**ЭКСПРЕСС-КУРСЫ ВЕРСТАЛЬЩИКОВ**

**ТЕМА 3:** Препроцессор CSS “SCSS”

**План**

1. Введение;
2. SCSS;
    a. Вложенные правила;
    b. Вложенные медиа запросы;
    c. Переменные;
    d. Комментарии;
    e. Миксины;
    f. Наследование;
    g. Математические операции;
    h. Функции
    i. Встроенные функции;
    j. Импорт;
    k. Условия
    l. Циклы

## 1. Введение;

Использование препроцессоров для CSS во время разработки шаблоны не является обязательным условием. Тем не менее, они позволяют сократить время разработки шаблона на 20-30%.

В наших шаблонах преимущественно используется препроцессор для CSS “SCSS”. Данный препроцессор имеет широкий спектр возможностей, с которым можно ознакомиться на официальном сайте.

В рамках данной теме будут рассмотрены только найболее часто используемые из них.

Для успешного прохождения данной темы, рекомендуется установить приложение Koala App, с помощью которого код препроцессора будет компилироваться в CSS.

## 2. SCSS

SCSS — это динамический язык стилей, который является диалектом языка SASS.

### Вложенность правил

Препроцессор SCSS позволяет вкладывать правила друг в друга. Например,

```
main {
  background: #fff;

  h3 {
    color: #333;
  }

}
```

После компиляции получим следующий код:

```
main {
  background: #fff;
}

main h3 {
  color: #333;
}
```

Используя вложенность селекторов можно также обращаться к родительскому селектору. Например,

```
.brand {
  text-align: center;

  &_name{
    overflow: hidden;
  }

}
```

После компиляции получим:

```
.brand {
  text-align: center;
}

.brand_name{
  overflow: hidden;
}
```

**Вложенность медиа запросов**

По мимо вложенности правил SCSS также поддерживает вложенность медиа запросов. Например,

```
header {
  background: #fff;

  .brand {
     float: left;
   }
```

```
  @media (max-width: 767px) {
    text-align: center;

    .brand {
      float: none;
    }

  }

}
```

На выходе получим:

```
header {
  background: #fff;
}

header .brand {
  float: left;
}

@media (max-width: 767px) {
  header {
    text-align: center;
  }
  header .brand {
    float: none;
  }
}
```

**Переменные**

Препроцессор SCSS позволяет объявлять и использовать переменные в коде. Например,

```
$blue: #5B83AD;

#header {
  color: $blue;
}
```

Данный пример, при компиляции, следующий CSS код:

```
#header {
  color: #5B83AD;
}
```

В качестве значения для переменной можно использовать любые другие значения для CSS свойств или другие переменные

```
$OpenSans: ‘Open Sans’, sans-serif;
$default-font: 400 14px/24px $OpenSans;

body {
  font: $default-font;
}
```

При компиляции, мы получим следующий CSS код:

```
body {
  font: 400 14px/24px ‘Open Sans’, sans-serif;
}
```

**Комментарии**

SCSS, по мимо обычных CSS комментариев вида /* */, позволяет использовать //.

Данный тип комментариев будет игнорируется компилятором и в выходном CSS файле не отображается.

Например,

```
// My Awesome Button
.btn {
  color: #4d8a0f;
}
```

Дает нам следующий CSS код:

```
.btn {
  color: #4d8a0f;
}
```

**Миксины**

Миксином в SCSS является набор CSS правил, либо других миксинов, объединенных в одну именованую группу, которую можно использовать во многих участках кода.

Миксины в SCSS определяются с помощью директивы @mixin,а в стилях подключаются с помощью директивы @include.

Рассмотрим пример использования миксина:

```
@mixin clearfix {

  &:before,
  &:after {
    display: table;
    content: "";
    line-height: 0;
  }

  &:after {
```

```
    clear: both;
  }

}
```

Данный миксин добавляет для контейнера свойства приема “clearfix”, описанного в предыдущей лекции. После объявления данного миксина его можно использовать в дальнейшем в коде.

```
header {
  @include clearfix();

  color: #fff;
  background: #333;

  .brand {
    float: left;
  }

}
```

После компиляции, получим следующий CSS код:

```
header {
  color: #fff;
  background: #333;
}

header:before, header:after {
  display: table;
  content: "";
  line-height: 0;
}

header:after {
  clear: both;
}

header .brand {
  float: left;
}
```

Миксины,  также могут быть параметризированными.

```
@mixin transform($value) {
  -ms-transform: $value;
  -webkit-transform: $value;
  transform: $value;
}
```

```
.caption {
  position: absolute;
  top: 0;

  background: #000;
  color: #FFF;

  @include transform(translateY(-50%));
}
```

После компиляции получим следующий CSS код:

```
.caption {
  position: absolute;
  top: 0;
  color: #FFF;
  -ms-transform: translateY(-50%);
  -webkit-transform: translateY(-50%);
  transform: translateY(-50%);
}
```

**Наследование**

В SCSS, помимо миксинов, также доступна возможность наследования. Принцип наследования в SCSS определяется с помощью деректива @extend.

```
.fa-twitter {
  display: inline-block;
  width: 60px;
  height: 60px;
  line-height: 60px;
  font-size: 30px;
  background: #000;
  color: #FFF;
}

.fa-facebook {
  @extends .fa-twitter;
  font-size: 20px;
}
```

После компиляции получим следующее:

```
.fa-twitter, .fa-facebook {
  display: inline-block;
  width: 60px;
  height: 60px;
  line-height: 60px;
  font-size: 30px;
  background: #000;
  color: #FFF;
}
```

```
.fa-facebook {
  font-size: 20px;
}
```

Также в SCSS есть возможность наследования так званных плейсхолдеров, которые определяются с помощью преффикса %. Плейсхолдеры, сами по себе, игнорируются и в итоговый CSS код не попадают. Например,

```
%icon {
  display: inline-block;
  width: 60px;
  height: 60px;
  line-height: 60px;
  font-size: 30px;
  background: #000;
  color: #FFF;
}

.fa-facebook {
  @extend %icon;
  font-size: 20px;
}
```

После компиляции получим,

```
.fa-facebook {
  display: inline-block;
  width: 60px;
  height: 60px;
  line-height: 60px;
  font-size: 30px;
  background: #000;
  color: #FFF;
}

.fa-facebook {
  font-size: 20px;
}
```

**Математические операции**

SCSS позволяет использовать математические операции в коде. Например,

```
.btn{
  $height: 50px;
  $line-height: 20px;

  padding: ($height - $line-height)/2 10px;
}
```

На выходе получим следующий код:

```
.btn{
  padding: 15px 10px;
}
```

**Функции**

В SCSS, подобно многим языкам программирования, есть возможность писать функции:

```
@function increment($value) {
  @return $value + 1;
}

#header {
  $padding: increment(19)*1px;
  padding-top: $padding;
}
```

Результат,

```
#header {
  padding-top: 20px;
}
```

**Встроенные функции**

SCSS поддерживает большое количество встроенных функций различного назначения. С полным списком можно ознакомиться тут:
http://sass-lang.com/documentation/Sass/Script/Functions.html.
Мы рассмотрим только некоторые из них:

**ceil()** - округляет дробное число к большему целому. Например,

```
.btn {
  padding: ceil(3px / 2);
}
```

Получим,

```
.btn {
  padding: 2px;
}
```

**floor()** - округляет дробное число к меньшему целому.

```
.btn {
  padding: floor(3px / 2);
}
```

Получим,

```
.btn {
  padding: 1px;
}
```

**lighten()** - делаем искомый цвет светлее указанного на определенный процент.

```
.btn {
  color: lighten(#80e619, 20%);
}
```

Получим,

```
.btn {
  color: #b3f075;
}
```

**darken()** - делаем искомый цвет темнее указанного на определенный процент.

```
.btn {
  color: darken(#80e619, 20%);
}
```

Получим,

```
.btn {
  color: #4d8a0f;
}
```

**Импорт**

SCSS позволяет включать другие *.scss файлы в ваш основной файл стилей на этапе компиляции.

Это очень удобно использовать для создания модульной структуры стилей.

Например,

// buttons.scss

```
.btn {
  color: #4d8a0f;
}
```

// styles.scss

```
.body {
  color: #000;
}

@import '../PATH/TO/buttons.scss';
```

После компиляции получим:

// styles.css

```
.body {
  color: #000;
```

```
}

.btn {
  color: #4d8a0f;
}
```

**Условия**

SCSS имеет специальные деректривы @if, @else для определения условий. Например,

```
$height: 20px;

.box {
  @if $height > 20px {
    height: 100%;
  } @else {
    height: 20px;
  }
}
```

На выходе получим,

```
.box {
  height: 20px;
}
```

**Циклы**

SCSS также имеет специальные деректривы @for, @each для реализации циклов.

Например, цикл @for

```
@for $i from 1 through 3 {
  .item-#{$i} {
     width: 20px * $i;
   }
}
```

Данный код даст на выходе:

```
.item-1 {
    width: 20px;
}

.item-2 {
    width: 40px;
}

.item-3 {
    width: 60px;
```

```
}
```

Пример для цикла @each

```
@each $animal in puma, sea-slug, egret, salamander {
  .#{$animal}-icon {
     background-image: url('/images/#{$animal}.png');
  }
}
```

После компиляции получим следующий код:

```
.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');
}
```