    }
  } @else {
    @error 'No value found for `#{$breakpoint}`. '
         + 'Please make sure it is defined in `$breakpoints` map.';
  }
}
```

참고

- Managing Responsive Breakpoints in Sass
- Approaches to Media Queries in Sass

미디어 쿼리 사용법

```
.foo {
  color: red;

  @include respond-to('medium') {
    color: blue;
  }
}
```

```
.foo {
  color: red;
}

@media (min-width: 800px) {
  .foo {
    color: blue;
  }
}
```

참고

- Sass and Media Queries
- Inline or Combined Media queries? Fight!
- Sass::MediaQueryCombiner

변수의 스코프

- 전역 스코프에 이미 존재하는 변수를 내부 스코프 (선택자, 함수, 믹스인...)에서 선언할 때, '지역 변수가 전역 변수를 가린다' 고 한다.
- 기본적으로, 지역 변수가 지역 스코프 내에서는 우선시 된다.

```
// 루트 수준에 전역 변수를 초기화합니다.
$variable: 'initial value';

// 전역 변수를 덮어쓰게 하는 믹스인을 만듭니다.
@mixin global-variable-overriding {
  $variable: 'mixin value' !global;
}

.local-scope::before {
  // 전역 변수를 가리는 지역 변수를 만듭니다.
  $variable: 'local value';

  // 믹스인 인클루드: 전역 변수를 덮어씁니다.
  @include global-variable-overriding;

  // 변수의 값을 출력합니다.
  // 전역 변수를 가리기 때문에, **지역** 변수의 값이 출력됩니다.
  content: $variable;
}

// 변수 가림이 없는 다른 선택자에서 변수를 출력합니다.
// 예상대로, **전역** 변수의 값이 출력됩니다.
.other-local-scope::before {
  content: $variable;
}
```

!default 플래그

- 라이브러리나 프레임워크, 그리드 시스템, 혹은 배포되어 외부의 개발자들에 의해 사용될 Sass 소품을 개발할 때는, 덮어 쓰일 수 있도록 모든 환경설정 변수들을 !default 플래그를 붙여 정의한다.

```
$baseline: 1em !default;
```

!global 플래그

- 지역 스코프로부터 전역 변수를 정의할 때에만 사용되어야 한다.
- 루트 레벨에서 변수를 정의할 때, !global 플래그는 생략되어야 한다.

```
// Yep
$baseline: 2em;

// Nope
$baseline: 2em !global;
```

여러 개의 변수 혹은 맵

- 맵 사용의 장점
  - 여러 개의 다른 변수들 대신 맵을 사용하는 것
  - 사용이 편리한 API를 제공하는 작은 getter 함수를 만드는 기능

```
/// Z-index 맵. 어플리케이션의 z 레이어들을 한데 모음.
/// @access private
/// @type Map
/// @prop {String} 키 - 레이어 이름
/// @prop {Number} 값 - 키에 연결된 z 값
$z-indexes: (
  'modal': 5000,
  'dropdown': 4000,
  'default': 1,
  'below': -1,
);

/// 레이어 이름으로부터 z-index 값을 가져온다.
/// @access public
/// @param {String} $layer - 레이어 이름
/// @return {Number}
/// @require $z-indexes
@function z($layer) {
  @return map-get($z-indexes, $layer);
}
```

Mixins 기본

- Nicolas Gallagher의 마이크로 클리어픽스 핵

```
/// 내부 float을 해제하는 헬퍼
/// @author Nicolas Gallagher
/// @link http://nicolasgallagher.com/micro-clearfix-hack/ Micro Clearfix
@mixin clearfix {
  &::after {
    content: '';
    display: table;
    clear: both;
  }
}
```

Mixins 기본

- 요소의 크기를 조절하는 믹스인
  - width와 height를 동시에 정의

```
/// 요소 크기를 설정하는 헬퍼
/// @author Hugo Giraudel
/// @param {Length} $width
/// @param {Length} $height
@mixin size($width, $height: $width) {
  width: $width;
  height: $height;
}
```

참고

- Sass Mixins to Kickstart your Project
- A Sass Mixin for CSS Triangles
- Building a Linear-Gradient Mixin

Mixins 매개변수 리스트

- 믹스인에 들어가는 매개변수의 개수를 알 수 없을 때는, arglist를 사용한다.
- Arglist는 임의의 수의 매개변수를 믹스인이나 함수에 전달할 때 암묵적으로 사용되는 Sass의 여덟 번째 데이터 유형이라고 생각할 수 있으며, ...이 그 특징이다.

```
@mixin shadows($shadows...) {
  // type-of($shadows) == 'arglist'
  // …
}
```

```
@mixin dummy($a, $b, $c) {
  // …
}

// Yep
@include dummy(true, 42, 'kittens');

// Yep but nope
$params: (true, 42, 'kittens');
$value: dummy(nth($params, 1), nth($params, 2), nth($params, 3));

// Yep
$params: (true, 42, 'kittens');
@include dummy($params...);

// Yep
$params: (
  'c': 'kittens',
  'a': true,
  'b': 42,
);
@include dummy($params...);
```

참고

- Sass Multiple Arguments, Lists or Arglist

조건문

- 필요한 경우가 아니라면 괄호 없이
- @if 앞에는 빈 새 줄 하나
- 여는 중괄호({) 뒤에는 줄 바꿈
- @else 문은 이전의 닫는 중괄호(})와 같은 줄에
- 다음 줄이 닫는 중괄호(})가 아닌 한 마지막 닫는 중괄호(}) 뒤에는 빈 새 줄 하나

```
// Yep
@if $support-legacy {
  // …
} @else {
  // …
}

// Nope
@if ($support-legacy == true) {
  // …
}
@else {
  // …
}
```

조건문

- 거짓 값을 테스트할 때는, false나 null 대신 언제나 not 키워드를 사용한다.

```
// Yep
@if not index($list, $item) {
  // …
}

// Nope
@if index($list, $item) == null {
  // …
}
```

조건문

- 변수 부분은 조건문의 왼쪽에, 결과는 오른쪽에 표기한다.

```
// Yep
@if $value == 42 {
  // …
}

// Nope
@if 42 == $value {
  // …
}
```

조건문

- 조건에 따라 다른 결과를 반환하는 함수 안에서 조건문을 사용할 때는, 반드시 함수가 조건문 블록 밖에서도 @return문을 갖도록 만든다.

```
// Yep
@function dummy($condition) {
  @if $condition {
    @return true;
  }

  @return false;
}

// Nope
@function dummy($condition) {
  @if $condition {
    @return true;
  } @else {
    @return false;
  }
}
```

반복문 - Each

- @each 반복문은 Sass가 제공하는 반복문들 중에서 가장 많이 사용한다.

```
@each $theme in $themes {
  .section-#{$theme} {
    background-color: map-get($colors, $theme);
  }
}
```

- 맵에서 반복할 때, 일관성을 강제하기 위해 언제나 $key와 $value를 변수 이름으로 사용한다.

```
@each $key, $value in $map {
  .section-#{$key} {
    background-color: $value;
  }
}
```

반복문 - For

- @for 반복문은 CSS의 :nth-* 가상 클래스와 결합되었을 때 유용할 수 있다.

```
@for $i from 1 through 10 {
  .foo:nth-of-type(#{$i}) {
    border-color: hsl($i * 36, 50%, 50%);
  }
}
```

- 일반적인 관례를 따라 항상 $i를 변수 이름으로 사용한다.
- to 키워드 대신 through를 사용한다.

그리드 시스템

- Susy : 활성화된 커뮤니티
- csswizardry-grids : 경량 그리드 시스템
- Singularity

참고

- Singularity
- Singularity: Grids Without Limits
- Singularity Grid System
- Susy
- Build Web Layouts Easily with Susy
- A Complete Tutorial to Susy 2
- Sass Grids: From Neat to Susy
- Bootstrap's Grid System vs Susy: a Comparison
- How to Use Susy: Superpowered Sass Grids
- A Creative Grid System with Sass and calc()

요약

- 탭 대신 스페이스 두(2) 칸을 들여쓴다.
- 행의 너비는 80 글자.
- CSS를 여러 행으로 적절히 작성한다.
- 공백을 의미 있게 사용한다.
- 문자열과 URL에는 인용 부호(작은 따옴표)를 붙인다.
- 뒤따르는 0은 표기하지 않는다. 앞장서는 0은 반드시 표기한다.
- 연산은 괄호로 감싼다.
- 매직 넘버를 피한다.
- 색은 키워드 > HSL > RGB > 16진법 순으로 표기한다.
- 리스트는 쉼표로 구분한다.
- 리스트에는 뒤따르는 쉼표를 붙이지 않는다.
- 맵에는 뒤따르는 쉼표를 붙인다.
- 가상 클래스와 가상 요소 외에는 선택자를 내포하지 않는다.
- 작명 시 하이픈으로 구분한다.
- 많은 주석을 붙인다.
- SassDoc을 이용하여 API 주석을 표기한다.
- @extend는 제한적으로 사용한다.
- 간단한 믹스인을 사용한다.
- 반복문은 최소한으로 사용하고, @while은 사용하지 않는다.
- 의존성의 수를 줄인다.
- 경고와 오류를 의미 있게 사용한다.