

SASS / SCSS

Sass



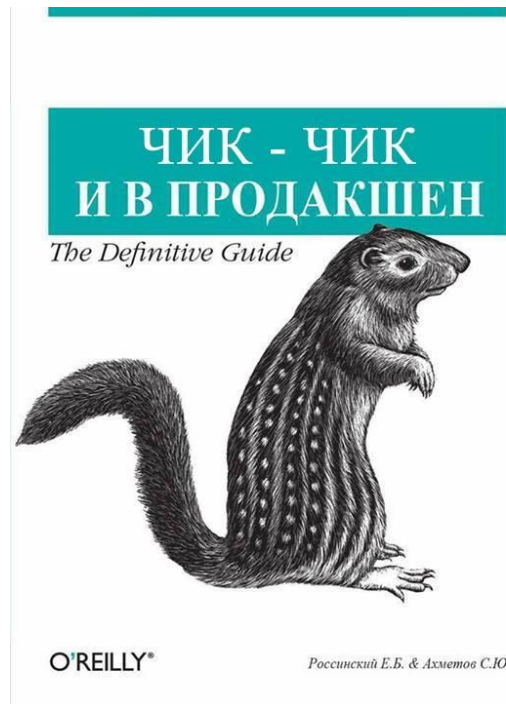
Препроцессоры

Препроцессоры в терминах веба - это программа которая транслирует входящий в нее код (как правило расширяющий возможности и упрощающий написание на стандартных языках) в код другого языка читаемый браузерами.



Ускорение процесса разработки

- вложенности селекторов
- составные селекторы
- арифметические операции
- переменные
- функции
- и много прочих вкусностей...



LoftSchool
ОТ МЫСЛИТЕЛЯ К СОЗДАТЕЛЮ

SASS или SCSS ?

SASS ([Syntactically Awesome Style Sheets](#)) - это язык который при помощи компилятора написанного на Ruby, транслируется в CSS.

SCSS - css-подобный диалект SASS (любой CSS код - это валидный код SCSS)

```
.class  
  color: red  
  border: 1px solid blue
```

sass

```
.class {  
  color: red;  
  border: 1px solid blue;  
}
```

scss



Вложенности селекторов

Если написать дополнительный селектор внутри основных фигурных скобок, то этот селектор становится **зависимым!**

scss

```
.class {  
  color: red;  
  
  .dependent-class {  
    background: blue;  
  }  
}
```

css

```
.class {  
  color: red;  
}  
  
.class .dependent-class {  
  background: blue;  
}
```



Составные селекторы

Оператор - **&** - конкатенирует строку написанную после него с первоначальным селектором.

SCSS

```
.class {  
  font-size: 12px;  
  
  &__inner{  
    color: red;  
  }  
  
  &:hover{  
    font-size: 16px;  
  }  
  
  & + .class {  
    font-size: 20px;  
  }  
}
```

CSS

```
.class {  
  font-size: 12px;  
}  
  
.class__inner {  
  color: red;  
}  
  
.class:hover {  
  font-size: 16px;  
}  
  
.class + .class {  
  font-size: 20px;  
}
```



Арифметические операции

SASS умеет производить операции над числами, но если будут складываться **разные** единицы измерения то будет ошибка.

SCSS

```
.class {  
  padding: 0 20px;  
  width: 400px - (20 * 2);  
}
```

// будет ошибка

```
.error {  
  padding: 20px + 30em;  
}
```

CSS

```
.class {  
  padding: 0 20px;  
  width: 360px;  
}
```



Переменные

Переменные в **SASS** могут хранить что угодно - строки, числа, цвета, списки (массивы) и т.д. Имя переменной может начинаться либо с латиницы либо с подчеркивания.

SCSS

```
$color: #fff;
$width: 200px;
.class {
  color: $color;
  width: $width - 50;
}

$content-text: 'КОНТЕНТ';
.class {
  &:after {
    content: $content-text;
  }
}
```

CSS

```
.class {
  color: #fff;
  width: 150px;
}

.class:after {
  content: "КОНТЕНТ";
}
```



Используем функции

SASS содержит в себе огромное количество встроенных функций (работы с цветами, с массивами, со строками и пр.)

SCSS

```
.class {  
  background: lighten(#000, 10%);  
}
```

CSS

```
.class {  
  background: #1a1a1a;  
}
```



Так же функции можно создавать

Для создания функций есть специальные конструкции `@function` и `@return` для возвращения результата.

SCSS

```
@function funcName($param) {  
  $value : $param;  
  
  @return $value;  
}
```



Повторное использование кода

- примеси
- циклы
- наследование
- “плейсхолдеры”
- условия



Примеси (mixins)

Примеси созданы для генерации свойств в зависимости от входных параметров

SCSS

```
@mixin transition ($value){  
  -webkit-transition: $value;  
  -moz-transition: $value;  
  -ms-transition: $value;  
  -o-transition: $value;  
  transition: $value;  
}  
.class{  
  opacity : .8;  
  @include transition(opacity .3s);  
}
```

CSS

```
.class {  
  opacity: .8;  
  -webkit-transition: opacity 0.3s;  
  -moz-transition: opacity 0.3s;  
  -ms-transition: opacity 0.3s;  
  -o-transition: opacity 0.3s;  
  transition: opacity 0.3s;  
}
```



LoftSchool
ОТ МЫСЛИТЕЛЯ К СОЗДАТЕЛЮ

Бесконечное число параметров

Иногда в свойство CSS можно передавать несколько параметров что бы в примесях это не вызывало ошибок, нужно применить оператор (...)

SCSS

```
@mixin transition ($value...) {  
  -webkit-transition: $value;  
  -moz-transition: $value;  
  -ms-transition: $value;  
  -o-transition: $value;  
  transition: $value;  
}  
  
.class {  
  @include transition(opacity .3s, color .3s);  
}
```



LoftSchool
от мыслителя к создателю

Циклы

Перебирать массивы можно при помощи конструкции `@each in`

SCSS

```
$drinks: beer, rum, absinthe;

@each $drink in $drinks {
  .my__lovely-#{ $drink } {
    background: url('/img/for-#{ $drink }');
  }
};
```

CSS

```
.my__lovely-beer {
  background: url('/img/for-beer');
}

.my__lovely-rum {
  background: url('/img/for-rum');
}

.my__lovely-absinthe {
  background: url('/img/for-absinthe');
}
```



LoftSchool
ОТ МЫСЛИТЕЛЯ К СОЗДАТЕЛЮ

Условия

Конечно же есть `@if`, `@else` и `@else if`

SCSS

```
$bg: backgrounded, colored;

@each $type in $bg {
  .colored-block__#{ $type } {
    @if $type == colored {
      background: red;
    } @else {
      background: url('/img/pic.jpg');
    }
  }
};
```

CSS

```
.colored-block__backgrounded {
  background: url("/img/pic.jpg");
}

.colored-block__colored {
  background: red;
}
```



Наследование

Если нам нужно расширить или унаследовать свойства класса, мы можем применить директиву **@extend**

SCSS

```
.class {  
  color: red;  
  width: 200px;  
  height: 300px;  
}  
  
.same-class {  
  @extend .class;  
  margin-left: 200px;  
}
```

CSS

```
.class, .same-class {  
  color: red;  
  width: 200px;  
  height: 300px;  
}  
  
.same-class {  
  margin-left: 200px;  
}
```



Плейшолдеры

Что бы не дублировать функциональные классы, можно использовать специальную конструкцию

SCSS

```
%inlineblock {  
  display: inline-block;  
  vertical-align: top;  
}  
  
.one {  
  @extend %inlineblock;  
}  
  
.two {  
  @extend %inlineblock;  
}
```

SCSS

```
.one, .two {  
  display: inline-block;  
  vertical-align: top;  
}
```



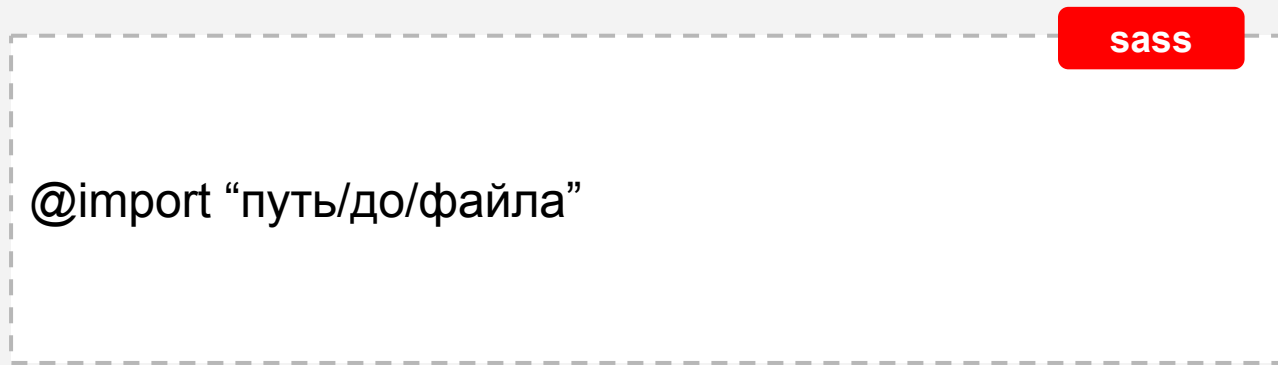
LoftSchool
ОТ МЫСЛИТЕЛЯ К СОЗДАТЕЛЮ

Импортирование стилей

Импорт стилей

Одним из факторов влияющих на скорость загрузки страницы - это количество файлов подключенных в неё. Поэтому в документе должен быть подключен только один файл со стилями. Такие файлы называют **входной точкой (entry file)**.

Но, что бы файл стилей не разрастался до огомных размеров, и весь код не сливался в одну “кучу” принято **разбивать CSS** на логические части. Это помогает реализовать конструкция **@import**.



«Лучше один раз увидеть,
чем сто раз услышать»

- Людвиг ван Бетховен

