

SASS (SCSS)

1. Wprowadzenie

1.1. Na temat SASS (SCSS)

Nazwa pochodzi od wyrażenia Syntactically Awesome Stylesheets

1.2. CSS vs SASS vs SCSS

Kod Sass to nic innego jak kod CSS wzbogacony o pewne elementy. Na potrzeby tej lekcji będziemy posługiwali się plikiem o rozszerzeniu `*.scss`. SCSS jest bliższy składnią do CSS niż SASS (mimo że to SASS był pierwszy), zatem przejście z CSS na SCSS nie powinno być problemem.

Główne różnice:

- SASS używa wcięć zamiast nawiasów,
- SASS używa nowej linii zamiast średnika,

1.3. Konwersja SASS (SCSS) do CSS

Kod Sass musi być skompilowany (zamieniony) do formatu CSS przez specjalny program, zanim trafi on do przeglądarki. Można skorzystać:

- ze specjalnych pluginów do IDE, które rozszerzają możliwości takich programów jak VisualStudio Code,
- przy pomocy webpacka lub gulp'a zamieniać kod na CSS

2. Podstawy

2.1. Zagnieżdżanie

Zagnieżdżanie właściwości CSS

Kod HTML:

```
<div class="wrapper">
  Lorem ipsum
  <h1> Title </h1>
  <h2> Subtitle </h2>
  <a href="#"> Link </a>
</div>
```

Kod SCSS:

```
.wrapper { max-width: 250px; background-color: lightblue;
  h1 {color: white; }
  h2 {color: blue;}
  a {text-decoration: none;
    &:hover{color:brown;}
  }
}
```

Selektor rodzica

Kolejny przykład:

```
<div class="sampleClass">
  Lorem ipsum
  <div class="sampleClass__block">Hello World
</div>
</div>
```

Kod SCSS:

```
.sampleClass{
  &__block{color: yellow}
}
```

Przykład:

```
.box { padding: 20px; }
.box-header { font-size: 30px; }
.box-header-small { font-size: 25px;}
.box-body { color: white; }
.box-footer { font-size: 20px; }
```

Zagnieżdżanie MediaQueries

```
.box {  
width: 10vw;  
@media screen and (max-width: 768px) { width: 80vw;  
} }
```

Zagnieżdżanie fontów

```
@import url('https://fonts.googleapis.com/css?family=Roboto+Condensed'  
);  
  
$font-family: "Roboto Condensed";  
$font-size: 20px;  
  
.wybor_fontu{  
font-family: $font-family;  
font-size: $font-size;  
}
```

2.2. Zmienne

W przypadku wystąpienia tych samych wartości w różnych miejscach w CSS:

```
h1 {color: red;}  
button {border: 1px solid red; color: white; }
```

W SCSS zmienne definiujemy:

```
$nazwa_zmiennej: wartość_zmiennej;
```

Powyższy kod wyglądałby:

```
$color: red  
h1 {color: $color;}  
button {border: 1px solid $color; color: white; }
```

Co możemy przechowywać w zmiennej?

Najważniejsze rzeczy to:

- wartości liczbowe (32px, 16vh, 100%, 1rem, 1em)
- serię wartości (20px 10px 20px 10px) - przydatne jako parametry dla margin, padding itp.
- kolory (#000000, red, rgba(0,0,0,0.4))
- wartości tekstowe

- odwołania do innych zmiennych *

Nazwy zmiennych

Nie można w nich stosować znaków specjalnych (oprócz znaku `-` oraz `_`).

Zamiast stosować `$color: red` albo `$red: red` lepiej zastosować:

```
$color-main: lightblue;
$color-header: #000000;
$color-footer: #ccc;
$color-link: #ffffff;
$color-button: $color-main;
```

Zasięg zmiennych

Zmienne w SASS mogą być lokalne, jak i globalne - zupełnie jak w innych językach programowania.

Przykład zmiennej lokalnej:

```
.wrapper {
  $width: 60vh;
  width: $width;
}
```

Zmienna lokalna jest dostępna jedynie w obrębie danej reguły (ale pamiętaj, że reguły można w sobie zagnieźdzać, więc dostęp do niej będzie wtedy możliwy).

Aby zmienna lokalna stała się zmienną globalną musisz użyć flagi `!global`

```
.wrapper {
  $width: 60vh !global;
  width: $width;
}
```

Wartości domyślne

```
$font-size: 50px !default;
```

2.3. Komentowanie kodu

Rodzaje komentarzy:

- komentarze wielowierszowe — `/* komentarz */`;
- komentarze jednowierszowe, tzw. ciche (silent) — `// komentarz`;

- komentarze głośne (loud), wielowierszowe — `/*! komentarz */`.

3. Techniki zaawansowane

3.1 Dyrektywa @import

Importować pliki można w obrębie całego kodu. Należy jednak pamiętać, że kluczowa jest kolejność importowania - zatem należy je zaimportować odpowiednio wcześniej, aby móc z nich skorzystać.

Import z innego pliku:

```
@import "folder/file"
```

Import taki można przeprowadzić globalnie, jak i lokalnie.

3.2. Domieszki (mixin)

Wyjaśnienie

Domieszka pozwala na zdefiniowanie określonego fragmentu kodu, a następnie użycie go w innym miejscu (regule CSS) - zupełnie tak, jak ma to miejsce z funkcjami w JS.

Zalety stosowania mixinów:

- pozwala uniknąć przepisywania powtarzających się fragmentów kodu - zgodnym z regułą DRY (Don't Repeat Yourself — nie powtarzaj się).
- Ponadto w razie zmian jakiejś wartości lub właściwości CSS, zmiany te będą automatycznie zastosowane we wszystkich miejscach, bez konieczności ręcznej edycji kodu.

Domieszki - struktura

```
<a href="www.google.pl"> Link </a>  
<p> Lorem ipsum </p>
```

Kod SCSS:

```

@mixin zmiana(){
  transition: all 1.2s ease-in-out;
}
$color: red;
$color-hover: blue;
a{
  color: $color;
  &:hover{
    @include zmiana();
    color: $color-hover;
  }
}

p{
  color: $color;
  &:hover{
    @include zmiana();
    color: $color-hover;
  }
}

```

Przykład: <https://codepen.io/MarcinDI/pen/QJxyda>

Przekazywanie argumentów

Kod HTML

```

<div class="wrapper">
  Sample text!
</div>

<div class="wrapper2">
  Sample text 2!
</div>

```

Kod SCSS:

```

////////// CSS //////////
// .wrapper{
//   width: 250px;
//   height: 100px;
//   background-color: blue;
//   color: red;
// }
////////// SCSS //////////
$width: 250px;
$height: 100px;
$background-color: blue;

@mixin size($width,$height,$background-color)
{
  width: $width;
  height: $height;
  background-color: $background-color;
  color: red;
}

.wrapper{
  @include size($width,$height,$background-color);
}

.wrapper2{
  @include size($width*2,$height*0.5,green);
}

```

Przykład: <https://codepen.io/MarcinDI/pen/PxaPRw>

Wartości domyślne

```
@mixin nazwa-domieszki($argument1: wartość1, $argument2:wartość2)
```

Dyrektywa @content

Dyrektywa @content pozwala przekazać blok zawartości do mixinu.

Przykład

```

@mixin center-block {
  display: flex;
  align-items: center;
  justify-content: center;
}

.button {
  @include center-block;
}

```

Kolejny przykład:

```
$bp-small: 34.375em; // 550px;
$bp-medium: 47.5em; // 760px;
$bp-desktop: 64em; // 1024px;
$bp-wide: 105em; // 1680px;

@mixin res($breakpoint) {
  @if $breakpoint == wide {
    @media only screen and (min-width: $bp-wide) {
      @content;
    }
  }
  @if $breakpoint == desktop {
    @media only screen and (min-width: $bp-desktop) {
      @content;
    }
  }
  @if $breakpoint == tab {
    @media only screen and (min-width: $bp-medium) {
      @content;
    }
  }
  @if $breakpoint == phone {
    @media only screen and (min-width: $bp-small) {
      @content;
    }
  }
}

body {
  @include res(tab) {
    background-color: red;
  }
}
```

Przykład: <https://codepen.io/MarcinDI/pen/jOPrKQx>

Garść porad oraz kiedy stosować

- gdy określone deklaracje CSS się powtarzają,
- nie twórz zbyt szczegółowych zestawów właściwości, które będą mogły być rzadko wykorzystane,
- mixiny nie zawsze muszą mieć argumenty,
- stosuj adekwatne nazewnictwo, tak aby łatwo rozpoznać przeznaczenie domieszki,
- domieszki najlepiej umieścić w osobnym pliku częściowym

3.3. Dziedziczenie (dyrektywa @extend)

Wyjaśnienie

Kod HTML:

```
.message { border-width: 1px; border-style: solid;
border-color: grey; color: grey; padding: 1em;
}
.message-success { border-width: 1px; border-style:
solid; border-color: green; color: green; padding:
1em;
}
.message-error { border-width: 1px; border-style:
solid; border-color: red; color: red;
padding: 1em; }
```

Po refaktoryzacji:

```
.message, .message-success, .message-error { border:
1px solid grey;
color: grey;
padding: 1em;
}

.message-success { border-color: green; color:
green;
}

.message-error { border-color: red; color: red;
}
```

Z wykorzystaniem dyrektywy @extend

```
.message {
border: 1px solid grey; color: grey;
padding: 1em;
}
.message-success {
@extend .message;
border-color: green;
color: green; }
.message-error {
@extend .message;
border-color: red;
color: red; }
```

Kiedy stosować

Tak więc rozszerzanie reguł — w odróżnieniu od domieszek — nie powiela fragmentów tego samego kodu, rozszerza jedynie istniejące reguły o dodatkowe selektory. Dzięki czemu kod jest krótszy.

Kiedy stosować?

- kiedy istnieje wyraźna relacja pomiędzy elementami i ich stylami (np. reguła `.error`, definiująca wspólny kolor komunikatów błędów dla różnych elementów na stronie),
- unikaj wielokrotnego i łańcuchowego wywoływania dyrektywy `@extend`,
- staraj się używać rozszerzeń w stosunku do selektorów występujących w arkuszach tylko raz (wtedy łatwo będziesz w stanie przewidzieć rezultat rozszerzenia),
- nie wykorzystuj rozszerzania z poziomu zapytań medialnych (celem uniknięcia błędów kompilacji).

4. Dobre praktyki

4.1. Struktura projektów

Struktura projektu

Przykład 1.

```
Project-Name/  
├── ...  
├── css/  
│   └── main.css  
└── scss/  
    ├── partials/  
    │   ├── _variables.scss  
    │   ├── _base.scss  
    │   ├── _helpers.scss  
    │   ├── _layout.scss  
    │   ├── _typography.scss  
    │   ├── _modules.scss  
    │   └── _mixins.scss  
    └── main.scss
```

Przykład 2.

```

└─ scss/
  └─ base
    ├── _base.scss
    ├── _form.scss
    ├── _layout.scss
    ├── _normalize.scss
    └── _typography.scss
  └─ components
    ├── _buttons.scss
    └── _header.scss
  └─ utils
    ├── _functions.scss
    ├── _helpers.scss
    ├── _mixins.scss
    └── _variables.scss
  └─ vendor
    └── _bootstrap.scss
  └─ main.scss

```

Przykład 3.

```

└─ scss/
  └─ base
    ├── _base.scss
    ├── _normalize.scss
    └── _typography.scss
  └─ components
    └─ buttons
      ├── _mixins.scss
      └── _variables.scss
    └─ panels
      ├── _mixins.scss
      └── _variables.scss
    ├── _buttons.scss
    └── _panels.scss
  └─ layout
    └─ grid
      ├── _mixins.scss
      └── _variables.scss
    └── _grid.scss
  {...}
  └─ main.scss

```

4.2 Przydatne techniki

- Użycie zmiennej ma sens wtedy, kiedy wiemy, że określona wartość ma istotne znaczenie, będzie wykorzystywana wielokrotnie lub jej wartość będzie często zmieniana
- spójna konwencja nazewnictwa,
- stosując zagnieżdżenia, staraj się unikać odwzorowywania struktury kodu HTML
(`body #page .main .box div.body > div a`)
- stosowanie maksymalnie 3 – 4 poziomów zagnieżdżeń,
- W zdecydowanej większości przypadków użycie domieszek jest sensowniejsze. W przeciwieństwie do rozszerzeń — trudniej o nieprzewidziane rezultaty, a sam kod Sass jest również łatwiejszy do zrozumienia,
- Używaj domieszek, jeśli istnieje potrzeba przekazania argumentów, a gdy chcesz wykorzystać wielokrotnie powtarzające się fragmenty kodu, dobrym rozwiązaniem będą selektory zastępcze,
- staraj się, aby zawsze istniał w Twoim projekcie jeden główny plik Sass, który będzie służył wyłącznie do importowania innych plików

5. Przykłady

5.1. Zerowanie stylów

<https://github.com/appleboy/normalize.scss>

5.2. Centrowanie elementów

```
@mixin centerer {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translate(-50%, -50%);  
}
```

5.3. Wygląd linków

```

@mixin link-style ($link, $visited:darken($link,10),
$hover: lighten($link,10), $active: $hover) {
  a{
    color: $link;
    &:visited {color: $visited;}
    &:hover {color: $hover; }
    &:active {color: $active;}
  }
}

@include link-style(blue, $hover: lightblue);

```

5.4. Media Queries

```

$small: 'screen and (max-width: 320px)';
$medium: 'screen and (min-width: 321px) and (max-
width: 768px)'; $large: 'screen and (min-width:
769px)';

.box {
  @media #{$medium} {
    width: 80%; }
}

```

A z użyciem domieszek:

```

@mixin screen($size) {
  @if $size == large { @media
    #{$large} { @content; } }

  @else if $size == medium { @media #{$medium}
    { @content; } }

  @else if $size == small { @media #{$small} {
    @content; } }

  @else { @media #{$size} { @content;} }
}

.box {
  @include screen(medium) {
    width: 80%; }
}

```

5.5 Podsumowanie

- <https://codepen.io/collection/DVbgOP>