Документация по SASS на русском: <http://sass-scss.ru/documentation/index.html>

Настройка SASS в WebStorm лежит в моей инструкции к WebStorm

1. **Расширение CSS**

**ВЛОЖЕННЫЕ ПРАВИЛА**

Sass позволяет вкладывать правила CSS друг в друга. Вложенные правила применяются только для элементов, соответствующих внешним селекторам.

#main {

width: 97%;

p, div {

font-size: 2em;

a { font-weight: bold; }

}

pre { font-size: 3em; }

}

компилируется в:

#main { width: 97%; }

#main p, #main div { font-size: 2em; }

#main p a, #main div a {font-weight: bold; }

#main pre {font-size: 3em; }

**ССЫЛКА НА РОДИТЕЛЯ СЕЛЕКТОРА**

Иногда полезно использовать родительский селектор с другим предназначением, чем по-умолчанию. Например, вы можете пожелать специальные стили для случая, когда над выбранными элементами находится курсор или когда тело элемента имеет определенный класс. В этих случаях, с помощью символа **&** вы можете явно указать, где должен быть вставлен родительский селектор, например:

#main {

color: black;

a {

font-weight: bold;

&:hover { color: red; }

}

}

компилируется в:

#main {color: black; }

#main a {font-weight: bold; }

#main a:hover {color: red; }

Символ **&** должен быть указан в начале составного селектора, также он может содержать последующий суффикс, который будет добавлен к родительскому селектору. Например:

#main {

color: black;

&-sidebar { border: 1px solid; }

}

компилируется в:

#main { color: black; }

#main-sidebar { border: 1px solid; }

**ВЛОЖЕННЫЕ СВОЙСТВА**

Sass упрощает процесса написания пространства имен просто напишите пространство имен единожды, а внутри вы можете вписать любое из вторичных свойств. Например:

.funky {

font: {

family: fantasy;

size: 30em;

weight: bold;

}

}

компилируется в:

.funky {

font-family: fantasy;

font-size: 30em;

font-weight: bold;

}

**ШАБЛОННЫЕ СЕЛЕКТОРЫ**

Sass поддерживает специальный тип селекторов под названием шаблонные селекторы. Они похожи на селекторы классов и идентификаторов, только вместо **#** или **.** указывается **%.** Они предназначены для использования с директивой **@extend**

Сами по себе, без использования **@extend**, наборы правил, которые используют шаблонные селекторы не окажут влияния на CSS. Например:

%inline-type { display: inline; zoom: 1; }

%for-grids {

padding-left: 10px;

padding-right: 10px;

}

.sgrid-N {

@extend %for-grids;

}

компилируется в

.sgrid-N {

padding-left: 10px;

padding-right: 10px;

}

1. **Комментирование**

Sass поддерживает стандартный многострочный CSS комментарий, обозначаемый /\* \*/, а также однострочный комментарий, обзначаемый //. Многострочные комментарии сохраняются на выходе в файле CSS, где это возможно, в то время как однострочные комментарии удаляются. Например:

/\* Это многострочный комментарий. Так как этот тип комментирования поддерживается в CSS, то он попадет в скомпилированный CSS-файл. \*/

body { color: black; }

// Это однострочный комментарий. CSS не поддерживает данный вариант комментирования.

// Поэтому данного комментария не будет в CSS-файле.

a { color: green; }

компилируется в:

/\* Это многострочный комментарий. Так как этот тип комментирования поддерживается в CSS, то он попадет в скомпилированный CSS-файл. \*/

body { color: black; }

a { color: green; }

Если первым символом многострочного комментария является знак **!**, то комментарий всегда будет оставаться в результирующем CSS, даже в режиме сжатия. Это полезно, например, для добавления уведомления об авторских правах на ваш CSS.

Поскольку многострочные комментарии становятся частью результирующего CSS, в них разрешена интерполяция. Например:

$version: "1.1.3";

/\* Этот CSS построен с использованием SomeGrids. Версия: #{$version}. \*/

компилируется в:

/\* Этот CSS построен с использованием SomeGrids. Версия: 1.1.3. \*/

1. **SassScript**

По настройке способов ввода\вывода, помощи и прочей лабуды - в документацию

**ПЕРЕМЕННЫЕ**

Переменные начинаются со знака доллара **$** и задаются как свойства CSS. Переменные доступны только в пределах того уровня вложенности селекторов, на котором они определены. Если они определяются вне каких-либо вложенных селекторов, они доступны глобально. Если вы хотите, чтобы объявленная на каком-либо уровне вложенности переменная также была доступна глобально, вы можете определить её со специальной меткой **!global**

#main {

$width: 5em !global;

width: $width;

}

#sidebar { width: $width; }

компилируется в

#main { width: 5em; }

#sidebar { width: 5em; }

По историческим причинам, в именах переменных (и прочих Sass идентификаторах) дефисы и подчеркивания взаимозаменяемы. Например, если вы определяете переменную $main-width, вы можете получить к ней доступ, написав $main\_width, и наоборот.

**ИНТЕРПОЛЯЦИЯ**

Вы также можете использовать переменные SassScript в селекторах и в названиях свойств используя синтаксис **#{}** интерполяции:

$name: foo;

$attr: border;

p.#{$name} {

#{$attr}-color: blue;

}

компилируется в

p.foo { border-color: blue; }

Также можно использовать **#{}**, чтобы вставить в SassScript значение свойств. В большинстве случаев это не лучше, чем любой другой способ, но использование **#{}** означает, что любые операции в нем будут рассматриваться как запись CSS. Например:

p {

$font-size: 12px;

$line-height: 30px;

font: #{$font-size}/#{$line-height};

}

компилируется в

p { font: 12px/30px; }

**СТРОКИ**

Когда используется **#{}** интерполяция, кавычки игнорируются. Это упрощает использование, скажем, имен селекторов в миксинах. Например:

@mixin firefox-message($selector) {

body.firefox #{$selector}:before {

content: "Hi, Firefox users!";

}

}

@include firefox-message(".header");

компилируется в

body.firefox .header:before { content: "Hi, Firefox users!"; }

**СПИСКИ**

**вернутся и прописать функции по ссылкам**

**Ссылка** [**http://sass-scss.ru/documentation/sassscript/spiski.html**](http://sass-scss.ru/documentation/sassscript/spiski.html)

Списки в Sass представлены значениями CSS-деклараций вроде margin: 10px 15px 0 0 или font-face: Helvetica, Arial, sans-serif. Списки - это всего навсего серия из нескольких однотипных значений разделенных пробелами или запятыми. Фактически, индивидуальные значения также являются списками, просто это списки, которые состоят из одного элемента.

Сами по себе списки не делают ничего особенного, но список функций SassScript сделал их полезнее. Функия **nth** может получить доступ к элементам в списке, функция **join** может объединить несколько списков вместе, а функция **append** может добавлять элементы в список. Директива **@each** также может добавлять стили для каждого элемента в списке.

**МАПЫ (ИЛИ АССОЦИАТИВНЫЕ МАССИВЫ)**

Мапы, или ассоциативные массивы, представляют собой связь между ключами и значениями, где ключи используются для поиска значений. Они позволяют легко собирать значения в именованные группы и получать доступ к этим группам динамически. Они не имеют прямого аналога в CSS, хотя они синтаксически похожи на media-запросы:

$map: (key1: value1, key2: value2, key3: value3);

В отличие от списков, мапы всегда должны быть заключены в скобки и всегда должны быть разделены запятыми. И ключи и значения в мапах могут быть любыми объектами SassScript. Мап может иметь только одно сопоставление с заданным ключом (хотя это значение может быть списком). Значение ключа, конечно же, может быть сопоставлено многим ключам.

Как и со списками, манипуляции с мапами в основном происходят через функции SassScript. Функция map-get смотрит значение в мапе, функция map-merge добавляет значения в мапе. Директива @each может быть использована для добавления стилей для каждой пары ключ-значение в мапе. Порядок пар всегда остается неизменным с момента создания мапа.

$status-colors: (

primary: #000,

success: #27BA6C,

info: #03a9f4,

warning: #FF8833,

danger: #ff1a1a

);

.message {

@each $status, $color in $status-colors {

&--#{$status} {

background: $color;

}

}

}

компилируется в

.message--primary { background: #000; }

.message--success { background: #27ba6c; }

.message--info { background: #03a9f4; }

.message--warning { background: #f83; }

.message--danger { background: #ff1a1a; }

**ДЕЛЕНИЕ И ЗНАК /**

Существует три ситуации, в которых символ **/** будет интерпретироваться именно как деление. Они охватывают подавляющее большинство случаев, когда подразумевается именно деление:

* Если значение или любая его часть, хранится в переменной или возвращается функцией
* Если значения заключены в круглые скобки
* Если значение используется как часть другого арифметического выражения

Например:

p {

font: 10px/8px; // Явный CSS, деление отсуствует

$width: 1000px;

width: $width/2; // Используется переменная, операция деления

width: round(1.5)/2; // Используется функция, операция деления

height: (500px/2); // Обособление скобками, операция деления

margin-left: 5px + 8px/2px; // Используется +, операция деления

}

**компилируется в**

p {

font: 10px/8px;

width: 500px;

height: 250px;

margin-left: 9px; }

Если вы хотите использовать символ **/** в переменной, в значении обычного CSS **/,** вы можете заключить переменную в **#{}**. Например:

p {

$font-size: 12px;

$line-height: 30px;

font: #{$font-size}/#{$line-height};

}

компилируется в

p { font: 12px/30px; }

**ОПЕРАЦИИ СО СТРОКАМИ**

Операция конкатенации строк осуществляется при помощи символа **+**. Например:

p { cursor: e + -resize;}

компилируется в

p { cursor: e-resize; }

Обратите внимание, что, если мы к строке без кавычек добавляем строку в кавычках (строка без кавычек слева от знака **+**), то результатом будет строка без кавычек. И, наоборот, если мы к строке в кавычках прибавляем строку без кавычек, результатом будет строка в кавычках. Например:

p:before {

content: "Foo " + Bar;

font-family: sans- + "serif";

}

компилируется в

p:before {

content: "Foo Bar";

font-family: sans-serif; }

По-умолчанию, если два значения находятся рядом друг с другом, они объединяются через пробел:

p { margin: 3px + 4px auto; }

компилируется в

p { margin: 7px auto; }

Возможно использовать интерполяцию **#{}** для размещения динамических значений в строке:

p:before { content: "I ate #{5 + 10} pies!";}

компилируется в

p:before { content: "I ate 15 pies!"; }

Значение **null** обрабатывается как пустая строка в интерполяции:

$value: null;

p:before { content: "I ate #{$value} pies!"; }

компилируется в

p:before { content: "I ate pies!"; }

**ЛОГИЧЕСКИЕ ОПЕРАЦИИ**

SassScript поддерживает операторы И, ИЛИ и НЕ (**and, or, not**) для логических значений. Например:

$menuOn: true;

$menuDropDown: false;

.menu li {

color: #ff0000;

text-decoration: underline;

@if $menuOn == true and $menuDropDown == true {

position:relative;

}

}

компилируется в:

.menu li {

color: #ff0000;

text-decoration: underline;

}

**СИМВОЛ "&" В SASSSCRIPT**

Также как и в объявлении ссылки на родителя, оператор & в SassScript относится к текущему родительскому селектору. Это список, разделенный запятыми, состоящий из списков, разделенных пробелами. Например:

.foo.bar .baz.bang, .bip.qux { $selector: &; }

В этом примере значением $selector является ((".foo.bar" ".baz.bang"), ".bip.qux"). Составные селекторы приведены, чтобы показать, что они строковые, хоть и указывались без кавычек. Даже если родительский селектор не содержит запятую или пробел, **&** всегда будет иметь два уровня вложенности, поэтому он может быть доступен постоянно.

Если нет родительского селектора, значение **&** будет нулевым. Это означает, что вы можете использовать его в миксинах, чтобы проверить наличие родительского селектора:

@mixin does-parent-exist {

@if & {

&:hover { color: red; }

} @else {

a { color: red; }

}

}

**МЕТКА !DEFAULT**

Вы можете присваивать значение по-умолчанию переменным, у которых еще нет значения, добавив метку **!default** в конце значения. В таком случае, если переменная уже имела значение, то оно не изменится; если же переменная пуста, ей будет присвоено новое указанное значение.

Переменные, имеющие значение null, рассматриваются меткой !default как не имеющие значение:

$content: null;

$content: "НЕ-null контент" !default;

#main { content: $content; }

компилируется в

#main { content: "НЕ-null контент"; }

1. **ПРАВИЛА И ДИРЕКТИВЫ**

**ДИРЕКТИВА @IMPORT**

Sass расширяет CSS правило **@import**, позволяя импортировать SCSS и SASS файлы. Все импортированные SCSS и SASS файлы могут быть объеденены в одном результируещем CSS-файле. Кроме того, любые переменные или миксины, объявленные в импортируемые файлы, могут использоваться в главном файле.**@import** использует название файла для импорта. По-умолчанию, @**import** ищет Sass-файлы, но есть несколько правил, по которым @import отрабатывает как CSS-правило:

* Если расширение файла .css
* Если имя файла начинается с http://
* Если имя файла вызывается через url()
* Если правило @import включает в себя любые медиа-запросы

Импортирование может содержать в себе интерполяцию **#{},** но только с некоторыми ограничениями. Невозможно динамически импортировать файлы Sass через переменные; интерполяция нужна только для правила @import в CSS. Как таковая, интерполяция работает только с **url().**

$family: unquote("Droid+Sans");

@import url("http://fonts.googleapis.com/css?family=#{$family}");

скомпилируется в

@import url("http://fonts.googleapis.com/css?family=Droid+Sans");

**ФРАГМЕНТЫ**

Если у Вас есть SCSS или SASS файл, который вы хотите импортировать, но не хотите его компилировать напрямую в CSS-файл, то добавьте нижнее подчеркивание в начало имени файла. Это сообщит Sass о том, что не нужно компилировать этот файл напрямую в CSS. Когда импортируете такой файл, то нижнее подчеркивание можно не указывать.

Например, у вас есть файл **\_color.scss**. Никакого **\_colors.css** не будет создано, вам всего лишь надо написать:

@import "colors";

и файл **\_color.scss** будет импортирован.

Обратите внимание на то, что невозможно импортировать фрагмент и полноценный файл Sass c одинаковым именем, а также и находящимися в одной папке, одновременно. Например, файл colors.scss не может сосуществовать с фрагментом \_colors.scss.

**ВЛОЖЕННЫЙ ИМПОРТ**  
Самое популярное место использования **@import** явлется начало документа, но с Sass директиву можно использовать и в CSS-свойствах и в медиа-условиях. Как и в базовом варианте, @import включает в себя контент импортируемого файла. Тем не менее, импортированные правила будут вложены в том же месте, где и вызывалась директива **@import**

Например, файл **example.scss** содержит в себе:

.example { color: red; }

далее импортируем

#main { @import "example"; }

компилируется в

#main .example { color: red; }

Директивы, которые могут использоваться только на базовом уровне документа, такие как **@mixin** и **@charset,** **не допускаются** в файлах, которые подключаются вложенным импортом.

Невозможно вкладывать **@import** в миксин или директиву управления.

**ДИРЕКТИВА (@MEDIA)**

Директива **@media** работает также как и стандартное правило CSS, только с дополнительными возможностями: директива может вкладываться в правила CSS. Если директива вложена в CSS-правило, то при компиляции она будет поднята наверх таблицы стилей, а все селекторы в которых была директива переместятся внутрь **@media**

.sidebar {

width: 300px;

@media screen and (orientation: landscape) { width: 500px; }

}

компилируется в

.sidebar { width: 300px; }

@media screen and (orientation: landscape) {

.sidebar { width: 500px; }

}

**@media** запросы могут вкладываться друг в друга. После компиляции эти @media будут объеденены оператором **and**.

@media screen {

.sidebar {

@media (orientation: landscape) {

width: 500px;

}

}

}

компилируется в

@media screen and (orientation: landscape) {

.sidebar { width: 500px; }

}

Ну и наконец, **@media** запросы могут содержать в себе все возможности SassScript(включая переменные, функции и операторы) в именах компонентов и различных значений. Например:

$media: screen;

$feature: -webkit-min-device-pixel-ratio;

$value: 1.5;

@media #{$media} and ($feature: $value) {

.sidebar { width: 500px; }

}

компилируется в

@media screen and (-webkit-min-device-pixel-ratio: 1.5) {

.sidebar { width: 500px; }

}

**ДИРЕКТИВА @EXTEND**

Используя директиву **@extend** можно наследовать наборы свойств CSS от одного селектора другому.

.message {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

.success {

@extend .message;

border-color: green;

}

.hoverlink {

@extend a:hover;

}

a:hover {

text-decoration: underline;

}

Одиночный селектор может быть расширен больше, чем одним селектором. Это означает, что он наследует все стили всех расширяющих селекторов.

.seriousError {

@extend .error;

@extend .attention;

border-width: 3px;

}

**СЕЛЕКТОРЫ РАСШИРЕНИЯ**

Шаблонные селекторы выглядят также как и селекторы классов или идентификаторов, только . или # заменены на **%**. Они могут быть использованы в любом месте, где могут находится классы или идентификаторы, и сами по себе они не дают набора правил в генерируемом CSS файле.

// Данный набор правил не будет сгенерирован

#context a%extreme {

color: blue;

font-weight: bold;

font-size: 2em;

}

Tем не менее, шаблонные селекторы могут быть расширены, также как классы или идентификаторы. Расширенные селекторы будут сгенерированы, а базовый шаблонный селектор - нет. Например:

.notice {

@extend %extreme;

}

компилируется в

#context a.notice {

color: blue;

font-weight: bold;

font-size: 2em;

}

**МЕТКА !OPTIONAL**

Иногда, если нужно, вы можете разрешить директиве @extend не создавать каких-либо новых селекторов. Сделать это можно добавив метку неоязательности !optional после селектора. Например:

a.important {

@extend .notice !optional;

}

**ВЛОЖЕНИЕ @EXTEND В ДИРЕКТИВЫ**

Если вы используете **@extend** в **@media** (или в других директивах), то вы можете расширить только те селекторы, которые находятся в том же блоке директивы.

Правильное использование:

@media print {

.error {

border: 1px #f00;

background-color: #fdd;

}

.seriousError {

@extend .error;

border-width: 3px;

}

}

и неправильное:

.error {

border: 1px #f00;

background-color: #fdd;

}

@media print {

.seriousError {

// INVALID EXTEND: .error is used outside of the "@media print" directive

@extend .error;

border-width: 3px;

}

}

Некорректное расширение: .error используется вне директивы "@media print"

**Управляющие директивы и выражения**

SassScript поддерживает базовые управляющие директивы и выражения для подключения стилей при определённых условиях или подключения одних и тех же стилей несколько раз с изменениями.

**ДИРЕКТИВА @IF**

Принимает выражение SassScript и использует стили, вложенные в неё в случае, если выражение возвращает любое значение, кроме false или null.

После выражения @if может следовать несколько выражений @else if и одно выражение @else. Если выражение @if вернёт false, то будут произведиться попытки вычисления выражений @else if, пока одно из них не вернёт истину или до достижения выражения @else.

$type: monster;

p {

@if $type == ocean {

color: blue;

} @else if $type == matador {

color: red;

} @else if $type == monster {

color: green;

} @else {

color: black;

}

}

компилируется в

p {

color: green; }

**Директива @For**

Директива **@for** выводит набор стилей заданное число раз. Для каждого повторения используется переменная-счётчик для изменения вывода.

Директива имеет две формы: @for $var from <начало> through <конец> и @for $var from <начало> to <конец>. Заметьте различие в словах through и to.

$var может быть любым именем переменной, таким как $i; <начало> и <конец> - выражения SassScript, которые должны возвращать целые числа. Если <начало> больше, чем <конец>, счётчик будет убывать, вместо того, чтобы расти.

Выражение @for устанавливает переменную $var в каждое последующее значение в заданном диапазоне и каждый раз выводит вложенные стили, используя очередное значение переменной $var. Форма from ... through, диапазон включает значения <начало> и <конец>, а форма from ... to не включает значение <конец>. При использовании синтаксиса through,

@for $i from 1 through 3 {

.item-#{$i} { width: 2em \* $i; }

}

компилируется в

.item-1 {

width: 2em; }

.item-2 {

width: 4em; }

.item-3 {

width: 6em; }

**ДИРЕКТИВА @EACH**

Директива **@each** обычно имеет вид **@each $var in** <список или карта значений>. **$var** может быть любой переменной, такой как **$length** или **$name,** и <список или карта значений> - это выражение SassScript, возвращающее список или карту значений.

Директива **@each** устанавливает $var в каждое из значений списка или карты и выводит содержащиеся в ней стили, используя соответствующее значение **$var**. Например:

@each $animal in puma, sea-slug, egret, salamander {

.#{$animal}-icon {

background-image: url('/images/#{$animal}.png');

}

}

компилируется в

.puma-icon {

background-image: url('/images/puma.png'); }

.sea-slug-icon {

background-image: url('/images/sea-slug.png'); }

.egret-icon {

background-image: url('/images/egret.png'); }

.salamander-icon {

background-image: url('/images/salamander.png'); }

**Множественные значения**

Директива **@each** также может использовать несколько переменных по следующему принципу: **@each $var1, $var2, ... in** <список>. Если <список> - это список списков, то значение каждого элемента внутреннего списка будет назначено соответстввующей переменной. Например:

@each $animal, $color, $cursor in (puma, black, default),

(sea-slug, blue, pointer),

(egret, white, move) {

.#{$animal}-icon {

background-image: url('/images/#{$animal}.png');

border: 2px solid $color;

cursor: $cursor;

}

}

компилируется в

.puma-icon {

background-image: url('/images/puma.png');

border: 2px solid black;

cursor: default; }

.sea-slug-icon {

background-image: url('/images/sea-slug.png');

border: 2px solid blue;

cursor: pointer; }

.egret-icon {

background-image: url('/images/egret.png');

border: 2px solid white;

cursor: move; }

Так как мапы обрабатываются как списки или пары значений, множественное присваивание с ними тоже работает. Например:

@each $header, $size in (h1: 2em, h2: 1.5em, h3: 1.2em) {

#{$header} {

font-size: $size;

}

}

компилируется в

h1 {

font-size: 2em; }

h2 {

font-size: 1.5em; }

h3 {

font-size: 1.2em; }

**ДИРЕКТИВА @WHILE**

Директива **@while** принимает выражение SassScript и циклично выводит вложенные в неё стили, пока выражение вычисляется как true. Она может быть использована для создания более сложных циклов, чем тех, для которых подходит **@for**, хотя она бывает необходима довольно редко. Например:

$i: 6;

@while $i > 0 {

.item-#{$i} { width: 2em \* $i; }

$i: $i - 2;

}

компилируется в

.item-6 { width: 12em; }

.item-4 { width: 8em; }

.item-2 { width: 4em; }

1. **МИКСИНЫ**

Миксины (примеси) позволяют определить стили, которые могут быть использованы повторно в любом месте документа

@mixin large-text {

font: {

family: Arial;

size: 20px;

weight: bold;

}

color: #ff0000;

}

Миксины подключаются так же, как и @extend, только командой **@include**:

.page-title {

@include large-text;

padding: 4px;

margin-top: 10px;

}

Миксины могут быть также вызваны вне какого-либо правила (то есть в корне документа), при условии, что они не определяют непосредственно правил и не используют ссылку на родителя. Например:

@mixin silly-links {

a {

color: blue;

background-color: red;

}

}

@include silly-links;

Так же миксины могут содержать в себе другие миксины

**АРГУМЕНТЫ МИКСИНОВ**

Миксины могут принимать значения SassScript как аргументы, которые передаются при подключении миксина и становятся доступными как переменные внутри него.

@mixin border-radius($radius) {

-webkit-border-radius: $radius;

-moz-border-radius: $radius;

-ms-border-radius: $radius;

border-radius: $radius;

}

.box { @include border-radius(10px); }

При определении миксина, аргументы, пишутся как имена переменных, разделённые запятыми, внутри круглых скобок сразу после имени. Затем, при подключении миксина, значения могут быть переданы аналогичным образом. Например:

@mixin sexy-border($color, $width) {

border: {

color: $color;

width: $width;

style: dashed;

}

}

p { @include sexy-border(blue, 1in); }

компилируется в

p {

border-color: blue;

border-width: 1in;

border-style: dashed; }

В миксинах также можно определять значения аргументов по-умолчанию, используя обычный синтаксис установки значений переменных. Затем, при подключении миксина, если ему не будет передан аргумент, то будет использовано значение по-умолчанию. Например:

@mixin sexy-border($color, $width: 1in) {

border: {

color: $color;

width: $width;

style: dashed;

}

}

p { @include sexy-border(blue); }

h1 { @include sexy-border(blue, 2in); }

Миксины также можно вызывать используя явно именованные аргументы. Например, вышеуказанный пример мог бы быть записан как:

p { @include sexy-border($color: blue); }

h1 { @include sexy-border($color: blue, $width: 2in); }

**ПЕРЕМЕННЫЕ АРГУМЕНТЫ**

Иногда имеет смысл миксину или функции принимать неизвестное количество аргументов. Например, миксин для создания тени блока может принимать любое количество значений тени. Для подобных ситуаций Sass поддерживает возможность передачи "переменных аргументов" - это такие аргументы, которые передаются последними в функцию или миксин, получают все остальные переданные параметры и упаковывают их в список. Эти аргументы выглядят как обычные, но после них следует многоточие. Например:

@mixin box-shadow($shadows...) {

-moz-box-shadow: $shadows;

-webkit-box-shadow: $shadows;

box-shadow: $shadows;

}

.shadows {

@include box-shadow(0px 4px 5px #666, 2px 6px 10px #999);

}

компилируется в

.shadows {

-moz-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;

-webkit-box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;

box-shadow: 0px 4px 5px #666, 2px 6px 10px #999;

}

Переменные аргументы также можно использовать в миксинах. Используя тот же синтаксис, можно развернуть список значений так, что каждое значение будет передано как отдельный параметр или развернуть карту значений так, что каждая пара будет трактована как именованный аргумент:

@mixin colors($text, $background, $border) {

color: $text;

background-color: $background;

border-color: $border;

}

$values: #ff0000, #00ff00, #0000ff;

.primary {

@include colors($values...);

}

$value-map: (text: #00ff00, background: #0000ff, border: #ff0000);

.secondary {

@include colors($value-map...);

}

компилируется в

.primary {

color: #ff0000;

background-color: #00ff00;

border-color: #0000ff;

}

.secondary {

color: #00ff00;

background-color: #0000ff;

border-color: #ff0000;

}

Вы можете передать одновременно список параметров и карту значений, но только если список передаётся перед мапом, как в этом примере: @include colors($values..., $map...).

**БЛОКИ КОНТЕНТА В МИКСИНАХ**

Существует возможность передачи блока стилей в миксин, которые будут расположены среди стилей, подключаемых этим миксином. Эти стили будут размещены вместо директив @content, расположенных внутри миксина. Это даёт возможность создания абстракций, зависящих от конструкций селекторов или директив. Например:

@mixin apply-to-ie6-only {

\* html {

@content;

}

}

@include apply-to-ie6-only {

#logo {

background-image: url(/logo.gif);

}

}

компилируется в

\* html #logo {

background-image: url(/logo.gif);

}

Переменные и миксины, которые используются в передаваемом блоке, зависят от других стилей, вблизи которых блок объявлен. Например:

#sidebar {

$sidebar-width: 300px;

width: $sidebar-width;

@include smartphone {

width: $sidebar-width / 3;

}

}

**ФУНКЦИИ**

Существует возможность определить собственные функции в Sass и использовать их в любом значении или контексте скрипта. Например:

$grid-width: 40px;

$gutter-width: 10px;

@function grid-width($n) {

@return $n \* $grid-width + ($n - 1) \* $gutter-width;

}

{ width: grid-width(5); }

компилируется в

#sidebar { width: 240px; }

Как вы видите, функции имеют доступ к любым глобальным переменным, а также принимают параметры как и миксины (примеси). Функция может содержать несколько операторов, и вы должны вызвать @return, чтобы установить возвращаемое значение функции.

Также, как и миксины, определённые в Sass функции могут быть вызваны с именованными аргументами. В предыдущем примере мы могли бы вызвать функцию так:

#sidebar { width: grid-width($n: 5); }

Рекомендуется использовать префиксы для функций во избежание конфликтов имён, а также, чтобы читатель ваших таблиц стилей понимал, что эти функции не являются частью Sass или CSS. Например, если бы вы работали в компании ACME, то функция выше могла бы быть названа вами как -acme-grid-width.