(Вводные: scss и sass в принципе одно и тоже, просто в sass нет {} и ;

- все делается через табы

Еще можно добавить Что и для чего: миксины, наследование и функции в Sass; Как запустить JavaScript функцию в SASS?; Как скрыть свойство Sass по условию?; Встроенные модули Sass; Все о математических функциях в Sass, но уже 18 страниц, мне кажется дофига, в основном брала из документации

Что-то переделать – нет проблем)

Что такое SCSS

**SCSS (Sassy CSS)** — это один из диалектов языка препроцессоров CSS, известный как Sass (Syntactically Awesome Style Sheets). SCSS расширяет возможности CSS, добавляя функции, которые делают разработку стилей более эффективной и удобной.

Переменные

Думайте о переменных, как о способе хранения информации, которую вы хотите использовать на протяжении написания всех стилей проекта. Вы можете хранить в переменных цвета, стеки шрифтов или любые другие значения CSS, которые вы хотите использовать. Чтобы создать переменную в Sass нужно использовать символ $. Рассмотрим пример:





Sass обрабатывается, он принимает значения, заданные нами в $font-stack и $primary-color и вставляет их в обычном CSS-файле в тех местах, где мы указывали переменные как значения. Таким образом переменные становятся мощнейшей возможностью, например, при работе с фирменными цветами, используемыми на всем сайте.

Вложенность (появилось в css)

При написании HTML, Вы, наверное, заметили, что он имеет четкую вложенную и визуальную иерархию. С CSS это не так.

Sass позволит вам вкладывать CSS селекторы таким же образом, как и в визуальной иерархии HTML. Но помните, что чрезмерное количество вложенностей делает ваш документ менее читабельным и воспринимаемым, что считается плохой практикой.

Чтобы понять что мы имеем ввиду, приведем типичный пример стилей навигации на сайте:



Вы заметили, что селекторы ul, li, и a являются вложенными в селектор nav? Это отличный способ сделать ваш CSS-файл более читабельным.

Миксины

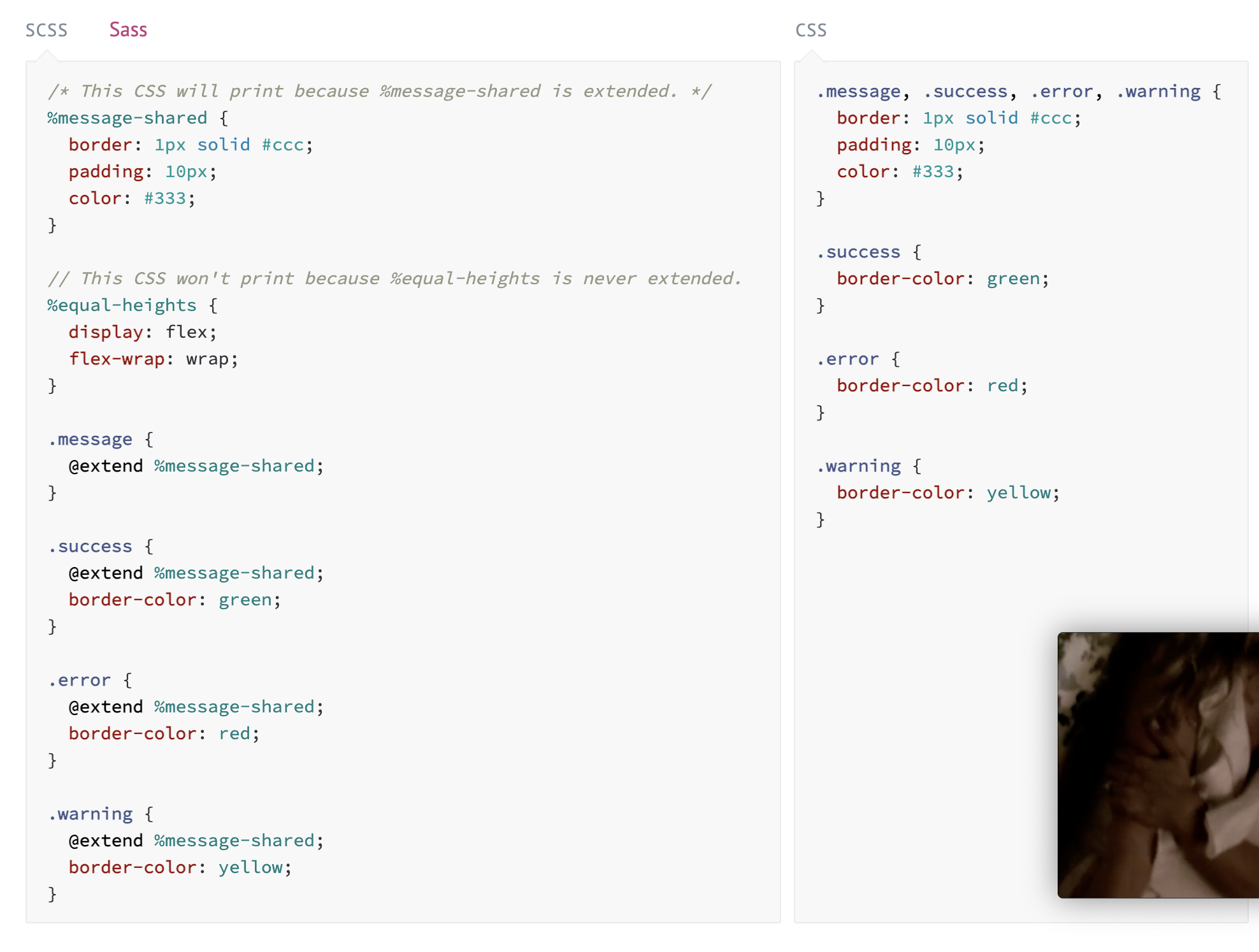
Некоторые вещи в CSS весьма утомительно писать, особенно в CSS3, где плюс ко всему зачастую требуется использовать большое количество вендорных префиксов. Миксины позволяют создавать группы деклараций CSS, которые вам придется использовать по нескольку раз на сайте. Вы даже можете передавать переменные в миксины, чтобы сделать их более гибкими. Так же хорошо использовать миксины для вендорных префиксов. Пример для transform:



Чтобы создать mixin, вы используете директиву @mixin и присваиваете ей имя. Мы назвали наше преобразование mixin . Мы также используем переменную $property внутри круглых скобок, чтобы мы могли передавать преобразование всего, что мы хотим. После того, как вы создадите свой mixin, вы можете использовать его как CSS-объявление, начинающееся с @include, за которым следует имя mixin.

Наследование

Это одна из самых полезных функций Sass. Используя директиву @extend можно наследовать наборы свойств CSS от одного селектора другому. Это позволяет держать ваш Sass-файл в «чистоте». В нашем примере мы покажем вам как сделать стили оповещений об ошибках, предупреждениях и удачных исходах, используя другие возможности Sass, которые идут рука-об-руку с расширением, классами-шаблонами. Класс-шаблон - особый тип классов, который выводится только при использовании расширения - это позволит сохранить ваш скомпилированный CSS чистым и аккуратным.



Вышеуказанный код сообщает классам .message, .success, .error и .warning вести себя как %message-shared. Это означает, что где бы не вызывался %message-shared, то и .message, .success, .error и .warning тоже будут вызваны. Магия происходит в сгенерированном CSS, где каждый из этих классов получает css-свойства, как и %message-shared. Это позволит вам избежать написания множества классов в HTML элементах.

Вы можете расширить большинство простых CSS селекторов прибавление к классам-шаблонам в Sass, однако, использование шаблонов - простейший способ быть уверенным, что вы не расширяете класс везде, где он используется в ваших стилях, что могло бы привести к непреднамеренным наборам стилей в вашем CSS.

Когда вы генерируете ваш CSS, то он будет выглядеть как пример ниже. Обратите внимание, %equal-heights не попадает в CSS, так как ни разу не был использован.

Операторы

Использовать математику в CSS очень полезно. Sass имеет несколько стандартных математических операторов, таких как +, -, \*, / и %. В нашем примере мы совершаем простые математические вычисления для расчета ширины aside и article.



Мы создали простую адаптивную модульную сетку, с шириной в 960 пикселей. Используя математические операторы, мы использовали полученные данные с пиксельными значениями и конвертировали их в процентные, причем без особых усилий

Асперанд

В SCSS используется директива &.



С помощью символа & вы можете явно указать, где должен быть вставлен родительский селектор.

Результат компиляции Sass (из предыдущего примера) в CSS ниже.



В итоге амперсанд был компилирован в название родительского элемента a (a:hover).

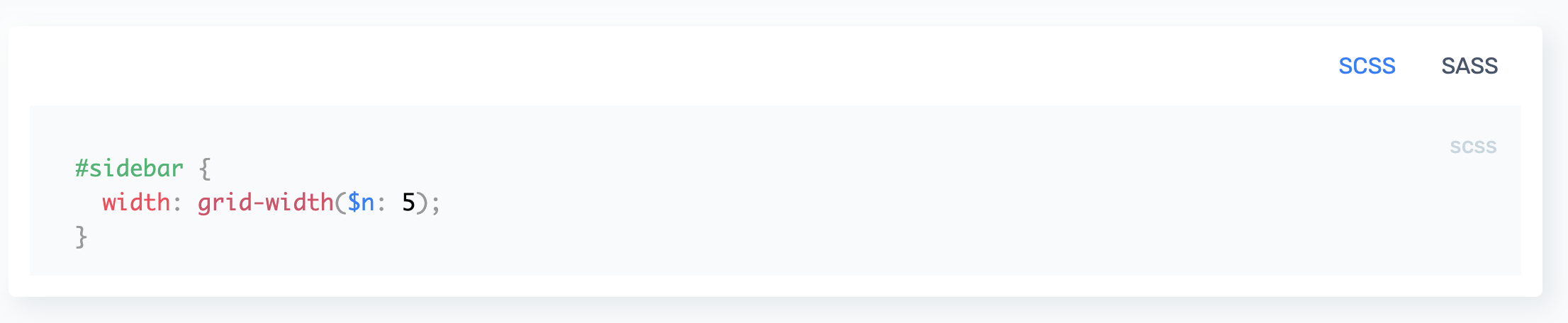
Функции

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



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:



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**.

Интерполяция

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



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:



Циклы и условия

Sass предоставляет набор инструментов, помогающих писать полноценные программы. Для этого он имеет встроенные функции, которые позволяют нам выполнять вычисления и операции, возвращающие определенное значение. Они варьируются от расчета цвета до математических операций, таких как получение случайных чисел и вычисление размеров, и даже условных выражений.

Также есть поддержка математических операторов, таких как **+**, **—**, **\**, **\***, **/**и **%**, которые мы можем использовать с функцией **calc**.

@use "sass:math";

@function pxToRem($pxValue) {

@return math.div($pxValue, 16px) \* 1rem;

}

div {

width: pxToRem(480px); // получим 30rem

}

Sass (SCSS)

Copy

## **Управляющая директива @if**

В SCSS директива **@if** позволяет условно применять стили на основе логического выражения. Это похоже на оператор if в программировании, и это может быть очень полезно для написания более динамичных и гибких стилей.

Вот пример того, как вы могли бы использовать директиву @if в SCSS:

$button-style: 'primary';

.button {

@if $button-style == 'primary' {

background-color: blue;

color: white;

} @else if $button-style == 'secondary' {

background-color: gray;

color: black;

} @else {

background-color: white;

color: black;

}

}

Sass (SCSS)

В этом примере мы определяем переменную с именем **$button-style** и устанавливаем ей значение ‘primary’. Затем мы используем директиву **@if**, чтобы проверить значение этой переменной и применить различные стили на основе ее значения.

Если **$button-style** равен ‘primary’, элемент **.button** будет иметь синий цвет фона и белый цвет текста. Если оно равно «secondary», цвет фона будет серым, а цвет текста — черным. В противном случае, цвет фона будет белым, а цвет текста — черным.

Директива **@else if** позволяет вам связать несколько условий вместе, а директива **@else** предоставляет параметр по умолчанию, если ни одно из условий не является истинным.

## **Управляющая директива @for**

В SCSS директива **@for** позволяет перебирать диапазон значений и динамически генерировать стили CSS. Это похоже на цикл for в программировании, и он может быть очень полезен для генерации повторяющихся стилей или создания сложных макетов.

Вот пример того, как вы могли бы использовать директиву @for в SCSS:

@for $i from 1 through 3 {

.box-#{$i} {

width: 100px \* $i;

height: 100px \* $i;

}

}

Sass (SCSS)

В этом примере мы используем директиву **@for** для создания трех элементов **.box** с увеличивающимися ширинами и высотами. Переменная **$i** устанавливается в диапазоне от 1 до 3 с помощью ключевого слова **through**, и мы используем интерполяцию строк (#{}), чтобы сгенерировать уникальное имя класса для каждого элемента (box-1, box-2, box-3).

Внутри цикла мы устанавливаем свойства **width** и **height** каждого элемента **.box** на основе текущего значения **$i**. Оператор **\*** позволяет нам умножить значение **$i** на фиксированное значение (в данном случае 100 пикселей), поэтому первый элемент **.box** будет иметь ширину и высоту 100 пикселей, второй будет иметь ширину и высоту 200 пикселей, а третий будет иметь ширину и высоту 300 пикселей.

## **Управляющая директива @each**

В SCSS директива **@each** позволяет вам выполнять итерации по списку значений и динамически генерировать стили CSS. Это похоже на цикл for each в программировании, и это может быть очень полезно для генерации повторяющихся стилей или создания сложных макетов.

Вот пример того, как вы могли бы использовать директиву @each в SCSS:

$colors: (

primary: #336699,

secondary: #993366,

tertiary: #669933

);

@each $name, $color in $colors {

.bg-#{$name} {

background-color: $color;

}

}

Sass (SCSS)

В этом примере мы определяем карту с именем **$colors**, которая содержит набор пар ключ-значение. Каждый ключ представляет название цвета (**primary**, **secondary**, **tertiary**), а каждое значение представляет цветовой код (#336699, #993366, #669933).

Затем мы используем директиву **@each** для перебора каждой пары ключ-значение в карте **$colors**. Внутри цикла мы используем интерполяцию строк, чтобы сгенерировать уникальное имя класса для каждого цвета (**bg-primary**, **bg-secondary**, **bg-tertiary**), и мы устанавливаем свойству **background-color** каждого элемента соответствующий цветовой код.

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:



Правила и директивы

Sass поддерживает все CSS3 @-правила, а также дополнительные специфические правила Sass, называемые "директивами". Они дают различные эффекты в Sass, которые мы рассмотрим далее.

# **Директива @import**

ahfuvtyns

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 \_colors.scss. Никакого \_colors.css не будет создано, вам всего лишь надо написать:

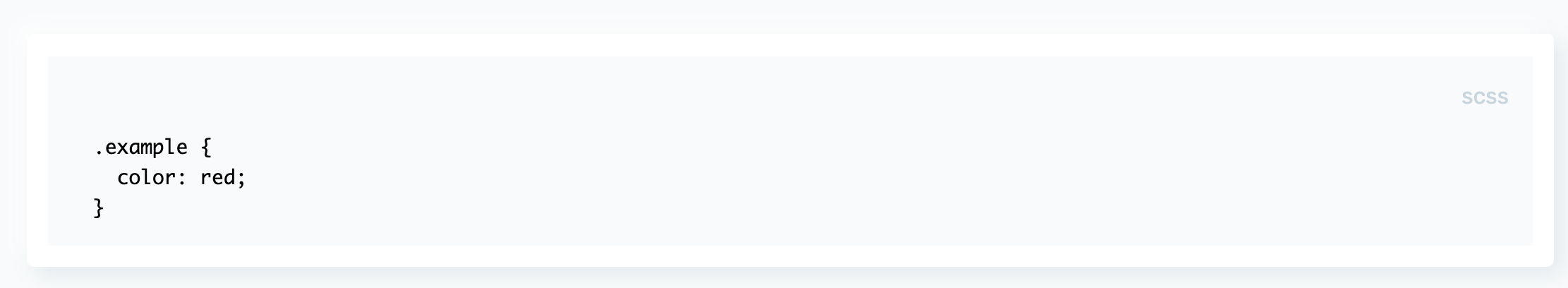
@import "colors";

и файл \_colors.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**

Самым популярным местом 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:



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

# **Директива @media**

Директива @media работает так же, 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. Такой метод позволяет легко добавлять правила в @media без повторения селекторов или нарушения потока таблицы стилей.



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

* 

@extend работает по принципу добавления расширяющего селектора (в нашем случае это ***.seriuosError***) в любой набор стилей, где встречается расширяемый селектор (в нашем случае - ***.error***). Таким образом, пример из предыдущей главы:



При слиянии селекторов директива @extend не допустит ошибок в виде дублирования, например, ***.seriuosError.seriuosError***, самостоятельно исправляя данные ошибки на ***.seriuosError***. Кроме того, директива не будет создавать селекторы, которые не смогут ничего значить, например, как ***#main#footer***.

# **Директива @at-root**

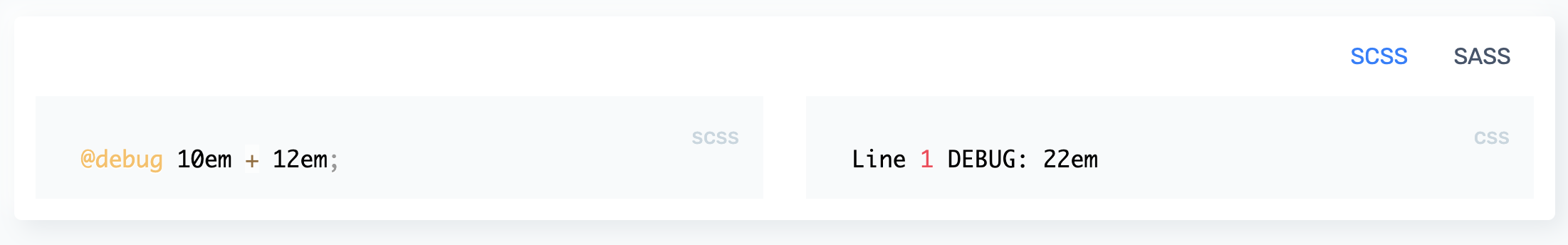
@at-root изымает одно или несколько правил из родительского селектора в корневой уровень документа. Такой способ можно использовать с селекторами первого уровня вложения:

Также этот способ может быть использован с несколькими селекторами:

* 

# **Директива @debug**

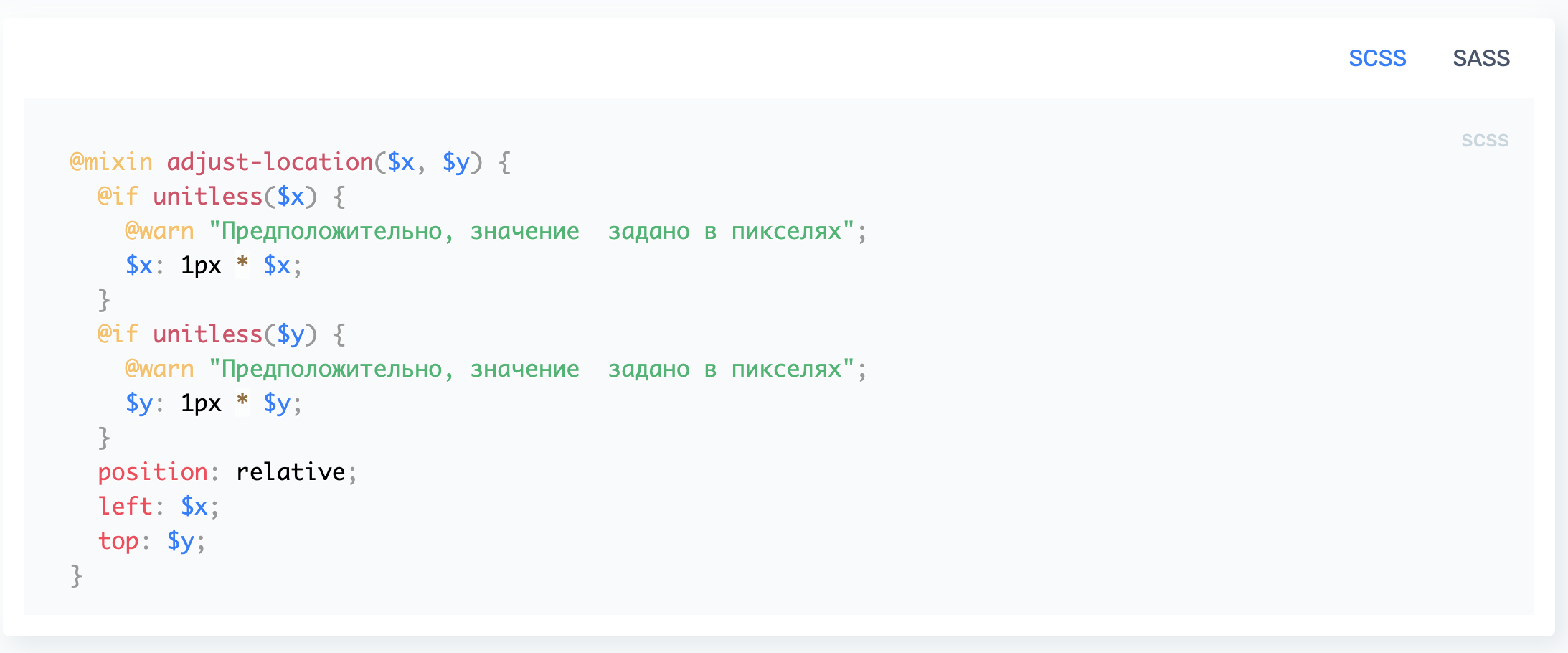
@debug выводит значения функций Sass средствами стандартного вывода ошибок. Это полезно для отладки функций Sass, особенно для тех, что содержат сложную структуру данных.



# **Директива @warn**

@warn выводит значения выражений Sass средствами стандартного вывода ошибок. Данная директива полезна для библиотек, которым нужно предупреждать пользователей об использовании устаревших или восстановленных после незначительных ошибок миксинов. Есть два основных различия между warn и debug:

* Вы можете отключить отображение предупреждений флагом --quiet в командной строке или опцией Sass :quiet.
* Предупреждение будет выведено вместе с отрывком таблицы стилей, чтобы пользователь мог понять, где в коде требуется его внимание.



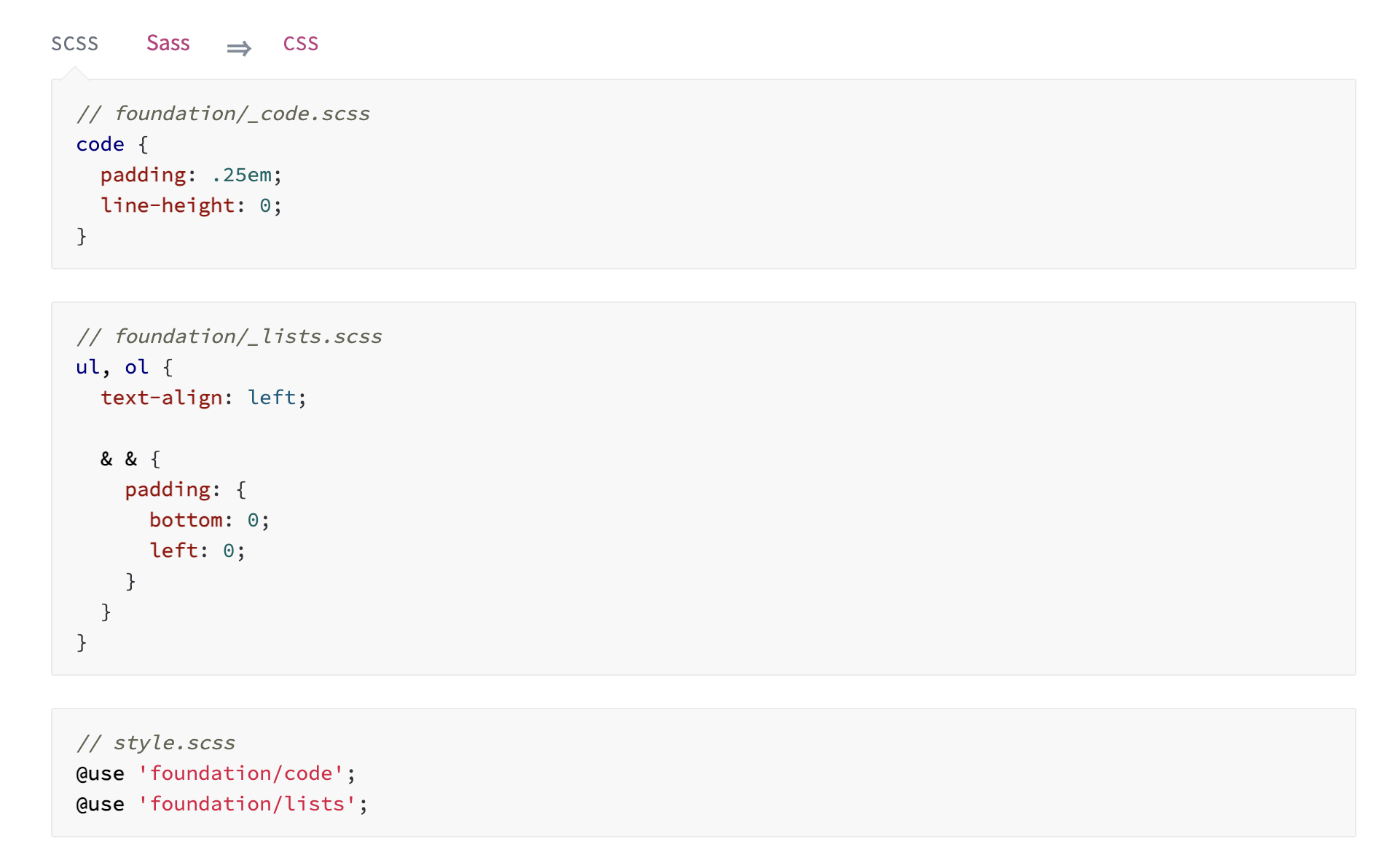
# **Директива @error**

@error отображает значение выражений и функций Sass как фатальную ошибку, включая нормальную часть стека трассировки. Эта директива полезна для проверки аргументов миксинов и функций.



Правило @use загружает [миксины](https://sass-lang.su/documentation/at-rules/mixin), [функции](https://sass-lang.su/documentation/at-rules/function) и [переменные](https://sass-lang.su/documentation/variables) из других таблиц стилей Sass и объединяет CSS из нескольких таблиц стилей вместе. Таблицы стилей, загруженные с помощью @use, называются "модулями". Sass также предоставляет [встроенные модули](https://sass-lang.su/documentation/modules), полные полезных функций.

Простейшее правило @use записывается как @use "<url>", которое загружает модуль по заданному URL. Любые стили, загруженные таким образом, будут включены ровно один раз в скомпилированный вывод CSS, независимо от того, сколько раз эти стили загружались.



Правило @forward загружает таблицу стилей Sass и делает ее [миксины](https://sass-lang.su/documentation/at-rules/mixin), [функции](https://sass-lang.su/documentation/at-rules/function) и [переменные](https://sass-lang.su/documentation/variables) доступными, когда ваша таблица стилей загружена с помощью [правила @use](https://sass-lang.su/documentation/at-rules/use). Это позволяет организовать библиотеки Sass по множеству файлов, позволяя пользователям загружать один файл точки входа.

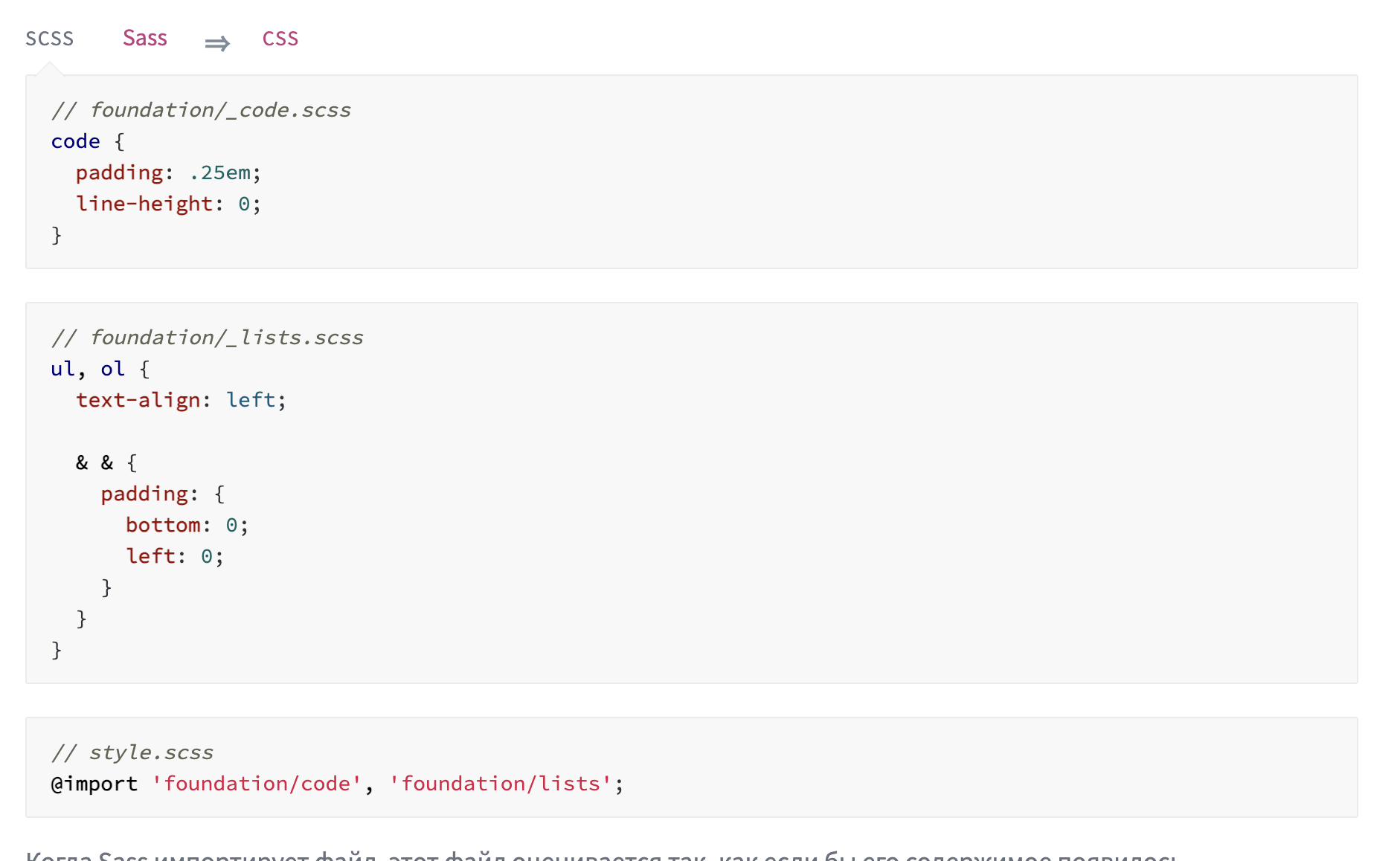
Правило написано @forward "<url>". Он загружает модуль по заданному URL точно так же, как @use, но делает участников [публичными](https://sass-lang.su/documentation/at-rules/use#private-members) загруженного модуля доступными для пользователей вашего модуля, как если бы они были определены непосредственно в вашем модуле. Однако эти участники недоступны в вашем модуле - если вы этого хотите, вам также необходимо написать правило @use. Не волнуйтесь, модуль загрузится только один раз!

Если вы @forward "<url>"пишете и @forward и @use для одного и того же модуля в одном файле, всегда рекомендуется сначала написать @forward. Таким образом, если ваши пользователи захотят [настроить перенаправленный модуль](https://sass-lang.su/documentation/at-rules/use#configuration), эта конфигурация будет применена к @forward до того, как ваш @use загрузит его без какой-либо конфигурации



Sass расширяет CSS [правило @import](https://developer.mozilla.org/en-US/docs/Web/CSS/@import) возможностью импортировать таблицы стилей Sass и CSS, обеспечивая доступ к [миксинам](https://sass-lang.su/documentation/at-rules/mixin), [функциям](https://sass-lang.su/documentation/at-rules/function) и [переменным](https://sass-lang.su/documentation/variables) и объединение CSS нескольких таблиц стилей вместе. В отличие от простого импорта CSS, который требует, чтобы браузер выполнял несколько HTTP-запросов при отображении вашей страницы, импорт Sass полностью обрабатывается во время компиляции.

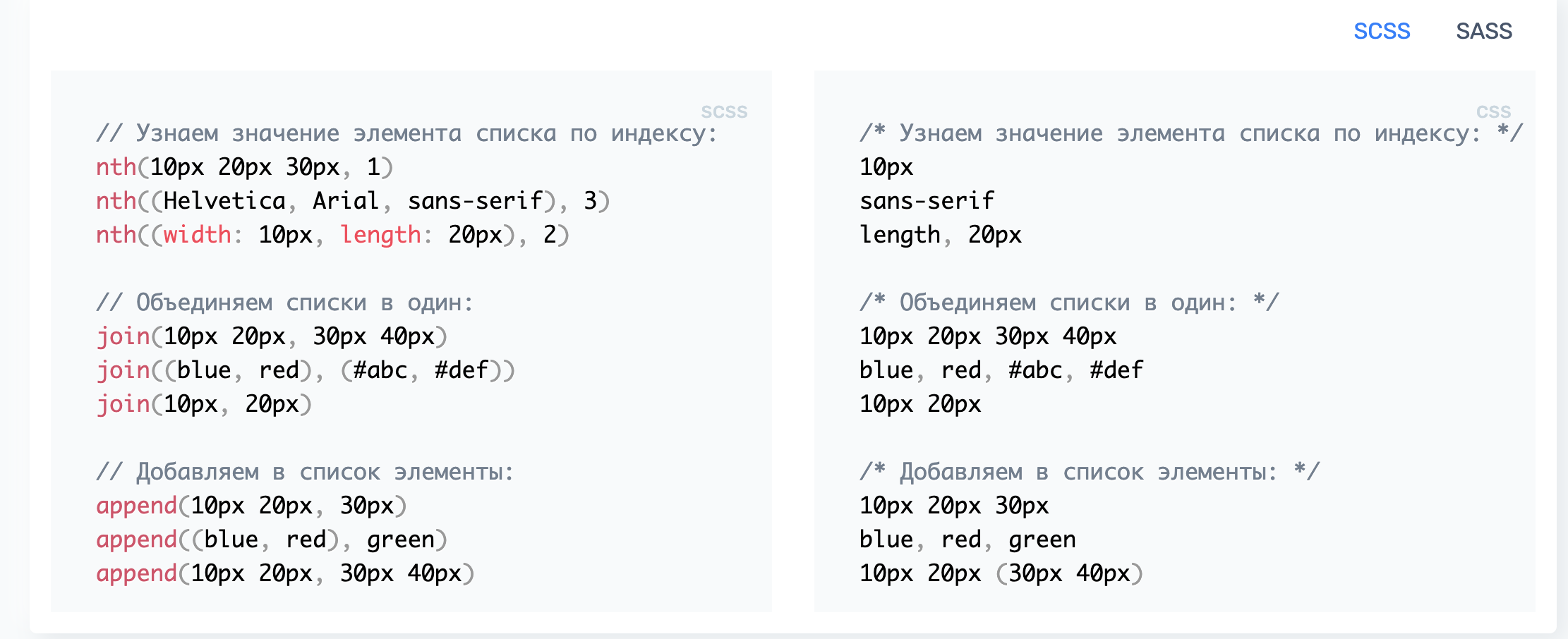
Импорт Sass имеет тот же синтаксис, что и импорт CSS, за исключением того, что он позволяет разделять несколько операций импорта запятыми, а не требует, чтобы каждый из них имел собственный @import. Кроме того, в [синтаксисе с отступом](https://sass-lang.su/documentation/syntax#the-indented-syntax) импортированные URL-адреса не обязательно должны иметь кавычки.



Списки

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. Функция 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.



В придачу к обычным значениям, списки могут содержать также другие списки. Например, ***1px 2px, 5px 6px*** - это список из двух элементов, который содержит список ***1px 2px*** и список ***5px 6px***. Если внутренние списки содержат тот же разделитель, что и внешний список, необходимо использовать скобки, чтобы было понятно где начинается и где заканчивается внутренний список. Например, ***(1px 2px) (5px 6px)*** - это список аналогичный предыдущему. Разница в том, что разделители у этого внешнего списка - пробелы, тогда как предыдущий список разделялся запятыми.

Когда списки преобразуются в обычный CSS, Sass не добавляет никаких скобок, так как CSS их не понимает. Это значит, что ***(1px 2px) (5px 6px)*** и ***1px 2px 5px 6px*** будут выглядеть идентично, когда они попадут в CSS. Тем не менее, это не то же самое для Sass: первый пример содержит два списка, тогда как второй содержит 4 числа.

Список может также не содержать значений вообще. Такой список объявляется как (), который также является пустым массивом. Они не могут быть непосредственно преобразованы в CSS, если вы попытаетесь написать что-то вроде font-family: (), то Sass выдаст ошибку. Если список содержит пустые списки или null-значения, например, как ***1px 2px () 3px*** или ***1px 2px null 3px***, то пустые списки и null-значения будут вырезаны при переводе в CSS.

Мапы (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:



Мапы не могут быть преобразованы в обычный CSS. Использование мапов в качестве значения переменной или аргумента CSS функции приведет к ошибке. Использование функций inspect($value) для получения результирующей строки полезно для отладки мапов. Например:



Операции с числами

SassScript поддерживает стандартные арифметические операции над числами (сложение +, вычитание -, умножение \*, деление / и остаток от деления по модулю %). Математические функции Sass сохраняют значение промежуточных результатов в течение выполнения арифметических операций. Это означает, что, как и в реальной жизни, вы не можете работать с числами у которых несовместимы типы данных (например, складывание рх и em) и двух чисел одного типа, перемножение которых даст квадратные единицы ( 10px \* 10px == 100px \* px ).

***Помните***, что ***px \* рх*** - это невалидный тип данных CSS, и вы получите сообщение об ошибке от Sass за попытку использовать недопустимые единицы в CSS.

Операторы сравнения ( <, >, <=, >=, ==, != ) также поддерживаются для чисел.