






SASS/SCSS ПРЕПРОЦЕССОРЫ - КУРС В ОДНОМ ВИДЕО. NPM, Parcel, Package json. Теория и Практика

| | |
|--|---|
|  YouTube | https://youtu.be/X0YoQz1RV6s |
|  Telegram | https://t.me/Dmitry_Kolotilshikov |
|  Boosty | https://boosty.to/dmitry_ko |



Ссылки на доки:

<https://sass-lang.com/> (англ. версия, более актуальная)

<https://sass-scss.ru/> (рус. версия, для базы отлично)

Конфигурация LiveSASSCompile расширения для VSCode:

<https://github.com/glenn2223/vscode-live-sass-compiler/blob/HEAD/docs/settings.md>

Ссылка на проект:

https://github.com/DmitryKolotilshikov/FE_1.0_SASS/tree/block_1



SASS | SCSS = CSS С СУПЕРСИЛОЙ



SASS (Syntactically Awesome Style Sheets) и SCSS (Sassy CSS) - это языки препроцессоров **CSS**, которые расширяют возможности обычного **CSS** и делают его более мощным и гибким.

На сегодняшний день SASS это уже как мини-язык программирование, который обладает широким набором возможностей.

Основное преимущество использования **SASS** и **SCSS** заключается в следующем:

1. **Переменные:** Вы можете определить переменные и использовать их для хранения значений, таких как цвета, шрифты, отступы и другие. Это позволяет легко изменять значения переменных в одном месте, и они автоматически применятся к соответствующим стилям на всем сайте. Также по сравнению с CSS переменные можно применять в значения для медиазапросов `@media`
2. **Вложенность:** Вы можете вкладывать стили в другие стили, что делает код более организованным и легко читаемым. Например, вы можете определить стили для конкретного элемента внутри другого элемента, без необходимости повторного указания селекторов.
3. **Миксины:** Миксины позволяют определить набор стилей, которые могут быть повторно использованы в разных местах. Это удобно для создания стандартных стилей для кнопок, сеток, анимаций и других компонентов.
4. **Вложенные свойства:** Вы можете определять вложенные свойства для элементов, такие как **hover**, **focus** и другие состояния. Это позволяет легко управлять различными состояниями элементов в одном месте.
5. **Импорт и разделение кода:** **SASS** и **SCSS** позволяют разделять код на отдельные файлы и импортировать их в основной файл. Это удобно для организации и поддержки больших проектов, где можно разделить стили на модули и подключать только необходимые. По умолчанию **Sass** кеширует компилируемые шаблоны и фрагменты. Это значительно ускоряет повторную компиляцию больших коллекций **Sass**-файлов, и работает лучше всего, если шаблоны **Sass** разделены на отдельные файлы, которые импортируются в один файл.
6. **Циклы, условные состояния, математические операции, дебагинг, приватные переменные, шаблонизация, условные операторы, встроенные модули** - это всё SASS

Не используйте директиву `@import`

<https://sass-lang.com/documentation/at-rules/import>

⚠ Heads up!

The Sass team discourages the continued use of the `@import` rule. Sass will gradually phase it out over the next few years, and eventually remove it from the language entirely. Prefer the `@use` rule instead. (Note that only Dart Sass currently supports `@use`. Users of other implementations must use the `@import` rule instead.)

What's Wrong With `@import`?

The `@import` rule has a number of serious issues:

- `@import` makes all variables, mixins, and functions globally accessible. This makes it very difficult for people (or tools) to tell where anything is defined.
- Because everything's global, libraries must add a prefix to all their members to avoid naming collisions.
- `@extend` rules are also global, which makes it difficult to predict which style rules will be extended.
- Each stylesheet is executed and its CSS emitted every time it's `@import`ed, which increases compilation time and produces bloated output.
- There was no way to define private members or placeholder selectors that were inaccessible to downstream stylesheets.

The new module system and the `@use` rule address all these problems.

How Do I Migrate?

We've written a migration tool that automatically converts most `@import`-based code to `@use`-based code in a flash. Just point it at your entrypoints and let it run!



Кратко, почему не использовать `@import`?

`@import` делает все переменные, миксины, функции глобальными. Следовательно т.к всё глобально, нужно добавлять префиксы, чтобы избежать конфликтов. Шаблонизация тоже глобальна, это значит, что сложно предсказать какие стили подтягиваются. Также ко всему увеличенное время компиляции и увеличенный выходного файла. Плюс ко всему отсутствует возможность определять приватные переменные, которые будут доступны в каскаде. Поэтому используем `@use`.