Полный гайд по SCSS/SASS



Далее я буду использовать названия **Sassy**, **Sass**и **SCSS**, поэтому давайте определимся, так как концептуально разница не велика, будем считать, что это одно и тоже. Вы поймёте различия по мере изучения темы. Отмечу, что **SCSS**сейчас наиболее популярна, вероятно потому, что является наиболее поздней версией оригинального **Sass**синтаксиса (некоторые считают её лучшей).

Чтобы по-настоящему почувствовать преимущества **Sass**, вам нужно понять ключевые идеи этого метаязыка. Об этом мы и поговорим.

Весь Sass/SCSS код компилируется обратно в стандартный CSS, поэтому браузер понимает его и корректно выводит результат. Текущие версии браузеров не поддерживают Sass/SCSS непосредственно (как и другие препроцессоры), а стандартная спецификация CSS не предлагает нам альтернатив (пока).

**Приступим!**

Пока вы хотя бы раз не попробуете сгенерировать значения для свойств с помощью цикла, вы не познаете настоящую мощь Sassy CSS. Но мы начнём с основ SCSS и постепенно доберёмся до самого интересного.

**Что такого есть в Sass/SCSS, чего нет в стандартном CSS?**

1. **Вложенные правила**: вы можете вкладывать CSS свойства, в несколько наборов скобок {}. Это сделает ваш CSS чище и понятней.
2. **Переменные**: в стандартном CSS тоже есть переменные, но переменные Sass куда более мощный инструмент. Например, вы можете использовать переменные в циклах и генерировать значения свойств динамически. Также можно внедрять переменные в имена свойств, например так: property-name-N { … }.
3. **Лучшая реализация операторов**: вы можете суммировать, вычитать, делить и умножать CSS значения. Sass реализация более интуитивна, чем стандартный функционал CSS calc().
4. **Функции**: Sass позволяет многократно использовать CSS стили, как функции.
5. **Тригонометрия**: помимо базовых операций (+, -, \*, /), SCSS позволяет писать собственные функции. Например, функции sin и cos можно написать, используя только синтаксис Sass/SCSS. Конечно, вам понадобятся знания тригонометрии. Такие функция могут понадобится для создания анимации.
6. **Удобный рабочий процесс**: вы можете писать CSS, используя конструкции, знакомые по другим языкам: for-циклы, while-циклы, if-else. Но имейте в виду, это только препроцессор, а не полноценный язык, Sass контролирует генерацию свойств и значений, а на выходе вы получаете стандартный CSS.
7. **Миксины**: позволяют один раз создать набор правил, чтобы потом использовать их многократно или смешивать с другими правилами. Например, миксины используют для создания отдельных тем макета.

**Препроцессор Sass**

Sass не динамичный язык. Он не позволяет генерировать или изменять CSS свойства и значения в реальном времени. Но вы можете генерировать их более эффективным способом и внедрять в стандартные свойства (например, для CSS анимации).

**Новый синтаксис**

SCSS на самом деле не добавляет ничего нового в сам язык CSS. Это просто новый синтаксис, который в большинстве случаев сокращает время написания стилей.

**Несколько фактов для начала**

CSS препроцессоры добавляют новый синтаксис в CSS.

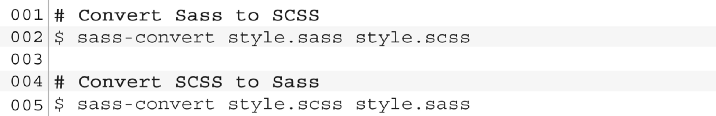
Существует 5 CSS препроцессоров: **Sass**, **SCSS**, **Less**, **Stylus**и **PostCSS**.

Здесь я буду говорить в основном о SCSS, который схож с Sass. О Sass можно почитать здесь: <https://www.sass-lang.com/>.

**SASS (.sass) S**yntactically **A**wesome**S**tyle**S**heets,

**SCSS (.scss)** **S**assy **C**ascading **S**tyle **S**heets. –Расшифровка sass.

Обратите внимание на расширения, они схожи, но отличаются. Для любителей командной строки, есть возможность конвертировать .sass в .scss и наоборот:



Конвертация форматов .scss и .sass. Используем команду sass-convert.

Sass — это первая спецификация Sassy CSS, её разработка началась в 2006 году. Позже разработали альтернативный синтаксис, который некоторые считают лучшей версией языка, а также изменилось первоначальное расширение .sass на новое .scss.

На данный момент ни один браузер не поддерживает CSS препроцессоры напрямую. Вы можете свободно экспериментировать с любым из 5 препроцессоров на сервисе **codepen.io**. Кроме того, используя препроцессор, вы должны будете установить его к себе на веб-сервер.

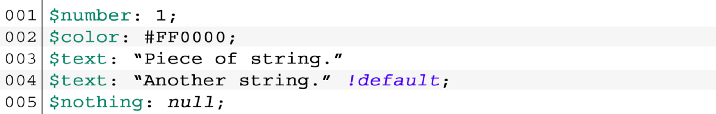
Эта статья о SCSS, но другие препроцессоры имеют схожие функции, хотя синтаксис может отличаться.

**Superset**

Sassy CSS в любом своём проявлении является «расширением» языка CSS, а значит, всё что работает в CSS, работает и в Sass/SCSS.

**Переменные**

В **Sass/SCSS** есть переменные, и они отличаются от тех, которые вы вероятно видели в CSS — они начинаются с двух тире (--color: #9c27b0). В SCSS переменная обозначается знаком доллара ($color: #9c27b0).



Определение переменных

Вы можете перезаписать имя переменной или задать для неё значение по умолчанию. Для этого добавьте метку **!default** к переменной, и если её значение не изменится в дальнейшем и не будет пустым, то будет использоваться заданное значение по умолчанию.

В примере выше окончательным значением переменной **$text** будет **“Piece of string.”**, а не **“Another string.”**, потому что переменная не пуста.



Sass переменные можно назначить любому свойству CSS

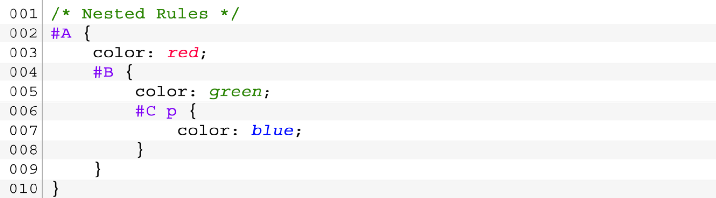
**Вложенные правила**

В стандартном CSS, вложенность достигается путём расстановки пробелов.



Вложенность в стандартном CSS

С помощью Sassy CSS этот код можно записать так:



Вложенность в Sass. Меньше повторений

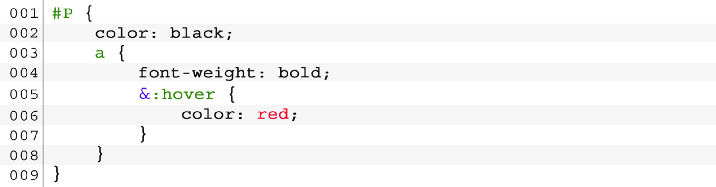
Конечно, в итоге это всё скомпилируется в обычный CSS. Это просто синтаксис, но как видите, он выглядит чище и с меньшими повторениями.

В частности, это помогает работать со сложными макетами. Выравнивание, в котором вложенные свойства записаны в коде, точно соответствует структуре макета приложения.

Тем временем под капотом, препроцессор компилирует всё в стандартный код CSS (как в первом примере), чтобы браузер смог его отобразить. Мы лишь изменили способ написания CSS.

**Оператор &**

В Sassy CSS можно использовать оператор & (И), давайте посмотрим, как это работает.



Используем символ &

В 5 строке мы видим **&:hover**, после компиляции, вместо символа &, подставляется имя родительского элемента, т.е. **a:hover**. В результате у нас получится такой код:

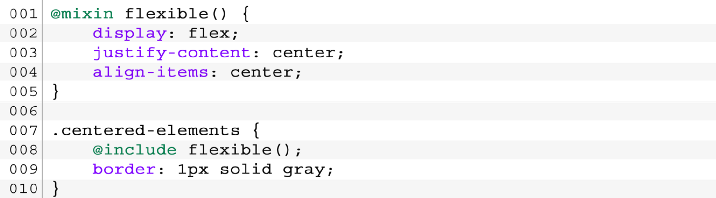
https://miro.medium.com/max/1400/1*V5CV8TYsZEyPGs2BR_IMUQ.png

Результат компиляции SCSS

**Миксины**

Для объявления миксина используйте директиву **@mixin** (ещё это называют **mixin rule**).

Давайте создадим миксин, который определяет поведение элемента как Flex:



Миксины

Теперь любой HTML элемент с классом **.centered-elements** будет иметь свойства Flexbox. Вы можете использовать миксины совместно с другими CSS свойствами.

Для класса **.centered-elements** я добавил миксин и правило **border:1px solid gray;**

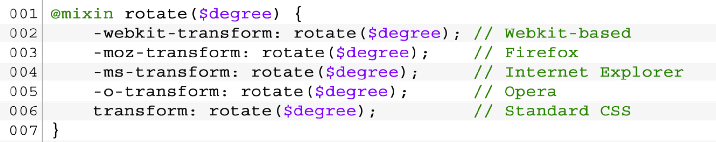
Вы можете передавать аргументы в **@mixin**, как если бы это была функция, а затем присваивать их свойствам CSS. Рассмотрим это в следующей части.

**Кросс-браузерность. Пример**

Некоторые экспериментальные фичи работают только в определённых браузерах.

С помощью миксин удобно определять CSS свойства, индивидуальные для каждого браузера, в одном классе.

Например, если вам нужно вращать элемент в браузере на движке Webkit, так же, как и в других браузерах, вы можете создать миксин, который принимает переменную **$degree** в качестве аргумента:



Реализуем кросс-браузерность с помощью миксин

Теперь нужно подключить (**@include**) этот миксин в CSS класс:

https://miro.medium.com/max/1400/1*pEfm4POcLW3QOe1F6QJoPw.png

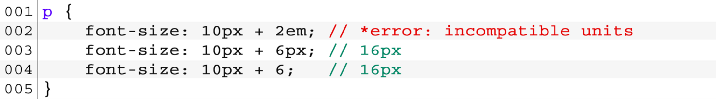
Правило Rotate совместимое со всеми браузерами

**Арифметические операции**

Как и в стандартном CSS, вам доступны операции **сложения**, **вычитания**, **умножения**и **деления**, но в отличии от классического CSS синтаксиса, вам необязательно использовать функцию **calc()**.

Есть несколько неочевидных особенностей, которые следует учитывать, чтобы избегать ошибок.

**Сложение**



Сложение без функции calc()

Убедитесь, что оба значения соответствуют одному формату.

**Вычитание**

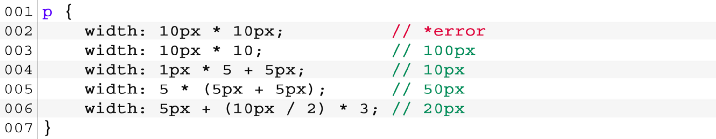
Этот оператор работает так же, как и предыдущий.



Вычитание

**Умножение**

Для умножения используется звёздочка, как и в стандартном CSS **calc(a \* b)**.

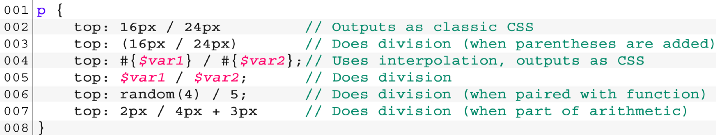


умножение и деление

**Деление**

Так как в стандартном CSS символ деления зарезервирован для других целей, есть некоторые особенности его использования. Например, font: **24/32px**определяет размер шрифта **25px** и **line-height 32px**. Т.е. символ деления используется для сокращённой записи свойств шрифта. Посмотрим, как работает SCSS, сохраняя совместимость со стандартным CSS.

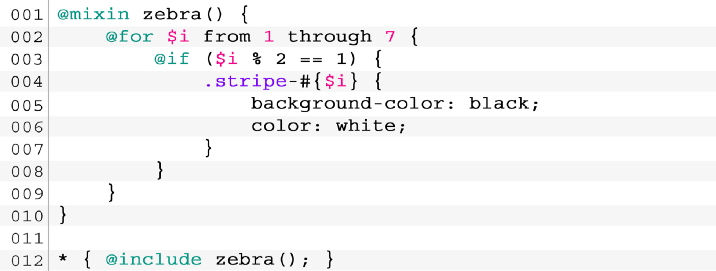
https://miro.medium.com/max/1400/1*AYaiskIAzTUhYNUFRyo0ag.png



Чтобы разделить два значения, просто оберните выражение в скобки. Иначе, оператор деления сработает только при использовании с другими операторами или функциями.

**Остаток**

Вы можете вычислить остаток от операции деления. Давайте попробуем создать «зебру» для набора HTML элементов.

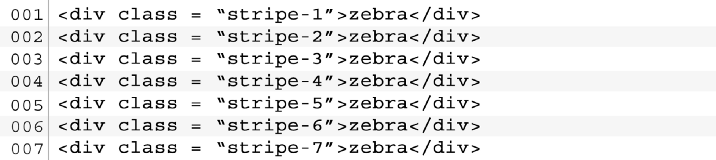


Создание зебры

Начнём с создания миксин.

**Примечание**: далее мы ещё поговорим о правилах **@for** и **@if**

Нам понадобится несколько HTML элементов:



HTML код

Вывод в браузере:

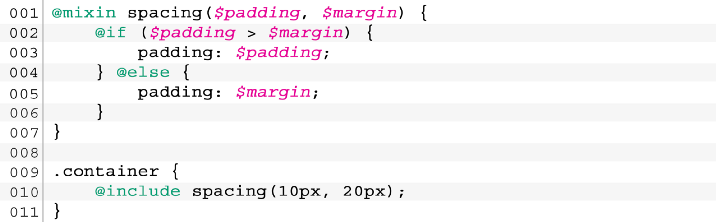


Сгенерированная зебра

**Операторы сравнения**

* **x==y** возвращает **true,** если **x**и **y** равны
* **x!=y**возвращает **true,**если **x**и **y** не равны
* **x>y**возвращает **true,**если **x**больше, чем **y**
* **x<y**возвращает **true,**если **x**меньше, чем **y**
* **x>=y**возвращает **true,**если **x**больше или равно**y**
* **x<=y**возвращает **true,**если **x**меньше или равно**y**

Чтобы показать, как операторы сравнения работают на практике, создадим миксин, который будет применять свойство padding, если его значение больше значения margin:



Операторы в действии

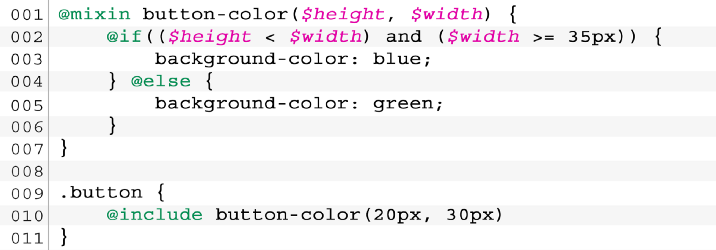
После компиляции, мы получим такой CSS:

https://miro.medium.com/max/1400/1*mEUStbPAo-MK-pjK06taJA.png

Результат сравнения

**Логические операторы**

* **x and y**возвращает **true**, если**x**и**y**равны **true**
* **x or y**возвращает **true**, если**x**или **y**равны **true**
* **not x**возвращает **true**, если**x**не равен**true**

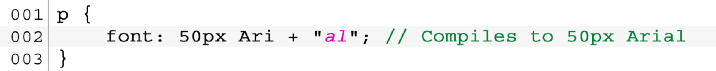


Применение логических операторов

Этот код создаёт класс button-color, который изменяет background-color в зависимости от ширины кнопки.

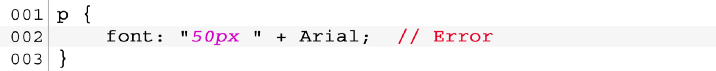
**Строки**

В некоторых случаях можно добавить строки в допустимые значения CSS без кавычек, но только если добавленная строка является завершающим элементом:



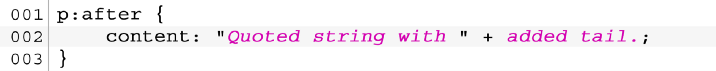
Комбинируем CSS значения свойства со строкой Sass/SCSS

А следующий пример приводит к ошибке:

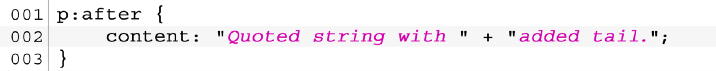


Этот код не сработает

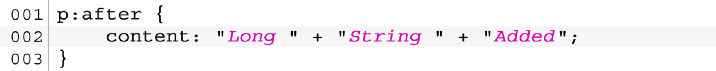
Можно складывать строки без двойных кавычек, если они не содержат пробелов. Пример ниже не будет скомпилирован:



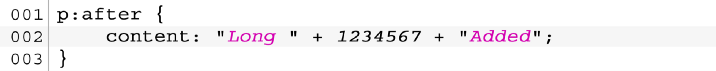
Этот пример тоже не сработает



Строка, содержащая пробелы, должна быть в кавычках



Складываем несколько строк



Складываем строки и числа

*Примечание: свойство content работает только с псевдо-селекторами :before и :after. Не рекомендуется использовать это свойство в CSS, вместо этого, следует указывать его между HTML тегов.*

**Операторы управления потоками**

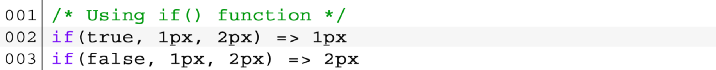
В SCSS есть функции (**fucntion()**) и директивы (**@directive**), их ещё называю «правилами». Мы уже создавали функцию, когда проходили миксины. Вы можете передавать аргументы функциям.

У функции обычно есть скобки, которые ставятся сразу после её имени. Директива/правило начинается с символа @.

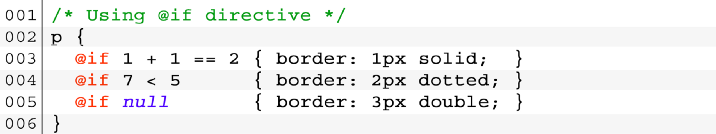
Как и в JavaScript или других языках, в SCSS есть стандартный набор операторов управления потоками.

**if()**

**if()**— это функция. Применяется она очень просто: оператор вернёт одно из двух значений, в зависимости от условия:

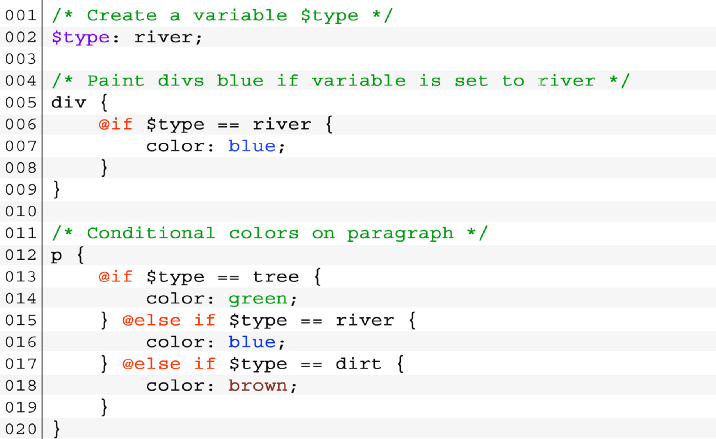


**@if** — это директива, её используют для создания ветвлений на основе условия.



Скомпилированный Sassy код:

https://miro.medium.com/max/1400/1*ilbB3zWHQC2vAuFVWL0w1w.png

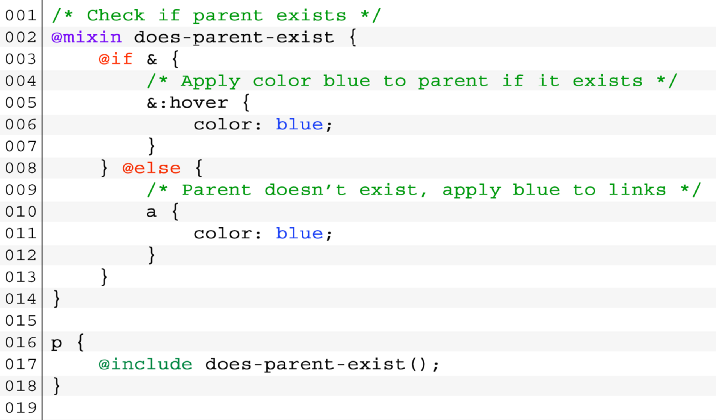


Пример использования одиночного if оператора и комбинации if-else

**Проверяем существует ли родительский элемент**

Символом **&**выбираем родительский элемент, если он существует, в противном случае возвращаем **null**. Можно использовать совместно с директивой **@if**.

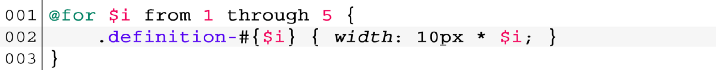
Далее мы создадим CSS стиль с условием, где результат будет зависеть от наличия родительского элемента.



Если родительский элемент не существует, оператор **&** выдаёт **null**и в результате применяется альтернативный стиль.

**@for**

Директиву **@for** используют для повторения CSS определений несколько раз подряд.



Цикл for