Лабораторная работа № 11
ПРЕПРОЦЕССОР SCSS

Цель работы: изучить принципы метаязыка Sass, научиться использовать функциональные возможности препроцессора SCSS.

Теоретические сведения для выполнения работы

Метаязык Sass

Sass (Syntactically Awesome Stylesheets) — препроцессор CSS, который представляет собой метаязык на основе CSS, предназначенный для расширения возможностей и сокращения записи CSS. Препроцессор — программа, имеющий собственный синтаксис, который затем компилируется в стандартный код другой программы. Кроме Sass существуют такие препроцессоры, как Stylus, PostCSS, SCSS, Less.

Для того чтобы использовать Sass необходимо установить программное обеспечение среды разработки языка программирования Ruby, а также основные файлы Sass. Загрузить установочный пакет Ruby можно с веб-сайта rubyinstaller.org/downloads. В редакторе Visual Studio Code для использования Sass необходимо установить расширение Live Sass Compiler и изменить параметры согласно рис. 11.1, предварительно выбрав Ctrl+Shhift+P>Open Workspace Setting(JSON)

Рис. 11.1 Настройки параметров Live Sass Compiler

В savePath указывается путь сохранения файла с расширением .css. После создания файл .scss следует нажать кнопку *Watch Sass* в нижней панели.

При использовании Sass создается файл с расширением .scss, который пользователь редактирует, и файл, который сгенерирует препроцессор Sass, т. е. файлы с расширением .css. Таким образом, наиболее распространенный способ организовать их расположение в корневом каталоге веб-сайта заключается в создании папки css, которая используется для хранения CSS-файлов, скомпилированных с помощью Sass, и другая папка scss — для файлов с расширением .scss.

Синтаксис SCSS

В синтаксисе SCSS используются переменные, вложенные правила, миксины, встроенный импорт, функции и операторы, директивы.

Создание переменной осуществляется по следующему синстаксису: **$имя переменной: значение;**. Имя переменной начинается с символа доллара и может содержать латинские символы, цифры, дефис и подчеркивание. Регистр букв имеет значение. После двоеточия указывается значение переменной, а в конце ставится точка с запятой. Пример создания переменной и их использование представлено на рис. 11.2

```scss
$white: #ffffff;
$black: #000000;
body {
    color: $white;
    background-color: $black;
}
```

а

```css
body {
  color: #ffffff;
  background-color: #000000;
}
```

б

а — создание переменных, б — результат компиляции

Рис. 11.2 Пример использования переменных в SCSS

Переменные могут быть глобальными, если создана вне блоков объявлений, и локальными, если будет создана внутри блока объявления. Переменную можно подставить в комментарий, строку, селектор, используя вид: **#{$имя переменной}**.

В большинстве при написании CSS селекторы часто дублируются. Для избежания дублирования используются вложенные правила или атрибут амперсанда **&**. Примеры представлены на рис. 11.3

```scss
$red: #f50b0b;
$blue: #630dec;
div {
    background: $red;
    p {
    color: $blue;
    }
}
```

а

```css
div {
  background: #f50b0b;
}
div p {
  color: #630dec;
}
```

б

```scss
$red: #f50b0b;
$blue: #630dec;
div {
    background: $red;
    &:hover {
    background: $blue;
    }
}
```

в

```css
div {
  background: #f50b0b;
}
div:hover {
  background: #630dec;
}
```

г

а, б — вложенность в scss; в,г — результат в css
11.3 Пример использования вложенных правил SCSS

Для создания определений стилей, которые можно повторно использовать используются шаблоны и миксины (примеси). Миксины в отличие от шаблонов могут содержать параметры, что позволяет настраивать значения стиля или генерировать. Вместо селектора можно указать **%название**. Такой вид называется шаблонным селектором или селектором-заполнителем. Чтобы вставить шаблон, нужно воспользоваться директивой **@extend**. Шаблонный селектор можно указать в составе обычных селекторов. Миксины позволяют вставить одно или несколько правил стиля вместо директивы **@include** с названием миксина сколько угодно раз. Для создания миксина используется директива **@mixin**. После директивы указывается название миксина и внутри круглых скобок могут указываться параметры, которые являются локальными переменными. Если миксин не содержит параметров,

то можно добавить пустые круглые скобки или вообще их не добавлять. На рис. 11.4 представлен пример использования шаблонов и миксинов.

```
%red-color {
    color: red;
}
p {
    @extend %red-color;
}
h1 {
    @extend %red-color;
}
```

а

```
h1, p {
  color: red;
}
```

б

```
@mixin flex {
    display:flex;
    justify-content: space-between;
}
.header {
    @include flex;
}
.container {
    @include flex;
}
```

в

```
.header {
  display: flex;
  justify-content: space-between;
}

.container {
  display: flex;
  justify-content: space-between;
}
```

г

а, в – шаблоны и миксины в Sass, б,г – результат в css
11.4 Пример испоользования шаблонов и миксинов

В Sass можно создавать фрагменты кода, которые можно вызывать неоднократно из любого места программы. Эти фрагменты кода называются функциями. Пример функции и ее вызов представлен на рис. 11.5

```
@function max-width() {
    @return 1000px;
}
$value: max-width();
div { width: 500px / max-width() * 100%; }
```

Рис. 11.5 Пример использования функции

Определение пользовательской функции начинается с директивы **@function**, после которой указывается имя функции с

круглыми скобками. В круглых скобках могут быть параметры функции, которые являются локальными переменными. Функция всегда что-то возвращает и возвращаемое значение указывается после оператора **@return**. Как видно из примера, вызывать функцию можно сколько угодно раз, что позволяет использовать один и тот же код многократно.

В функции можно применить логические, арифметические, условные операторы и цикла.

Оператор условия **@if** позволяет в зависимости от значения логического выражения выполнить блок программы. Для цикла используется оператор **@for**, чтобы выполнить блока программы определенное число раз. Пример использования операторов цикла и условий представлена на рис. 11.6

а

```
/* использование @if в Sass*/
@mixin p-color($height,$width) {
    @if (($height<$width) and ($width >= 30px))
     { color: red; }
    @else { color: green; }
}
.class {@include p-color (10px,30px)}
```

б

```
/* результат в CSS*/
.class {
  color: red;
}
```

в

```
@for $i from 1 through 3 {
    .class-#{$i} {
        width: 10px*$i;
    }
}
```

г

```
/* результат компиляции в CSS*/
.class-1 {width: 10px;}
.class-2 {width: 20px;}
.class-3 {width: 30px;}
```

д

```
$index: 5;
@while $index > 0 {
    .class-#{$index} {
        width: 10px * $index;
    }
}
```

е

```
/* результат компиляции в CSS*/
.class-5 {width: 50px;}
.class-4 {width: 40px;}
.class-3 {width: 30px;}
.class-2 {width: 20px;}
.class-1 {width: 10px;}
```

а, в, д — циклы и условия в Sass; б, г, е — результат в CSS

Рис. 11.6 Пример использования операторов цикла и условия

Производить арифметические операции позволяют операторы сложения (+), вычитания (–), унарный минус, умножение (*), деления (/), остатка от деления (%). Чтобы выполнить деление необходимо либо заключить выражение в круглые скобки, либо сохранить результат в переменной, либо возвратить результат из

функции. Приоритет выполнения операции умножения выше, затем деление и сложение. Изменить порядок вычисления выражения можно с помощью круглых скобок. Операции заключенные в них имеют наивысший приоритет.

К операторам сравнения относятся равно (==), не равно (!=), меньше (<), больше (>), меньше или равно (<=), больше или равно (=>). Два значения равны, если имеют одинаковый тип данных и значение. Числа равны, если равны их значения и единицы измерения.

Директива @import

Sass-правила можно разделить на несколько файлов, а затем объединить в один файл. В Sass эти файлы называются фрагментами. Чтобы сообщить препроцессору, что необходимо преобразовать эти фрагменты в отдельные CSS-файлы, их имена должны начинаться с символа нижнего подчеркивания. Чтобы скомпилировать определенный фрагмент используется директива **@import**.

Наиболее распространенный способ заключается в том, чтобы подготовить один Sass-файл, который будет преобразован в законченный CSS-файл и который не содержит ничего, кроме нескольких директив импорта. Содержимое Sass-файла может выглядеть следующим образом.

```
@import '_value.scss';
@import '_flex.scss';
@import '_grid.scss';
```

Рис. 11.6 Пример использования @import

При импорте фрагментов в другой Sass-файл нижнего подчеркивания (_) и расширение .scss можно опустить.

Задания к лабораторной работе № 11

Задание 1. Создайте веб-страницу согласно прототипу, представленному на рис. 11.6. Для файлов с расширениями css и

scss должны быть соответствующие им отдельные папки. Переменные должны хранится в отдельном файле.

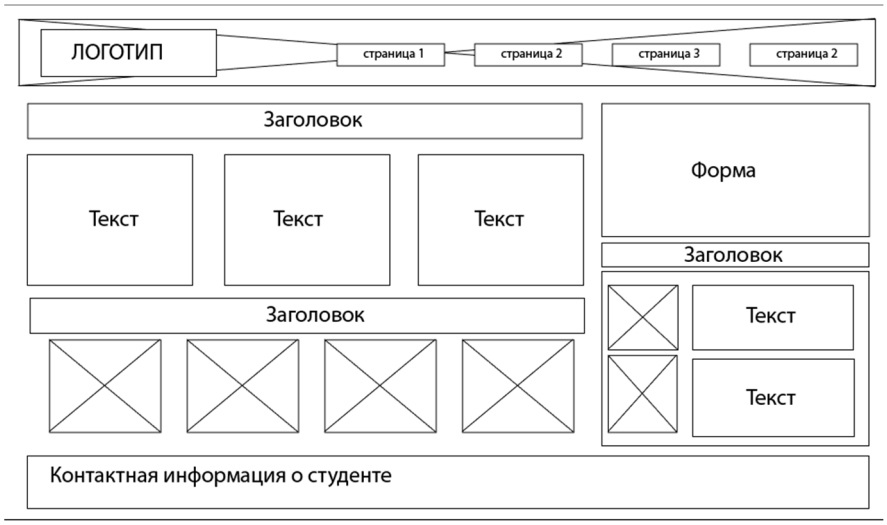

Рис. 11.7 Прототип веб-страницы к заданию 1

Задание 2. Для веб-страницы использовать вложенные правила при создании навигационного меню. При оформлении формы использовать шаблонные селекторы и миксины. Также их можно применить и для других элементов.

Задание 3. Создать функции, которые устанавливают зависимость между размером шрифта заголовка и текста после этого заголовка, а также функции с наличием условных и логических операторов.

Контрольные вопросы

1. Что такое Sass? Как расшифровывается Sass?
2. Как установить Sass?
3. Что такое препроцессор SCSS?
4. Как создаются переменные в SCSS?
5. На какие виды подразделяются переменные? В чем между ними разница?
6. Для чего используется @import в препроцессоре SCSS?
7. Что такое шаблонные селекторы?
8. Что такое миксины? Чем они отличаются от шаблонных селекторов?
9. Для чего предназначено @mixin?

10. Каким образом создать функции в Sass?

11. Какие операторы в Sass вы знаете?

12. Создайте файл SCSS с условием, которое заключается в том, что для текста размером 20px цвет будет красный.

13. Какие операторы относятся к арифметическим?

14. Какие операторы относятся к операторам сравнения?

15. Назовите директивы цикла и условия

16. Создайте файл SCSS с переменными и вложенными правилами и примените стили к элементам на странице