**인수의 기본값 설정**

인수(argument)는 기본값(default value)을 가질 수 있습니다. **@include** 포함 단계에서 별도의 인수가 전달되지 않으면 기본값이 사용됩니다.

**@mixin 믹스인이름($매개변수: 기본값) {**

**스타일;**

**}**

SCSS:

**@mixin dash-line($width: 1px, $color: black) {**

**border: $width dashed $color;**

**}**

**.box1 { @include dash-line; }**

**.box2 { @include dash-line(4px); }**

Compiled to:

**.box1 {**

**border: 1px dashed black;**

**}**

**.box2 {**

**border: 4px dashed black;**

**}**

**키워드 인수(Keyword Arguments)**

**@mixin 믹스인이름($매개변수A: 기본값, $매개변수B: 기본값) {**

**스타일;**

**}**

**@include 믹스인이름($매개변수B: 인수);**

Mixin에 전달할 인수를 입력할 때 명시적으로 키워드(변수)를 입력하여 작성할 수 있습니다.  
별도의 인수 입력 순서를 필요로 하지 않아 편리하게 작성할 수 있습니다.  
단, 작성하지 않은 인수가 적용될 수 있도록 기본값을 설정해 주는 것이 좋습니다.

**@mixin position(**

**$p: absolute,**

**$t: null,**

**$b: null,**

**$l: null,**

**$r: null**

**) {**

**position: $p;**

**top: $t;**

**bottom: $b;**

**left: $l;**

**right: $r;**

**}**

**.absolute {**

**// 키워드 인수로 설정할 값만 전달**

**@include position($b: 10px, $r: 20px);**

**}**

**.fixed {**

**// 인수가 많아짐에 따라 가독성을 확보하기 위해 줄바꿈**

**@include position(**

**fixed,**

**$t: 30px,**

**$r: 40px**

**);**

**}**

**.absolute {**

**position: absolute;**

**bottom: 10px;**

**right: 20px;**

**}**

**.fixed {**

**position: fixed;**

**top: 30px;**

**right: 40px;**

**}**

**가변 인수(Variable Arguments)**

때때로 입력할 인수의 개수가 불확실한 경우가 있습니다.  
그럴 경우 가변 인수를 사용할 수 있습니다.  
가변 인수는 매개변수 뒤에 **...**을 붙여줍니다.

**@mixin 믹스인이름($매개변수...) {**

**스타일;**

**}**

**@include 믹스인이름(인수A, 인수B, 인수C);**

**// 인수를 순서대로 하나씩 전달 받다가, 3번째 매개변수($bg-values)는 인수의 개수에 상관없이 받음**

**@mixin bg($width, $height, $bg-values...) {**

**width: $width;**

**height: $height;**

**background: $bg-values;**

**}**

**div {**

**// 위의 Mixin(bg) 설정에 맞게 인수를 순서대로 전달하다가 3번째 이후부터는 개수에 상관없이 전달**

**@include bg(**

**100px,**

**200px,**

**url("/images/a.png") no-repeat 10px 20px,**

**url("/images/b.png") no-repeat,**

**url("/images/c.png")**

**);**

**}**

**div {**

**width: 100px;**

**height: 200px;**

**background: url("/images/a.png") no-repeat 10px 20px,**

**url("/images/b.png") no-repeat,**

**url("/images/c.png");**

**}**

위에선 인수를 받는 매개변수에 **...**을 사용하여 가변 인수를 활용했습니다.  
이번엔 반대로 가변 인수를 전달할 값으로 사용해 보겠습니다.

**@mixin font(**

**$style: normal,**

**$weight: normal,**

**$size: 16px,**

**$family: sans-serif**

**) {**

**font: {**

**style: $style;**

**weight: $weight;**

**size: $size;**

**family: $family;**

**}**

**}**

**div {**

**// 매개변수 순서와 개수에 맞게 전달**

**$font-values: italic, bold, 16px, sans-serif;**

**@include font($font-values...);**

**}**

**span {**

**// 필요한 값만 키워드 인수로 변수에 담아 전달**

**$font-values: (style: italic, size: 22px);**

**@include font($font-values...);**

**}**

**a {**

**// 필요한 값만 키워드 인수로 전달**

**@include font((weight: 900, family: monospace)...);**

**}**

**div {**

**font-style: italic;**

**font-weight: bold;**

**font-size: 16px;**

**font-family: sans-serif;**

**}**

**span {**

**font-style: italic;**

**font-weight: normal;**

**font-size: 22px;**

**font-family: sans-serif;**

**}**

**a {**

**font-style: normal;**

**font-weight: 900;**

**font-size: 16px;**

**font-family: monospace;**

**}**

**@content**

선언된 Mixin에 **@content**이 포함되어 있다면 해당 부분에 원하는 **스타일 블록** 을 전달할 수 있습니다.  
이 방식을 사용하여 기존 Mixin이 가지고 있는 기능에 선택자나 속성 등을 추가할 수 있습니다.

**@mixin 믹스인이름() {**

**스타일;**

**@content;**

**}**

**@include 믹스인이름() {**

**// 스타일 블록**

**스타일;**

**}**

SCSS:

**@mixin icon($url) {**

**&::after {**

**content: $url;**

**@content;**

**}**

**}**

**.icon1 {**

**// icon Mixin의 기존 기능만 사용**

**@include icon("/images/icon.png");**

**}**

**.icon2 {**

**// icon Mixin에 스타일 블록을 추가하여 사용**

**@include icon("/images/icon.png") {**

**position: absolute;**

**};**

**}**

Compiled to:

**.icon1::after {**

**content: "/images/icon.png";**

**}**

**.icon2::after {**

**content: "/images/icon.png";**

**position: absolute;**

**}**

Mixin에게 전달된 스타일 블록은 Mixin의 범위가 아니라 스타일 블록이 정의된 범위에서 평가됩니다.  
즉, Mixin의 매개변수는 전달된 스타일 블록 안에서 사용되지 않고 전역 값으로 해석됩니다.  
전역 변수(Global variables)와 지역 변수(Local variables)를 생각하면 좀 더 쉽습니다.

SCSS:

**$color: red;**

**@mixin colors($color: blue) {**

**// Mixin의 범위**

**@content;**

**background-color: $color;**

**border-color: $color;**

**}**

**div {**

**@include colors() {**

**// 스타일 블록이 정의된 범위**

**color: $color;**

**}**

**}**

Compiled to:

**div {**

**color: red;**

**background-color: blue;**

**border-color: blue;**

**}**

**확장(Extend)**

특정 선택자가 다른 선택자의 모든 스타일을 가져야하는 경우가 종종 있습니다.  
이럴 경우 선택자의 확장 기능을 사용할 수 있습니다.  
다음 예제를 봅시다.

**@extend 선택자;**

SCSS:

**.btn {**

**padding: 10px;**

**margin: 10px;**

**background: blue;**

**}**

**.btn-danger {**

**@extend .btn;**

**background: red;**

**}**

Compiled to:

**.btn, .btn-danger {**

**padding: 10px;**

**margin: 10px;**

**background: blue;**

**}**

**.btn-danger {**

**background: red;**

**}**

컴파일된 결과가 마음에 드시나요?  
결과를 보면 **,**로 구분하는 다중 선택자(Multiple Selector)가 만들어졌습니다.

사실 **@extend**는 다음과 같은 문제를 고려해야 합니다.

* 내 현재 선택자(위 예제의 **.btn-danger**)가 어디에 첨부될 것인가?
* 원치 않는 부작용이 초래될 수도 있는가?
* 이 한 번의 확장으로 얼마나 큰 CSS가 생성되는가?

결과적으로 확장(Extend) 기능은 무해하거나 혹은 유익할 수도 있지만 그만큼 **부작용**을 가지고 있을 수 있습니다.  
따라서 확장은 사용을 권장하지 않으며, 위에서 살펴본 **Mixin을 대체 기능으로 사용**하세요.

사용을 권장하지 않는 이유에 대해서 좀 더 자세한 정보를 원하면 [Sass Guidelines Extend](https://sass-guidelin.es/ko/#extend)를 참고하세요.

**함수(Functions)**

자신의 함수를 정의하여 사용할 수 있습니다.  
함수와 Mixins은 거의 유사하지만 반환되는 내용이 다릅니다.

Mixin은 위에서 살펴본 대로 지정한 스타일(Style)을 반환하는 반면,  
함수는 보통 연산된(Computed) 특정 **값**을 **@return** 지시어를 통해 반환합니다.

**// Mixins**

**@mixin 믹스인이름($매개변수) {**

**스타일;**

**}**

**// Functions**

**@function 함수이름($매개변수) {**

**@return 값**

**}**

사용하는 방법에도 차이가 있습니다.  
Mixin은 **@include** 지시어를 사용하는 반면,  
함수는 함수이름으로 바로 사용합니다.

**// Mixin**

**@include 믹스인이름(인수);**

**// Functions**

**함수이름(인수)**

SCSS:

**$max-width: 980px;**

**@function columns($number: 1, $columns: 12) {**

**@return $max-width \* ($number / $columns)**

**}**

**.box\_group {**

**width: $max-width;**

**.box1 {**

**width: columns(); // 1**

**}**

**.box2 {**

**width: columns(8);**

**}**

**.box3 {**

**width: columns(3);**

**}**

**}**

Compiled to:

**.box\_group {**

**/\* 총 너비 \*/**

**width: 980px;**

**}**

**.box\_group .box1 {**

**/\* 총 너비의 약 8.3% \*/**

**width: 81.66667px;**

**}**

**.box\_group .box2 {**

**/\* 총 너비의 약 66.7% \*/**

**width: 653.33333px;**

**}**

**.box\_group .box3 {**

**/\* 총 너비의 25% \*/**

**width: 245px;**

**}**

위와 같이 함수는 **@include** 같은 별도의 지시어 없이 사용하기 때문에 내가 지정한 함수와 내장 함수(Built-in Functions)의 이름이 충돌할 수 있습니다.  
따라서 내가 지정한 함수에는 별도의 접두어를 붙여주는 것이 좋습니다.

내장 함수란, 응용 프로그램에 내장되어 있으며 최종 사용자가 액세스 할 수 있는 기능입니다.  
예를 들어, 대부분의 스프레드 시트 응용 프로그램은 행이나 열의 모든 셀을 추가하는 내장 SUM 함수를 지원합니다.

예를 들어, 색의 빨강 성분을 가져오는 내장 함수로 이미 **red()**가 있습니다.  
같은 이름을 사용하여 함수를 정의하면 이름이 충돌하기 때문에 별도의 접두어를 붙여 **extract-red()** 같은 이름을 만들 수 있습니다.

**// 내가 정의한 함수**

**@function extract-red($color) {**

**// 내장 함수**

**@return rgb(red($color), 0, 0);**

**}**

**div {**

**color: extract-red(#D55A93);**

**}**

혹은 모든 내장 함수의 이름을 다 알고 있을 수 없기 때문에 특별한 이름을 접두어로 사용할 수도 있습니다.  
**my-custom-func-red()**

**조건과 반복(Control Directives / Expressions)**

**if (함수)**

조건의 값(**true**, **false**)에 따라 두 개의 표현식 중 하나만 반환합니다.  
[조건부 삼항 연산자(conditional ternary operator)](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Operators/Conditional_Operator)와 비슷합니다.

조건의 값이 **true**이면 **표현식1**을,  
조건의 값이 **false**이면 **표현식2**를 실행합니다.

**if(조건, 표현식1, 표현식2)**

SCSS:

**$width: 555px;**

**div {**

**width: if($width > 300px, $width, null);**

**}**

Compiled to:

**div {**

**width: 555px;**

**}**

**@if (지시어)**

**@if** 지시어는 조건에 따른 분기 처리가 가능하며, [if 문(if statements)](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/if...else)과 유사합니다.  
같이 사용할 수 있는 지시어는 **@else**, **if**가 있습니다.  
추가 지시어를 사용하면 좀 더 복잡한 조건문을 작성할 수 있습니다.

**// @if**

**@if (조건) {**

**/\* 조건이 참일 때 구문 \*/**

**}**

**// @if @else**

**@if (조건) {**

**/\* 조건이 참일 때 구문 \*/**

**} @else {**

**/\* 조건이 거짓일 때 구문 \*/**

**}**

**// @if @else if**

**@if (조건1) {**

**/\* 조건1이 참일 때 구문 \*/**

**} @else if (조건2) {**

**/\* 조건2가 참일 때 구문 \*/**

**} @else {**

**/\* 모두 거짓일 때 구문 \*/**

**}**

조건에 **()**는 생략이 가능하기 때문에, **()** 없이 작성하는 방법이 좀 더 편리할 수 있습니다.

**$bg: true;**

**div {**

**@if $bg {**

**background: url("/images/a.jpg");**

**}**

**}**

SCSS:

**$color: orange;**

**div {**

**@if $color == strawberry {**

**color: #FE2E2E;**

**} @else if $color == orange {**

**color: #FE9A2E;**

**} @else if $color == banana {**

**color: #FFFF00;**

**} @else {**

**color: #2A1B0A;**

**}**

**}**

Compiled to:

**div {**

**color: #FE9A2E;**

**}**

조건에는 논리 연산자 **and**, **or**, **not**을 사용할 수 있습니다.

SCSS:

**@function limitSize($size) {**

**@if $size >= 0 and $size <= 200px {**

**@return 200px;**

**} @else {**

**@return 800px;**

**}**

**}**

**div {**

**width: limitSize(180px);**

**height: limitSize(340px);**

**}**

Compiled to:

**div {**

**width: 200px;**

**height: 800px;**

**}**

좀 더 복잡하지만 실용적인 예제를 봅시다.  
Sass의 내장 함수 **unitless()**는 숫자에 단위가 있는지 여부를 반환합니다.

SCSS:

**@mixin pCenter($w, $h, $p: absolute) {**

**@if**

**$p == absolute**

**or $p == fixed**

**or not $p == relative**

**or not $p == static**

**{**

**width: if(unitless($w), #{$w}px, $w);**

**height: if(unitless($h), #{$h}px, $h);**

**position: $p;**

**top: 0;**

**bottom: 0;**

**left: 0;**

**right: 0;**

**margin: auto;**

**}**

**}**

**.box1 {**

**@include pCenter(10px, 20px);**

**}**

**.box2 {**

**@include pCenter(50, 50, fixed);**

**}**

**.box3 {**

**@include pCenter(100, 200, relative);**

**}**

Compiled to:

**.box1 {**

**width: 10px;**

**height: 20px;**

**position: absolute;**

**top: 0;**

**bottom: 0;**

**left: 0;**

**right: 0;**

**margin: auto;**

**}**

**.box2 {**

**width: 50px;**

**height: 50px;**

**position: fixed;**

**top: 0;**

**bottom: 0;**

**left: 0;**

**right: 0;**

**margin: auto;**

**}**

**@for**

**@for**는 스타일을 반복적으로 출력합니다.  
[for 문](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/for)과 유사합니다.

**@for**는 **through**를 사용하는 형식과 **to**를 사용하는 형식으로 나뉩니다.  
두 형식은 종료 조건이 해석되는 방식이 다릅니다.

**// through**

**// 종료 만큼 반복**

**@for $변수 from 시작 through 종료 {**

**// 반복 내용**

**}**

**// to**

**// 종료 직전까지 반복**

**@for $변수 from 시작 to 종료 {**

**// 반복 내용**

**}**

차이점을 이해하기 위해 다음 예제를 살펴봅시다.  
변수는 관례상 **$i**를 사용합니다.

SCSS:

**// 1부터 3번 반복**

**@for $i from 1 through 3 {**

**.through:nth-child(#{$i}) {**

**width : 20px \* $i**

**}**

**}**

**// 1부터 3 직전까지만 반복(2번 반복)**

**@for $i from 1 to 3 {**

**.to:nth-child(#{$i}) {**

**width : 20px \* $i**

**}**

**}**

Compiled to:

**.through:nth-child(1) { width: 20px; }**

**.through:nth-child(2) { width: 40px; }**

**.through:nth-child(3) { width: 60px; }**

**.to:nth-child(1) { width: 20px; }**

**.to:nth-child(2) { width: 40px; }**

**to**는 주어진 값 직전까지만 반복해야할 경우 유용할 수 있습니다.  
그러나 **:nth-child()**에서 특히 유용하게 사용되는 **@for**는 일반적으로 **through**를 사용하길 권장합니다.

**@each**

**@each**는 List와 Map 데이터를 반복할 때 사용합니다.  
[for in 문](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/for...in)과 유사합니다.

**@each $변수 in 데이터 {**

**// 반복 내용**

**}**

List 데이터를 반복해 보겠습니다.

SCSS:

**// List Data**

**$fruits: (apple, orange, banana, mango);**

**.fruits {**

**@each $fruit in $fruits {**

**li.#{$fruit} {**

**background: url("/images/#{$fruit}.png");**

**}**

**}**

**}**

Compiled to:

**.fruits li.apple {**

**background: url("/images/apple.png");**

**}**

**.fruits li.orange {**

**background: url("/images/orange.png");**

**}**

**.fruits li.banana {**

**background: url("/images/banana.png");**

**}**

**.fruits li.mango {**

**background: url("/images/mango.png");**

**}**

혹시 매번 반복마다 Index 값이 필요하다면 다음과 같이 **index()** 내장 함수를 사용할 수 있습니다.

SCSS:

**$fruits: (apple, orange, banana, mango);**

**.fruits {**

**@each $fruit in $fruits {**

**$i: index($fruits, $fruit);**

**li:nth-child(#{$i}) {**

**left: 50px \* $i;**

**}**

**}**

**}**

Compiled to:

**.fruits li:nth-child(1) {**

**left: 50px;**

**}**

**.fruits li:nth-child(2) {**

**left: 100px;**

**}**

**.fruits li:nth-child(3) {**

**left: 150px;**

**}**

**.fruits li:nth-child(4) {**

**left: 200px;**

**}**

동시에 여러 개의 List 데이터를 반복 처리할 수도 있습니다.  
단, 각 데이터의 Length가 같아야 합니다.

SCSS:

**$apple: (apple, korea);**

**$orange: (orange, china);**

**$banana: (banana, japan);**

**@each $fruit, $country in $apple, $orange, $banana {**

**.box-#{$fruit} {**

**background: url("/images/#{$country}.png");**

**}**

**}**

Compiled to:

**.box-apple {**

**background: url("/images/korea.png");**

**}**

**.box-orange {**

**background: url("/images/china.png");**

**}**

**.box-banana {**

**background: url("/images/japan.png");**

**}**

Map 데이터를 반복할 경우 하나의 데이터에 두 개의 변수가 필요합니다.

**@each $key변수, $value변수 in 데이터 {**

**// 반복 내용**

**}**

**$fruits-data: (**

**apple: korea,**

**orange: china,**

**banana: japan**

**);**

**@each $fruit, $country in $fruits-data {**

**.box-#{$fruit} {**

**background: url("/images/#{$country}.png");**

**}**

**}**

**.box-apple {**

**background: url("/images/korea.png");**

**}**

**.box-orange {**

**background: url("/images/china.png");**

**}**

**.box-banana {**

**background: url("/images/japan.png");**

**}**

**@while**

**@while**은 조건이 **false**로 평가될 때까지 내용을 반복합니다.  
[while 문](https://developer.mozilla.org/ko/docs/Web/JavaScript/Reference/Statements/while)과 유사하게 잘못된 조건으로 인해 컴파일 중 무한 루프에 빠질 수 있습니다.  
사용을 권장하지 않습니다.

**@while 조건 {**

**// 반복 내용**

**}**

**$i: 6;**

**@while $i > 0 {**

**.item-#{$i} {**

**width: 2px \* $i;**

**}**

**$i: $i - 2;**

**}**

**.item-6 { width: 12px; }**

**.item-4 { width: 8px; }**

**.item-2 { width: 4px; }**

**내장 함수(Built-in Functions)**

Sass에서 기본적으로 제공하는 내장 함수에는 많은 종류가 있습니다.  
모두 소개하지 않고, 주관적 경험에 의거해 필요하거나 필요할 수 있는 함수만 정리했습니다.

[Sass Built-in Functions](http://sass-lang.com/documentation/Sass/Script/Functions.html)에서 모든 내장 함수를 확인할 수 있습니다.

* **[]**는 선택 가능한 인수(argument)입니다.
* [Zero-based numbering](https://en.wikipedia.org/wiki/Zero-based_numbering)을 사용하지 않습니다.

**색상(RGB / HSL / Opacity) 함수**

[mix($color1, $color2)](http://sass-lang.com/documentation/Sass/Script/Functions.html#mix-instance_method) : 두 개의 색을 섞습니다.

[lighten($color, $amount)](http://sass-lang.com/documentation/Sass/Script/Functions.html#lighten-instance_method) : 더 밝은색을 만듭니다.

[darken($color, $amount)](http://sass-lang.com/documentation/Sass/Script/Functions.html#darken-instance_method) : 더 어두운색을 만듭니다.

[saturate($color, $amount)](http://sass-lang.com/documentation/Sass/Script/Functions.html#saturate-instance_method) : 색상의 채도를 올립니다.

[desaturate($color, $amount)](http://sass-lang.com/documentation/Sass/Script/Functions.html#desaturate-instance_method) : 색상의 채도를 낮춥니다.

[grayscale($color)](http://sass-lang.com/documentation/Sass/Script/Functions.html#grayscale-instance_method) : 색상을 회색으로 변환합니다.

[invert($color)](http://sass-lang.com/documentation/Sass/Script/Functions.html#invert-instance_method) : 색상을 반전시킵니다.

[rgba($color, $alpha)](http://sass-lang.com/documentation/Sass/Script/Functions.html#rgba-instance_method) : 색상의 투명도를 변경합니다.

[opacify($color, $amount) / fade-in($color, $amount)](http://sass-lang.com/documentation/Sass/Script/Functions.html#opacify-instance_method) : 색상을 더 불투명하게 만듭니다.

[transparentize($color, $amount) / fade-out($color, $amount)](http://sass-lang.com/documentation/Sass/Script/Functions.html#transparentize-instance_method) : 색상을 더 투명하게 만듭니다.

**문자(String) 함수**

[unquote($string)](http://sass-lang.com/documentation/Sass/Script/Functions.html#unquote-instance_method) : 문자에서 따옴표를 제거합니다.

[quote($string)](http://sass-lang.com/documentation/Sass/Script/Functions.html#quote-instance_method) : 문자에 따옴표를 추가합니다.

[str-insert($string, $insert, $index)](http://sass-lang.com/documentation/Sass/Script/Functions.html#str_insert-instance_method) : 문자의 index번째에 특정 문자를 삽입합니다.

[str-index($string, $substring)](http://sass-lang.com/documentation/Sass/Script/Functions.html#str_index-instance_method) : 문자에서 특정 문자의 첫 index를 반환합니다.

[str-slice($string, $start-at, [$end-at])](http://sass-lang.com/documentation/Sass/Script/Functions.html#str_slice-instance_method) : 문자에서 특정 문자(몇 번째 글자부터 몇 번째 글자까지)를 추출합니다.

[to-upper-case($string)](http://sass-lang.com/documentation/Sass/Script/Functions.html#to_upper_case-instance_method) : 문자를 대문자를 변환합니다.

[to-lower-case($string)](http://sass-lang.com/documentation/Sass/Script/Functions.html#to_lower_case-instance_method) : 문자를 소문자로 변환합니다.

**숫자(Number) 함수**

[percentage($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#percentage-instance_method) : 숫자(단위 무시)를 백분율로 변환합니다.

[round($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#round-instance_method) : 정수로 반올림합니다.

[ceil($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#ceil-instance_method) : 정수로 올림합니다.

[floor($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#floor-instance_method) : 정수로 내림(버림)합니다.

[abs($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#abs-instance_method) : 숫자의 절대 값을 반환합니다.

[min($numbers…)](http://sass-lang.com/documentation/Sass/Script/Functions.html#min-instance_method) : 숫자 중 최소 값을 찾습니다.

[max($numbers…)](http://sass-lang.com/documentation/Sass/Script/Functions.html#max-instance_method) : 숫자 중 최대 값을 찾습니다.

[random()](http://sass-lang.com/documentation/Sass/Script/Functions.html#random-instance_method) : **0** 부터 **1** 사이의 난수를 반환합니다.

**List 함수**

모든 List 내장 함수는 기존 List 데이터를 갱신하지 않고 새 List 데이터를 반환합니다.  
모든 List 내장 함수는 Map 데이터에서도 사용할 수 있습니다.

[length($list)](http://sass-lang.com/documentation/Sass/Script/Functions.html#length-instance_method) : List의 개수를 반환합니다.

[nth($list, $n)](http://sass-lang.com/documentation/Sass/Script/Functions.html#nth-instance_method) : List에서 n번째 값을 반환합니다.

[set-nth($list, $n, $value)](http://sass-lang.com/documentation/Sass/Script/Functions.html#set_nth-instance_method) : List에서 n번째 값을 다른 값으로 변경합니다.

[join($list1, $list2, [$separator])](http://sass-lang.com/documentation/Sass/Script/Functions.html#join-instance_method) : 두 개의 List를 하나로 결합합니다.

[zip($lists…)](http://sass-lang.com/documentation/Sass/Script/Functions.html#zip-instance_method) : 여러 List들을 하나의 다차원 List로 결합합니다.

[index($list, $value)](http://sass-lang.com/documentation/Sass/Script/Functions.html#index-instance_method) : List에서 특정 값의 index를 반환합니다.

**Map 함수**

모든 Map 내장 함수는 기존 Map 데이터를 갱신하지 않고 새 Map 데이터를 반환합니다.

[map-get($map, $key)](http://sass-lang.com/documentation/Sass/Script/Functions.html#map_get-instance_method) : Map에서 특정 key의 value를 반환합니다.

[map-merge($map1, $map2)](http://sass-lang.com/documentation/Sass/Script/Functions.html#map_merge-instance_method) : 두 개의 Map을 병합하여 새로운 Map를 만듭니다.

[map-keys($map)](http://sass-lang.com/documentation/Sass/Script/Functions.html#map_keys-instance_method) : Map에서 모든 key를 List로 반환합니다.

[map-values($map)](http://sass-lang.com/documentation/Sass/Script/Functions.html#map_values-instance_method) : Map에서 모든 value를 List로 반환합니다.

**관리(Introspection) 함수**

[variable-exists(name)](http://sass-lang.com/documentation/Sass/Script/Functions.html#variable_exists-instance_method) : 변수가 현재 범위에 존재하는지 여부를 반환합니다.(인수는 **$**없이 변수의 이름만 사용합니다.)

[unit($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#unit-instance_method) : 숫자의 단위를 반환합니다.

[unitless($number)](http://sass-lang.com/documentation/Sass/Script/Functions.html#unitless-instance_method) : 숫자에 단위가 있는지 여부를 반환합니다.

[comparable($number1, $number2)](http://sass-lang.com/documentation/Sass/Script/Functions.html#comparable-instance_method) : 두 개의 숫자가 연산 가능한지 여부를 반환합니다.

**참고 자료(References)**

<http://sass-lang.com/documentation>  
<https://www.sitepoint.com/sass-basics-operators/>  
<https://sass-guidelin.es/ko/>  
<http://www.thesassway.com/>