# Wprowadzenie do Sass

Najpopularniejszego preprocesora CSS

# Metajęzyk?

...preprocesor, rozszerzenie CSS. Saas nie jest nowym językiem, nie jest to alternatywa dla CSS, bo i tak CSS trzeba znać bardzo dobrze.

Choć jest w tym pewne (spore) uproszczenie, to jest jak jQuery dla JavaScript.

Ułatwia, pozwala zrobić więcej, ale wymaga też praktyki.

**Sass = Syntactically Awesome Stylesheets.**
**(Syntaktyczny(Składniowy)/rozszerzający/ulepszony CSS)**

# Kiedy się używa

## Duże projekty
## W Software House'ach

Najlepiej uczyć się robiąc w nim projekty (nawet małe). Na rozmowie kwalifikacyjnej czasami trzeba też wytłumaczyć o co chodzi z zagnieżdżaniem, zmiennymi (różnica między zmiennymi w preprocesorach i css), co to jest mixin (domieszka) i jak używać, na czym polega importowanie i dlaczego warto je robić wreszcie o co chodzi z dziedziczeniem.

# Zalety

Rozszerza możliwości CSS w tym o elementy znane z programowania jak instrukcje warunkowe, zmienne, funkcje.

Pozwala lepiej organizować kod.

Pozwala stosować zasadę DRY (Don't Repeat Yourself)

# Wady

Wymaga kompilacji - przeglądarki go nie czytają. Kompilacja (zamiana) z sass/scss do css.

W małych projektach to przerost formy nad treścią.

# Inne preprocesory

- Sass jest najbardziej popularny
- wiele innych np. Stylus, Less
- podobne

# Kompilacja w wersji front-end (bez node)

np. koala, prepros, Visual Studio Code (rozszerzenie)

# Struktura katalogów prostego projektu

/css
    style.css
/sass
    style.sass
    _header.sass
    _main.sass

index.html

<link rel="stylesheet" href="css/style.css">

# VSC

- Dodatek Sass
- Dodatek Live Sass Compiler
- Live Server (do podglądu na żywo)

# Sass ma dwie składnie (dwa formaty)

style.scss

style.sass

I to i to jest SASS-em, więc ma te same możliwości. Poza składnią należy pamiętać, by zapisać w pliku z odpowiednim rozszerzeniem.

# SASS a SCSS

Sass i Scss to ... Sass.

Różnią się składnią. Scss bardzo przypomina składnię CSS.

Co więcej każdy plik .css jest poprawnym plikiem z punktu widzenia składni Scss

style.css -> style.scss

# Najważniejsze elementy Sass

- zagnieżdżanie
- importowanie plików (@import)
- zmienne ($nazwa-zmiennej)
- domieszki (mixins)
- dziedziczenie (@extend)
- operatory

# Składnia - zagnieżdżenie w CSS

```
nav {
   background-color: black;
}
nav ul {
      padding:0;
      list-style: none;
}
nav ul  .list-element {
       text-transform: uppercase;
}
nav ul  .list-element  a {
       display: block;
}
```

# długie selektory i powtarzanie można traktować jako efekt uboczny

```
nav {
   background-color: black;
}
nav ul {
      padding:0;
      list-style: none;
}
nav ul  .list-element {
       text-transform: uppercase;
}
nav ul  .list-element  a {
       display: block;
}
```

# Zagnieżdżenie w Sass - składnia .scss

```
nav {
  background-color: black;
  ul {
    padding:0;
    list-style: none;
    .list-element {
      text-transform: uppercase;
      a {
        display: block;
      }
    }
  }
}
```

# Pierwsze wrażenie? Bardziej przejrzyście?

```
nav {
   background-color: black;
   ul {
      padding:0;
      list-style: none;
      .list-element {
         text-transform: uppercase;
         a {
            display: block;
          }
       }
    }
}
```

# Modułowość zagnieżdżeń - nie cała struktura html. Za dużo zagnieżdżeń odbija się na czytelności. Najlepiej zagnieżdżać moduły (czy sekcje jak tu nav czy elementy)

```
nav {
  background-color: black;
  ul {
    padding:0;
    list-style: none;
    .list-element {
      text-transform: uppercase;
      a {
                display: block;
      }
    }
  }
}
```

# Zagnieżdżenie w Sass - składnia .sass

```
nav
  background-color: black
  ul
    padding:0
    list-style: none
    .list-element
      text-transform: uppercase
      a
        display: block
```

# Zagnieżdżenie (& - rodzic) & czyli ampersand

```
a {
   background-color: black;

   &:hover {
         border-bottom:2px solid red;
    }
}
```

---

```
a {background-color: black;}

a:hover {border-bottom:2px solid red;}
```

# Zagnieżdżenie (& - rodzic)

```
div {
   background-color: black;

   &.red{
         border-bottom:2px solid red;
    }
}
```

---

```
div {background-color: black;}

div.red {border-bottom:2px solid red;}
```

# Zagnieżdżenie (& - rodzic)

```
.red {
   background-color: black;

   div & {
         border-bottom:2px solid red;
   }
}
```

---

```
.red {background-color: black;}

div .red {border-bottom:2px solid red;}
```

# Zagnieżdżenie (> dziecko)

```
ul {
   background-color: black;

   > li  {
         border-bottom:2px solid red;
   }
}
```

---

```
ul {background-color: black;}

ul>li  {border-bottom:2px solid red;}
```

# Zagnieżdżenie (& - rodzic)

```
div {
  background-color: black;

  &.first {
            border-bottom:2px solid red;

            nav & p {
            color:white;
             }
    }
}
```

---

```
div {background-color: black;}
div.first {border-bottom:2px solid red;}
nav div.first p {color: white;}
```

# Zagnieżdżenie (& - rodzic)

```
.btn {
  background-color: black;
  &-red{
    border: 2px solid red;
   }
   &-blue{
    border: 2px solid blue;
  }
}
```

```
.btn {
background-color: black;
}

.btn-red {
border: 2px solid red;
}

.btn-blue {
border: 2px solid blue;
}
```

# Importowanie

Piszemy w mniejszych fragmentach kodu (np. dla sekcji/modułu) w osobnych plikach a na końcu go importujemy (łączymy) do jednego pliku. Docelowo możemy mieć jeden plik wyjściowy i do tego zminifikowany.

header.sass

nav.scss

footer.sass

style.css

# Importowanie style.sass

dyrektywa @import

```
@import 'header.scss';
@import 'main';
```

Importujemy w pliku wynikowym.
Plik wynikowy po kompilacji umieszczamy w naszym css.

# Importowanie

style.sass

@import 'header'
@import 'main'

Import może być użyty w dowolnym miejscu pliku .scss. W tym miejscu gdzie umieszczamy pojawi się zaimportowany kod.

# Importowanie - pliki cząstkowe

Pliki cząstkowe z podkreśleniem na początku. To informacja dla kompilatora by ich nie kompilować (nie zmieniać na css).

Project

- sass
  - style.sass
  - _header.sass
  - _nav.scss
  - _btn.sass
- css
  - style.css

# Importowanie - pliki cząstkowe

zmieniany będzie tylko plik style.sass, który będzie wszystkie te pliki importował.

Project
- sass
  - style.sass
  - _header.sass
  - _nav.scss
  - _btn.sass
- css
  - style.css

style.sass

```
@import 'header'
@import 'nav'
@import 'btn'
```

# ZMIENNE W SASS

# Zmienne (.scss)

```
$fontTitle: 2.4rem;
$main-color: #e4aa19;

h1 {
        font-size: $fontTitle;
}

input.text:focus {
        border: 2px solid $main-color;
}
```

# Zmienne (.sass)

```
$fontTitle: 2.4rem;
$main-color: #e4aa19;

h1
  font-size: $fontTitle


input.text:focus
  border: 2px solid $main-color
```

# zmienne (css)

Zmienne w CSS już są, do tego są dynamiczne (widoczne w trakcie działania strony), więc mają przewagę nad preprocesorami.

```
--mainColor: #34e29a;

p {
color: var(--mainColor);
}
```

# zmienne w innych preprocesorach

Ciekawostka w Less tworzymy zmiennę za pomocą takiej składni:

@name: wartość

```
@bgcMain: #43ee29;  //Less

$bgcMain: #43ee29;  //Sass
```

# Zmienne w preprocesorach a w CSS

## Zmienne w CSS znacznie lepsze od preprocesorowych

1. Możemy na nie wpływać (przez JS/JQ)
2. Mamy wiedzę o stronie i możemy ją wykorzystać.

```
div {
        --width: 600px;
        width: calc(100% - var(--width))
}
```

# ZMIENNE

Na zmienną --width możemy wpływać podczas działania strony. W Sass nie mamy po kompilacji zmiennych i w czasie działania strony nic nie zrobimy.

# Zmienna

Raz zadeklarowana może być używana w wielu miejscach. Wystarczy potem zmienić tylko zawartość zmiennej i zmieniamy wartości wszystkich właściwości gdzie została użyta.

$text_color: #488; //scss

$text_color: #488 //sass

Możemy przechowywać tekst, liczby, wartości z jednostakami.

# Zmienna

```
$border_color: #488

p
  border: 2px solid $border_color
```

Po skompilowaniu do css

```
p {
   border: 2px solid #488;
}
```

# Zmienna - nazwy

Podkreślenie, myślniki, notacja wielbłądzia.

$btn-width: 300px //najczęściej wybierany sposób

$btn_width: 300px

$btnWidth: 300px

Wielkość liter w zmiennej ma znaczenie. Można użyć cyfry, ale nie jako pierwszego znaku.

# Zmienna - nazwy

Semantycznie odpowiadają przeznaczeniu zmiennych. Krótkie ale zrozumiałe nazwy.

```
//wersja 1 uporządkowania
$btn-bgc: #321
$btn-color: #cdd

//wersja 2 uporządkowania
$color-btn: #321
$color-h1: cadetblue
```

# Operacje matematyczne

```
$number: 10
$width: 100px

main {
        width: $number * $width;
}
```

# DOMIESZKI/WSTAWKI (MIXINS)

Stosujemy gdy określone fragmenty kodu się powtarzają (wiele razy).

# Domieszki (mixins)

wycinek kodu (zbioru właściwości css), któremu możemy nadać jakąś nazwę. Taki fragmenty do wielkrotnego użytku w różnych miejscach nadawania styli.

Zasada mixins: nie pisz dwa razy tego samego!

# Domieszki (mixins) - składnia .scss

```scss
@mixin flexCenterColumn() {
        display:flex;
        justify-content: center;
        align-items: center;
        flex-direction: column;
}

//Deklaracja (po lewej)
//Użycie (po prawej)
```

```scss
.main {
  @include flexCenterColumn()
  max-width:1400px;
}

.footer {
  @include: flexCenterColumn()
  background-color: gray;
}
```

# Domieszki (mixins) - składnia .scss

```
@mixin flex-center-column() {
        display:flex;
        justify-content: center;
        align-items: center;
        flex-direction: column;
}
```

```
.main {
   @include flex-center-column()
   max-width:1400px;
}
```

W pewien sposób rozbudowane zmienne. Zmiana w inicjacji domieszki spowoduje aktualizacje w każdym miejscu kodu.

# Domieszki (mixins) - składnia .scss

```
@mixin flex-center-column() {
        display:flex;
        justify-content: center;
        align-items: center;
        flex-direction: column;
}
```

```
.main {
display:flex;
justify-content: center;
align-items: center;
flex-direction: column;
max-width:1400px;
}
//kolejność oczywiście ma znaczenie
```

# Domieszki (mixins) - scss

```
@mixin nazwa($item1, $item2) {
        właściwość: $item1;
        właściwość: $item2;
}
```

```
selektor {
        @include nazwa(100px, #eee);
        inne właściwości
}
```

# Domieszki

Dwie dyrektywy @mixin oraz @include do zapamiętania.

@include można używać wielokrotnie w kodzie.

# Domieszki - w formacie .sass

```
=nazwaDomieszki()
        właściwość: wartość
        właściwość: wartość

p
        +nazwaDomieszki()
```

# Domieszki mogą mieć parametry

```
@mixin block($bgc, $paddingV, $paddingH) {
        background-color: $bgc;
        padding:  $paddingV $paddingH;
}
nav {
  @include block (black, 10px, 20px);
}
section {
  @include block (#fe45aa, 15%, 10rem);
}
```

# Domieszki do prefiksów

```
@mixin transform($param) {
  -webkit-transform:  $param;
  -moz-transform: $param;
  transform: $param;
}

div {
  @include transform(rotate(45deg) translateY(-50%));
}
```

# Domieszki do prefiksów

```
@mixin trans($time) {
-webkit-transition: all $time linear;
-moz-transition: all $time linear;
-o-transition: all $time linear;
transition: all $time linear;
}
div {
   @include trans(.5s)
}
```

# Domieszki z parametrami - w formacie .sass

```
=nazwaDomieszki($var1, $var2)
        właściwość: $var1
        właściwość: $var2

p
        +nazwaDomieszki(100px, #444)
```

# Dziedziczenie/rozszerzenie - reguła @EXTEND

Jeden selektor dziedziczy z innego selektora.

# Dziedziczenie/rozszerzenie - EXTENDING

Dyrektywa @extend umożliwia wykorzystanie reguł CSS użytych do jakiegoś elementu w innym elemencie.

```
.h4 {
font-size: 14px;
color: gray;
line-height: 160%;
}

.h4-title {
@extend .h4;
font-weight: bold;
color: black;
}
```

```
.h4, .h4-title {
font-size: 14px;
color: gray;
line-height: 160%;
}

.h4-title {
font-weight: bold;
color: black;
}
```

# Sass - dwie składnie

.sass

.scss

RÓŻNICE?

TAK, W SKŁADNI. JEDNAK I TO I TO JEST SASSEM.

ZOBACZMY RÓŻNICĘ

# Sass | Scss | Css

```
.class
  property: value
  element
    property: value
```

```
.class {
  property: value;
  element {
    property: value;
  }
}
```

```
.class {
  property: value;
}

.class element {
  property: value;
}
```

# Sass | Scss | Css

```
.new
  color: black
  span
    font-size: 12px
```

```
.new {
  color: black;
  span {
    font-size: 12px;
  }
}
```

```
.new {
  color: black;
}

.new span {
  font-size: 12px;
}
```

# Sass | Scss | Css

```
a
  color: black
  &:hover
    font-size: 12px
```

```
a{
  color: black;
  &:hover {
    font-size: 12px;
  }
}
```

```
a{
  color: black;
}

a:hover {
  font-size: 12px;
}
```

# Sass | Scss

```
=nameMixin($color)
+nameMixin($color)
```

```
@mixin nameMixin($color)
@include nameMixin($color)
```

Generalnie idea składni .sass jest taka, że jest ona krótsza niż .scss

# Sass | Scss a JADE?

Nawet html ma swój preprocesor, który bardzo przypomina składnię .sass dlatego Ci którzy pracują na jade pracują często na składni sass

https://naltatis.github.io/jade-syntax-docs/

```
div
  p.klasa
  p#cos
    span Idę sobie
```

```
<div>
  <p class="klasa"></p>
  <p id="cos"><span>Idę sobie</span></p>
</div
```

# Komentarze

```
// jednoliniowy
/* wielowierszowy */

/*! globalny - widoczny po kompilacji */
```

# Przejdźmy do VSC