Sass

- koncepcje
- praktyczne wykorzystanie
- Sass i BEM

Sass wspiera dwie składnie

- Składnia z wcięciami. Używamy z rozszerzeniem pliku .sass
- Składnia SCSS, z rozszerzeniem .scss
- Można w projekcie mieszać (i importować) obie składnie

W kursie pracujemy ze składnią SCSS, chyba że zaznaczone będzie inaczej.

Składnia z wcięciami (składnia Sass)

- te same możliwości co SCSS
- w tej składni nie używamy nawiasów klamrowych i średników. Koniec linii oznacza w praktyce to samo co średnik, a wcięcie (w prawo) to nawias klamrowy otwierający lub (wcięcie w lewo) zamykający.
- istnieją też inne różnice na poziomie składni

Składnia SCSS vs Sass

po kompilacji CSS będzie wyglądał tak samo

```scss
$text-size: 2rem;

@mixin reset {
 margin: 0;
 padding: 20;
 box-sizing: border-box;
}

.header {
 @include reset;
 margin: 0;
 display: flex;
 &:hover {
   background-color: black;
 }
}
```

```sass
$text-size: 2rem

=reset
  margin: 0
  padding: 20
  box-sizing: border-box

.header
  +reset
  margin: 0
  display: flex
  &:hover
    background-color: black
```

Reguła

Sass opiera się na CSS i zasady tworzenia podstawowych reguł są tu takie same. Definiujemy selektor i deklarujemy pary właściwość - wartość.

Reguła składa się z selektora (który może być grupą selektorów) i bloku deklaracji. Pojedyncza deklaracja składa się z pary właściwość - wartość. W regule może znaleźć się wiele deklaracji.

```
/* SCSS */
p, div {
 font-size: 1.2em;
}

/* Sass */
p, div
  font-size: 1.2em
```

Kompilacja i transpilacja

Kompilacja to proces przekształcenia kodu jednego języka (czy meta języka) w inny język. Najczęściej kompilacja oznacza zmianę języka o wysokiej abstrakcji na język o niskiej abstrakcji np. język programowania na język maszynowy.

Kompilator - program pozwalający skompilować wybrany język w inny język.

Transpilacia różni się od kompilacji tym, że w przypadku tej pierwszej dokonujemy zamiany jednej wersji danego języka w inną wersję tego samego języka (np. ECMAScript 6 i wyższego do ECMAScript 5) lub że jeden język zamieniamy na inny, ale o tym samym poziomie abstrakcji.

Porozmawiajmy o zagnieżdżeniach

Zagnieżdżenia

Dzięki zagnieżdżeniu nasz kod może stać się bardziej przejrzysty (o ile zagłębień nie jest zbyt dużo), możemy też pisać trochę mniej kodu (a więc pisać szybciej)

```scss
/* SCSS */
header {
 height: 30vh;
 h1 {
   font-size: 2rem;
   span {
     background-color: #333;
     color: white;
   }
 }
}
```

kompilacja

```css
/* CSS */
header {
 height: 30vh;
}

header h1 {
 font-size: 2rem;
}

header h1 span {
 background-color: #333;
 color: white;
}
```

Zagnieżdżenia

“inteligentna” kompilacja.

W oparciu o zagnieżdżenia tworzy selektor.

```
header {
 h1 {
   span {
     background-color: #333;
     color: white;
   }
 }
}
```

```
header h1 span {
 background-color: #333;
 color: white;
}
```

Zagnieżdżenia

“inteligentna” kompilacja.

Grupowanie selektorów.

```
header {
 h1, h2 {
   span {
     background-color: #333;
     color: white;
   }
 }
}
```

```
header h1 span, header h2 span {
 background-color: #333;
 color: white;
}
```

Zagnieżdżenia

“inteligentna” kompilacja.

Kompiluje tylko reguły, które posiadają deklaracje.

```
header {
 h1 {
   span {
   }
 }
}
```

```
/* pusty plik */
```

Zagnieżdżenia

Oczywiście możliwe jest użycie kombinatorów w zagnieżdżeniach. Dodane ich w elemencie nadrzędnym spowoduje, że otrzymają go wszystkie zagnieżdżone selektory

```
header > {
 h1 {
   font-size: 3rem;
 }
 h2 {
   font-size: 2.6rem;
 }
}
```

```
header > h1 {
 font-size: 3rem;
}

header > h2 {
 font-size: 2.6rem;
}
```

Zagnieżdżenia

Jeśli chcemy dodać go tylko do konkretnego selektora, użyjemy kombinatora przed zagnieżdżonym selektorem.

```
header  {
 + h1 {
   font-size: 3rem;
 }
 h2 {
   font-size: 2.6rem;
 }
}
```

```
header + h1 {
 font-size: 3rem;
}

header h2 {
 font-size: 2.6rem;
}
```

Zagnieżdżenia - płytkie zagnieżdżenia

Nie ma limitów zagnieżdżeń, ale przejrzystość zachowuje się przy 2-3 zagnieżdżeniach. Jak już wiesz użycie metodologii BEM sprawi, że te zagnieżdżenia będą rzeczywiście niezbyt głębokie.

bardzo płytkie zagnieżdżenia

Część deweloperów rezygnuje z głębszych zagnieżdżeń i używa ich przede wszystkim do pseudoelementów i pseudoselektorów oraz klas danego elementy.

& w kodzie? Zaraz wyjaśnimy.

```scss
.element {
  color: rgb(10,23,12);
  &.red {
    color: rgb(172, 84, 84);
  }
  &:hover {
    color: rgb(45,232,12);
  }
  &::after {
    content: '';
  }
}
```

```css
.element {
 color: #0a170c;
}

.element.red {
 color: #ac5454;
}

.element:hover {
 color: #2de80c;
}

.element::after {
 content: '';
}
```

Zagnieżdżenia - & (ampersand)

Ampersand symbolizuje selektor rodzica.

```
.text {
 font-size: 20px;

 .red {
   color: red;
 }
 &.red {
   color: red;
 }
}
```

```
.text {
 font-size: 20px;
}

.text .red {
 color: red;
}

.text.red {
 color: red;
}
```

Selektor rodzica odwołuje się do całego (kompletnego) selektora, który istnieje na danym poziomie zagnieżdżenia

```scss
.text {
   font-size: 20px;
   div {
      .red {
         color: red;
      }
      &.red {
         color: red;
      }
   }
}
```

```scss
.text {
   font-size: 20px;
}

.text div .red {
   color: red;
}

.text div.red {
   color: red;
}
```

selektor rodzica &

```scss
div {
 color: red;
 & .red {
   color: red;
 }
 &.blue {
   color: blue;
   &.large {
     width: 60vw;
   }
 }
}
```

Na kadym etapie zagnieżdżenia selektor rodzica wskazuje na coś innego.

```scss
div {
 color: red;
}
div .red {
 color: red;
}
div.blue {
 color: blue;
}
div.blue.large {
 width: 60vw;
}
```

Zagnieżdżenia - warto pamiętać

- Zagnieżdżenie może, ale nie musi odwzorowywać strukturę HTML. W takiej sytuacji (odwzorowanie HTML) kod Sass jest bardzo uzależniony od HTML, i jakakolwiek zmiana tam, wymusza zmianę w zagnieżdżeniach.

- Bardzo dobrym pomysłem jest wykorzystanie zagnieżdżeń dla konwencji BEM.

- Można wykorzystywać grupy selektorów w zagnieżdżeniach (jak w przykładzie niżej), choć może to zmniejszać czytelność, zyskuje natomiast DRY

```
main, .overview {
    article, .btn {
       margin: 10px;
    }
}
```

```
main article, main .btn, .overview
article, .overview .btn {
 margin: 10px;
}
```

Porozmawiajmy o zagnieżdżeniach i BEM

& a BEM? Dobrze!

Wykorzystanie selektora rodzica jako prefiks nowego selektora

```scss
.menu {
   border: 2px solid #ccc;
   &__list {
      display:flex;
      flex-direction: column;
      &--horizontal {
         flex-direction: row;
      }
   }
   &__item {
      background-color: #999;
      &--is-open {
         background-color: #333;
      }
   }
}
```

```scss
.menu {
 border: 2px solid #ccc;
}
.menu__list {
 display: flex;
 flex-direction: column;
}
.menu__list--horizontal {
  flex-direction: row;
}
.menu__item {
 background-color: #999;
}
.menu__item--is-open {
 background-color: #333;
}
```

Element możemy zagnieżdżać (slajd poprzedni) lub stworzyć dla niego nową regułą jak w przykładzie poniżej. Ma to sens szczególnie przy bardziej rozbudowanych komponentach.

```scss
.menu__item {
    padding-left:20px;
    &:hover {
        color: darkcyan;
    }
    &--active {
        background-color: #fff;
        &:hover {
            text-decoration: underline;
        }
    }
    .menu--theme-oldschool & {
        &:hover {
            color: cadetblue;
        }
    }
}
```

```scss
.menu__item {
 padding-left: 20px;
}
.menu__item:hover {
 color: darkcyan;
}
.menu__item--active {
 background-color: #fff;
}
.menu__item--active:hover {
 text-decoration: underline;
}
.menu--theme-oldschool .menu__item:hover {
 color: cadetblue;
}
```

Element możemy zagnieżdżać (slajd poprzedni) lub stworzyć dla niego nową regułą jak w przykładzie poniżej. Ma to sens szczególnie przy bardziej rozbudowanych komponentach.

```scss
.menu__item {
   padding-left:20px;
   &:hover {
      color: darkcyan;
      .menu--theme-oldschool & {
         color: cadetblue;
      }
   }
   &--active {
      background-color: #fff;
      &:hover {
         text-decoration: underline;
      }
   }
}
```

```css
.menu__item {
 padding-left: 20px;
}
.menu__item:hover {
 color: darkcyan;
}
.menu--theme-oldschool .menu__item:hover {
 color: cadetblue;
}
.menu__item--active {
 background-color: #fff;
}
.menu__item--active:hover {
 text-decoration: underline;
}
```

Porozmawiajmy o zmiennych

zmienne (variables)

Zmienną definiujemy za pomocą znaku dolara i nazwy.

```
$nazwa-zmiennej: wartość;
```

```
$text-medium-size: 1.8rem;

p {
 font-size: $text-medium-size;
}
```

Wartości zmiennej możemy potem (po deklaracji) użyć wielokrotnie w kodzie. By użyć zmiennej podajemy jej nazwę poprzedzoną znakiem dolara w miejscu właściwości.

Efekt po kompilacji:

```
p {
 font-size: 1.8rem;
}
```

zmienne - kiedy i do czego używać

Gdy z danej wartości korzystamy wielokrotnie i gdy, jeśli dojdzie do zmiany, chcemy by zmiana nastąpiła we wszystkich miejscach gdy ta zmienna występuje.

Do zmiennej możemy przypisać różne typy wartości.

Na zmiennych możemy wykonywać te same działania (funkcje, operacje matematyczne) co bezpośrednio na wartościach, które one przechowują.

```
$one: 100px;
$two: 200px;
div {
   height: 100px + 200px; //wynik to 300
   width: $one + $two; //wynik to 300
}
```

zmienne - jak nazywać

- w nazwie można używać myślnika i podkreślenia (znaczą to samo) oraz liter i cyfr, przy czym nazwa zmiennej nie może rozpoczynać się od cyfry.

```
$one-1: 1px;
$two: 2px;
div {
   width: $one_1 + $two; //one_1 i one-1 wskazują na tą samą zmienną
}
```

- Nazwa zmiennej oczywiście powinny odnosić się do jej zawartości. Nie powinna być myląca czy niewiele wnoszące (jak te powyżej). Nazwa powinna jasno wskazywać do czego służy/co zawiera zmienna.

- Nazwę zmiennej można rozpocząć od elementu, którego dotyczą, albo od nazwy właściwości do której się odnoszą. Oczywiście każda nazwa, to tylko konwencja programistów/projektu w którym się pracuje. Pracując sam, sam decydujesz :)

zmienne - jak nazywać - przykłady

```
#jakiej właściwości dotyczy (nazywanie po właściwości)
$color-text
$color-text-large
$color-main
$padding-size-default
$padding-size-thin

# jakiego elementu dotyczy (nazywanie po przeznaczeniu)
$text-content-size
$button-bg-color
$page-font-default
$page-font-alternate
$box-border
$title-color
```

zmienne - użycie złożone - przykład

```scss
//właściwości podstawowe
$red: rgb(205,21,10);
//właściwości główne
$color-text-main : $red;
$bg-color-default: #ede;
//przypisanie właściwości głównych do właściwości elementów
$article-text-color: $color-text-main;
$button-bg-color: $bg-color-default;

.article__content {
color: $article-text-color;
}
.article__button {
background-color: $button-bg-color;
}
```

```scss
.article__content {
 color: #cd150a;
}

.article__button {
 background-color: #ede;
}
```

zmienne - kompilacja

Podczas kompilacji w miejsce zmiennej pojawia się wartość, którą ta zmienna przechowuje. Oczywiście podczas kompilacji wykonywane są też działania/funkcje.

Sama deklaracja zmiennej Sass oczywiście nie jest przenoszona do CSS.

```
$color-text: #e24a55;

p {
 color: $color-text;
}

.article {
 color: $color-text;
}
```

```
p {
 color: #e24a55;
}

.article {
 color: #e24a55;
}
```

zmienne - natywna zmienna CSS

CSS dysponuje również własnymi, wspieranymi już przez wszystkie nowe i popularne przeglądarki, zmiennymi. Ich plus jest taki, że mamy je dostępne także po kompilacji.
Zmienne CSS muszą być deklarowana w regułach. By miały charakter ogólnodostępny (globalny) najlepiej umieścić je w selektorze :root lub html. Świetne do pracy gdy strona jest już skompilowana.

```
$color-text: #e24a55;

:root {
    --main-bgc: #87e001;
}

p {
    color: $color-text;
}

.article {
    color: $color-text;
    background-color: var(--main-bgc);
}
```

```
:root {
 --main-bgc: #87e001;
}
p {
 color: #e24a55;
}
.article {
 color: #e24a55;
 background-color: var(--main-bgc);
}
```

zmienne - kolejność ma znaczenie

Zmienna musi być zadeklarowana przed swoim pierwszym użyciem. W przeciwnym razie kompilator zgłosi błąd.

```
p {
 color: $color-text;
}

.article {
 color: $color-text;
}

$color-text: #e24a55;
```

```
Error: Undefined variable:
"$color-text".
```

Użycie zmiennych

Zmiennych możemy użyć przede wszystkim jako wartości właściwości.

```scss
$border-size: 2px;
$border-color: #eee;

div {
 border: $border-size solid $border-color;
}
p {
 font-size: 8 * $border-size; /* technicznie ok */
}
```

```css
div {
 border: 2px solid #eee;
}

p {
 font-size: 16px;
}
```

Zasięg zmiennych (scope)

Zmienne możemy deklarować globalnie. Dzieje się kiedy deklarujemy zmienną poza zagnieżdżeniem. Możemy powiedzieć, że jest to domyślne rozwiązanie w Sass.

Ale w niektórych sytuacjach chcemy ograniczyć zasięg zmiennej do jakiegoś zagnieżdżenia (np. do jakiegoś komponentu czy elementu, jeśli myślimy o projekcie w kategoriach BEM). O takiej zmiennej mówimy, że jest lokalna. Takie zmienna jest zadeklarowana w jakiejś regule i nie jest dostępna poza nią (poza zagnieżdżeniem)

```
$color-text: #e24a55; //globalna

.article {
 $color-title: #98a109; //lokalna
 color: $color-text;
 .article__title {
   color: $color-title;
 }
}
```

Zasięg zmiennych (scope)

Zmienna lokalna nadpisuje (tylko) w swoim zasięgu zmienną globalną o tej samej nazwie.

```scss
$color-text: #e24a55;

.article {
   $color-text: #98a109;
   color: $color-text;
   .article__title {
      color: $color-text;
   }
}
```

```css
.article {
 color: #98a109;
}

.article .article__title {
 color: #98a109;
}
```

Zasięg zmiennych (scope)

Zmienna lokalna nadpisuje w swoim zasięgu zmienną globalną o tej samej nazwie. Przy czym kolejność ma znaczenie i nadpisanie następuje dopiero w chwili gdy zmienna lokalna jest definiowana w kodzie

```
$color-text: #e24a55;

.article {
  color: $color-text;
  $color-text: #98a109;

  .article__title {
     color: $color-text;
  }
}
```

```
.article {
 color: #e24a55;
}

.article .article__title {
 color: #98a109;
}
```

Zasięg zmiennych !global

Zmienna lokalna jeśli nadamy jej flagę !global staje się globalna. Ps. na pewno kojarzysz flagi w CSS za sprawą !important (samo important nie ma oczywiście żadnego wpływu na zmienną i jest przenoszone w procesie kompilacji bo jest to flaga CSS).

```
.article {
   $color-text: #98a109 !global;
   color: $color-text;
}

.footer {
   color: $color-text !important;
}
```

```
.article {
 color: #98a109;
}

.footer {
 color: #98a109 !important;
}
```

Co warto wiedzieć jeszcze o zmiennych

- zmienna $main-font i $main_font są tym samym dla Sass i teoretycznie można używać ich zamiennie (ale dla czytelności oczywiscie tego nie rób)
- zmienne lokalne mają sens m.in. wtedy gdy piszemy kod komponentowo (każdy komponent niezależny), wtedy mamy bardziej niezależne komponenty. Do rozważenie przy metodologii BEM.
- użcie słowo kluczowego !global czyni zmienną zadeklarowaną w regule zmienną globalną.
- zmiennych używamy do wartości istotnych, używanych wielokrotnie i takich, które mogą być zmieniane.

Porozmawiajmy
Jakie typy wartości można przechowywać w zmiennych Sass

Typy wartości w Sass

- łańcuch znaków
- liczby (bez i z jednostkami)
- kolory (podobne do liczb)
- wartości logiczne (bool) i null
- listy - coś jak tablica, czyli wiele wartości w jednym obiekcie.
- mapy - coś jak obiekty/tablice asocjacyjne, czyli pary klucz - wartość

Stringi(ciąg znaków)

Mogą być w cudzysłowie - podwójny lub apostrof, ale nie muszą, przy czym, najczęściej to dobra praktyka.

```
$font1: "open sans";
$font2: open sans;

.string {
 font-family: $font1;
 font-family: "" + $font2; //sposób na umieszczenie w cudzysłowie
}
```

Liczby (wartość liczbowa, number)

Na wartościach liczbowych możemy dokonywać operacji matematycznych.

```
$example1: 20 + px;
//po kompilacji 20px, jednak tworzy wartość tekstową a nie liczbową, ponieważ px jest tu traktowane jak tekst. Na takiej wartości nie dokonamy już potem działania matematycznego.
$example2: 20 * 1px;
//20px - tworzy wartość liczbową, ponieważ 1px jest wartością liczbową.
```

Jeśli chcemy do liczby nie posiadające jednostki dodać jednostkę to pomnóżmy ją przez tę jednostkę (1px, 1rem itd.)

```
p {
   width: $example1 + 20px; // po kompilacji 20px20px
   width: $example2 + 20px; // po kompilacji 40px
}
```

Kolory

Na kolorach możemy dokonywać operacji matematycznych. Choć nie ma zazwyczaj to sensu, bo lepiej pracować z funkcjami kolorów, które wkrótce poznasz.

```
# deklaracja
$example3: #333;
$example4: rgba(20,20,30,0.2);
$example5: red;

# użycie w Sass
color: $example3 * 1.2;
color: $example4 + 12;
color: $example5 / 1.2;
```

```
# po kompilacji
color: #3d3d3d;
color: rgba(32, 32, 42, 0.2);
color: #d50000;
```

Lista

- sekwencje wartości np. fontów, marginesów i inne.
- lista jest zbiorem wartości, które od siebie oddzielamy przecinkiem lub spacją.
- Do elementu listy można odnieść się poprzez indeks (za pomocą funkcji Sass). Indeks listy liczony jest od 1

```
$colors: rgb(20,120,10), rgb(120,20,50), #333;
$sizes: 1200px, 800px, 400px;
$shadow: 2px 3px 2px 2px #ccc;
$box-margins: 20px 10px 1em, 400px 200px 5em, 5rem 2rem 100px; //lista z zagnieżdżonymi listami
```

Listę można wykorzystać w pętli Sass i przetwarzać za pomocą wbudowanych funkcji.

mapa (map)

Struktura typu obiekt. Składa się ze zbioru par klucz-wartość.

```
$fonts: (
 main: 'roboto',
 second: 'arial',
 theme-old: 'verdana'
);

//można też w jednej linii, ale mniej czytelne
$fonts: (main: 'roboto', second: 'arial', theme-old: 'verdana');
```

Funkcja pobierająca wartość z mapy

```
map-get($map: $nazwa-mapy, $key: nazwa-klucza);
map-get($nazwa-mapy, nazwa-klucza);
-------------------------------------------------------------------------------
$fonts: (main: 'roboto', second: 'arial', theme-old: 'verdana');

.example {
   font-family: map-get($fonts, second);
}
```

```
.example {
   font-family: "arial";
}
```

Jest mnóstwo wbudowanych funkcji dla list i map, których wykorzystujemy przy instrukcjach warunkowych i pętlach.

Jaki typ w zmiennej? Funkcja type-of

```
$example1: coś;
$example2: #ccc;
$example3: 300px;

.example {
   value: type-of($example1);
   value: type-of($example2);
   value: type-of($example3);
}
```

```
.example {
   value: string;
   value: color;
   value: number;
}
```

Ps. Oczywiście użycie w takim kontekście tej funkcji nie ma sensu :) Właściwość value, będzie zignorowana przez przeglądarkę. Użycie tej funkcji ma sens w instrukcja warunkowych.

/* Trochę miejsca na komentarze w Sass */
// pora by coś powiedzieć i o nich
/*! To naprawdę ważne */

Komentarze - 3 typy w Sass

/* */ - znane z CSS, wielowierszowe. widoczne po kompilacji (chyba, że do pliku skompresowanego, wtedy taki plik css nie zawiera tych komentarzy

// - znane np. z JavaScript (jednowierszowe) - nie są widoczne po kompilacji

/*! */ - widoczne nawet po kompilacji do formatu skompresowanego.

Co komentować?

- dokumentuj kod, by był dla Ciebie bardziej przyjazny, ale także łatwiej pomógł odnaleźć się innym członkom zespołu oraz osobom, które być może w przyszłości będą chcieli z niego skorzystać.
- komentuj to co uważasz, że może nie być dla Ciebie/innych jasne w przyszłości.

Dodaj komentarze dla wybranych domieszek, funkcje czy placeholders selektory

Porozmawiajmy o plikach cząstkowych i strukturze projektu...

Dzielenie i łączenie plików

- Modułowość (modularność) - podział kodu na mniejsze (tematyczne), niezależne części.
- Za sprawą instrukcji @import możemy finalnie połączyć cały nasz kod w jeden plik.
- Dzięki modułowości kod staje się odseparowany i niezależny (to oczywiście zależy też od innych rzeczy) i łatwiej nad nim pracować (nie tylko) w zespole.
- chaos vs. porządek (“chaos is a ladder”, “chaos is the most dangerous thing in the world”)

Instrukcja @import

Dodanie zawartości jednego arkusza w innym arkuszu.

W chwili kompilacji pliki są łączone. Możemy łączyć pliki obu składni.

```
// W pliku main.scss
@import 'configuration/variables.scss';
@import 'configuration/base.sass';
@import 'components/button.scss';
```

Pliki cząstkowe

```
main.scss  /* łączymy pliki cząstkowe i kompilacja */
_base.scss
_fonts.scss
_button.scss
```

Plik Sass, którego pierwszym znakiem jest podkreślenie nazywany jest plikiem cząstkowym. Informujemy w ten sposób kompilator by bezpośrednio nie kompilował tego pliku. Najczęściej jedynym plikiem bez podkreślenie na początku będzie plik główny (main.scss, style.scss). którego jedynym celem jest łączenie pozostałych plików.

Importowanie - kolejność ma znaczenie

```
@import 'config/base.sass';
@import 'components/button.scss';
@import 'config/variables.scss';
```

Kolejność ma oczywiście znaczenie. Zmienne (domieszki, funkcje itd) muszą być dostępne kiedy są używane. Należy też pamiętać, że kolejność reguł CSS to jeden z elementów kaskadowości. reguły występujące później mogą nadpisać reguły występujące wcześniej.

Importowanie - kolejność ma znaczenie

```
@import 'config/variables';
@import 'config/base';
@import 'components/button';
```

Tak będzie lepiej.

Importowanie - rozszerzenie?

Mamy plik cząstkowy _base.scss. Jak go możemy zaimportować?

```
@import 'base'; /* najlepszy sposób */
@import '_base.scss'; /* ok */
@import '_base'; /* ok */
@import 'base.scss'; /* nie zadziała, bo takiego pliku nie ma */
```

Importowanie - także w regułach

```scss
//main.scss
section {
    p {
        color: green;
    }
    @import 'box';
}

//_box.scss
div {
    border: 2px solid gray;
}
```

W ten sposób ograniczamy importowany kod tylko do reguły (i jej zagnieżdżeń). Dotyczy to reguł jak w przykładzie, ale też np. zmiennych.

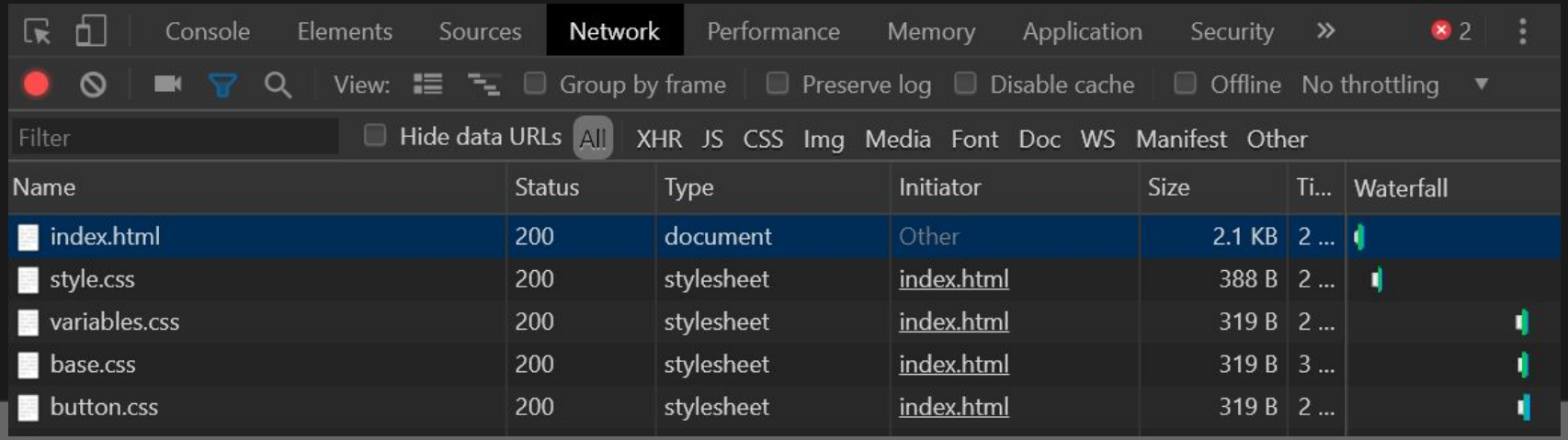
```scss
section p {
 color: green;
}

section div {
 border: 2px solid gray;
}
```

Instrukcja @import w CSS

```
# style.css
1  @import 'variables.css';
2  @import 'base.css';
3  @import 'button.css';
```

W CSS też istnieje instrukcja @import, ale w praktyce efekt jej działania ma jeden poważny minus. Wymaga to kolejnych zapytań do serwera, co spowalnia cały proces ładowania/wyświetlenia strony.



Architektura aplikacji

Poszczególne elementy umieszczamy w osobnych plikach cząstkowych.

-- konfiguracja (ustawienia podstawowe)

-- zmienne, domieszki

-- komponenty

Przejrzysty i zrozumiały kod wielokrotnego użytku

Architektura aplikacji
struktura dla małego/średniego projektu

- main.scss - importowanie pozostałych plików
- podstawowe [/base] (reset/normalize, stylowanie podstawowych elementów, layout, animacje)
- użytkowe [/utlils] - te elementy, które nie są kompilowane bezpośrednio do CSS (zmienne, domieszki, funkcje)
- komponenty [/components] - komponenty naszej aplikacji (nawet jak nie korzystamy z BEM, to taki sposób myślenia oczywiście ma sens)
- biblioteki [/libs] - zewnętrzne źródła Sass
- inne [/themes; /pages; /vendors (zamiast libs); /layouts]

Aplikacja/strona powinna umożliwiać swobodne rozrastanie - kolejne foldery i pliki tworzone przez nowe komponenty i biblioteki itd.

Przed nami domieszki.
Czyli o tym jak pisać kod wielokrotnego użytku

Mixins (domieszki)

- kod wielokrotnego użytku przypisany do identyfikatora (nazwa domieszki)
- dwie instrukcje: @mixin (definiowanie) i @include (użycie)

```
#wersja podstawowa
```

@mixin nazwa-domieszki{zawartość} - przy tworzeniu

@include nazwa-domieszki - przy użyciu (można wielokrotnie)

```
#wersja rozszerzona z parametrami
```

@mixin nazwa-domieszki($parametry) {zawartość}

@include nazwa-domieszki(argumenty)

Mixins (domieszki) - przykład prosty

@include powoduje umieszczenie (w danym miejscu) zawartości zdefiniowanej w @mixin

```scss
@mixin text {
  font-family: 'roboto', sans-serif;
  color: #333;
  font-size: 2rem;
  line-height: 1.5;
}
.article {
  text-align: justify;
  @include text;
}
```

```scss
.article {
  text-align: justify;
  font-family: 'roboto', sans-serif;
  color: #333;
  font-size: 2rem;
  line-height: 1.5;
}
```

Mixins (domieszki) - przykład zaawansowany

parametry pełną rolę zmiennych lokalnych, dostępnych w domieszce.

```scss
@mixin position($type, $top, $right, $bottom, $left) {
  position: $type;
  top: $top;
  right: $right;
  bottom: $bottom;
  left: $left;
}

.modal {
  background-color: rgb(200,200,100);
  @include position(absolute, auto, 200px, 5rem, auto);
}
```

```scss
.modal {
  background-color: rgb(200, 200, 100);
  position: absolute;
  top: auto;
  right: 200px;
  bottom: 5rem;
  left: auto;
}
```

Co może zawierać domieszka

Przede wszystkim właściwości CSS, ale nie tylko.

- zmienne
- inne domieszki
- reguły (w tym reguły zagnieżdżone)

```
@mixin box {
 div {
   width: 10vh;
   height: 10vh;
   border: 2px solid darkblue;
 }
}

@include box;
```

Co może zawierać domieszka - reguły

```scss
@mixin list {
 ul {
   list-style: none;
   display:flex;
   flex-direction: column;
   li {
     flex-basis: 200px;
   }
 }
}

nav {
 @include list;
}
```

```scss
nav ul {
  list-style: none;
  display: flex;
  flex-direction: column;
}

nav ul li {
  flex-basis: 200px;
}
```

Gdzie umieszczać domieszki (include) ?

Domieszkę najczęściej wstawiamy w regułę, choć technicznie nie ma problemu by umieścić ją także poza, co zaowocuje błędem jeśli domieszka będzie zawierała właściwości.

```
@mixin flexbox {
   display: -webkit-box;
   display: flex;
}
.box {
   @include flexbox; /* wklejone zostaną właściwości określone w domieszce */
}
@include flexbox;/* błąd przy kompilacji, bo właściwości nie mogą być poza regułą*/
Error: Properties are only allowed within rules, directives, mixin includes, or other
properties.
```

Po co mixins?

- zmieniasz coś w jednym miejscu (domieszka), zmiana jest widoczna wszędzie (gdzie jest wykorzystywana domieszka)
- stosujesz zasadę DRY (Don’t Repeat Yourself) - po stronie Sass, nie CSS.
- kod może być zbiorem określającym jakieś cechy wyglądu czy pełnić funkcję (np. technika clearfix czy prefiksy)

Nazwy

Duża dowolność, ale zasady, których warto przestrzegać:

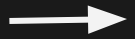
- nazwa niesie znaczenie/zawartość domieszki
- małe litery
- myślniki jako separatory słów

Czyli tak jak to jest w zmiennych czy funkcjach (które poznamy w niedalekiej przyszłości).

mixins - przykład prefiksów

```scss
@mixin flexbox {
 display: -webkit-box;
 display: -ms-flexbox;
 display: flex;
}

.box {
 color: #7d5fe6;
 @include flexbox;
}
```



```scss
.box {
 color: #7d5fe6;
 display: -webkit-box;
 display: -ms-flexbox;
 display: flex;
}
```

Domieszki - parametry

```
//nazwy parametrów dowolne, ale ze znakiem dolara
//tworzy zmienną lokalną, do wykorzystania wewnątrz domieszki

@mixin box-margin($vertical, $horizontal) {
   margin: $vertical $horizontal;
}

div {
   //przy instrukcji include przekazujemy
   //argumenty/wartości
   @include box-margin(20px,40px)
}
```

```
div {
  margin: 20px 40px;
}
```

Domieszki - domyślne wartości

```scss
@mixin box-margin($vertical: 20px, $horizontal: 40px) {
 margin: $vertical $horizontal;
}

div {
 @include box-margin(10px)
}
.article {
 @include box-margin;
//@include box-margin();
}
```

```scss
div {
 margin: 10px 40px;
}

.article {
 margin: 20px 40px;
}
```

Domieszki - domyślne wartości - wersja pro

```scss
@mixin box-margin($top: 0, $right: $top, $bottom:$top, $left:$right) {
   margin: $top $right $bottom $left;
}

div {
   @include box-margin(10px)
}
.article {
   @include box-margin();
}

p {
   @include box-margin(2.5rem, 1.5rem);
}
```

```scss
div {
   margin: 10px 10px 10px 10px;
}

.article {
   margin: 0 0 0 0;
}

p {
   margin: 2.5rem 1.5rem 2.5rem 1.5rem;
}
```

Domieszki - kolejność i brak argumentu

```scss
@mixin box-margin($horizontal, $vertical) {
 margin: $vertical $horizontal;
}

div {
 @include box-margin(10px, 15px)
}
.article {
 @include box-margin(20%);
}
```

Compilation Error

Error: Mixin box-margin is missing argument $vertical.

```scss
div {
 margin: 15px 10px;
}

// do kompilacji nie dojdzie przez błąd ale gdyby nie on efekt dla div byłby taki jak powyżej.
```

Domieszki - domyślne wartości równa null

```scss
@mixin box-margin($vertical: null, $horizontal: $vertical) {
 margin: $vertical $horizontal;
}

div {
 @include box-margin(15px)
}
.article {
 @include box-margin();
}
```

```scss
div {
 margin: 15px 15px;
}

.article {
}
```

Domieszki - domyślne wartości - użycie zmiennych

```scss
$font-size: 1rem;
$text-color: #220299;

@mixin title ($size: $font-size * 2, $color: $text-color ) {
     font-size: $size;
     color: $color;
}

h1 {
     @include title;
}
```

```css
h1 {
 font-size: 2rem;
 color: #220299;
}
```

Domieszki - odwołanie do nazw parametrów

```scss
$text-color: #220299;

@mixin title ($size: 2rem, $color: $text-color ) {
   font-size: $size;
   color: $color;
}
// Dzięki użyciu nazw parametrów, kolejność nie ma znaczenia
h1 {
   @include title($color: blue);
}
```

```scss
h1 {
  font-size: 2rem;
  color: blue;
}
```

Domieszki - instrukcja @content

Używając domieszki oprócz argumentów możemy przekazać dodatkowe właściwości, które mają być umieszczone wraz z tymi zadeklarowanymi. By tak się stało musimy umieścić w deklaracji instrukcję @content, a przy umieszczeniu domieszki przekazać do niej właściwości (między nawiasy klamrowe)

```
@mixin title ($size: 2rem) {
    font-size: $size;
    @content;
}
h1 {
    @include title(2.2rem) {
       line-height: 150%;
       color: blue;
    }
}
```

```
h1 {
   font-size: 2.2rem;
   line-height: 150%;
   color: blue;
}
```

Na co uważać przy mixins?

- twórz i stosuj z umiarem,
- gdy jakaś grupa właściwości jest często powtarzana,
- nie twórz zbyt rozległych (zbyt wiele właściwości) gdyż wtedy nie są uniwersalne,
- z umiarem stosuj też parametry w domieszkach. Część ich potrzebuje, ale wiele nie.
- kod domieszki nie jest generowany w CSS (podobnie jak zmienna), przy kompilacji są jedynie wstawiane właściwości domieszki w miejscu w którym użyjemy instrukcji @include
- nazwy domieszek box-position i box_position (- i _)są traktowane tak samo (podobnie jak przy zmiennych)
- domieszki można (i powinno się) umieszczać w osobnym pliku cząstkowym (np _mixins.scss)

Mixins i duplikowanie kodu

Kod który powstaje po kompilacji jest w praktyce zduplikowany tyle razy ile razy została użyta domieszka. Czy to problem?

I tak i nie. Plik CSS rzeczywiście jest większy, ale z punktu widzenia jego rozmiaru czy szybkości wczytywania danych przez przeglądarkę nie ma to istotnego znaczenia. Pamiętaj, że przede wszystkim liczy się wersja deweloperska na której pracujesz. Dzięki domieszką to właśnie ona jest czytelniejsza i zgodna z zasadą DRY. CSS, który powstaje po kompilacji jest dla przeglądarki, a nie dla ludzi.

A co z dziedziczeniem?
Porozmawiajmy czy jest coś takiego w Sass

Dziedziczenie w Sass?

Przyjrzyjmy się zasadom działania **instrukcji @extend**

Dziedziczenie/Rozszerzenie (Extend)

- rozbudowujemy regułę o kolejne selektory. Brzmi zagadkowo ;)

- nie jest związane z dziedziczeniem znanym z kaskadowości (gdzie część właściwości jest powielana z rodzica). Instrukcja @Extend to zupełnie inne znaczenie dziedziczenia. Lepiej pasuje tu określenie rozszerzenie.

- wymaga dużej uwagi, bo łatwo coś zepsuć. Budzi kontrowersje...

instrukcja @Extend

```
.text {
   font-size: 20px;
   font-family: Arial, Helvetica, sans-serif;
   color: darkgray;
}

.article__text {
   @extend .text;
}

.footer__copyright {
   @extend .text;
   font-size: 22px;
}
```

instrukcja @Extend

```scss
.text {
 font-size: 20px;
 font-family: Arial, Helvetica, sans-serif;
 color: darkgray;
}

.article__text {
 @extend .text;
}

.footer__copyright {
 @extend .text;
 font-size: 22px;
}
```

```scss
.text, .article__text, .footer__copyrigth {
 font-size: 20px;
 font-family: Arial, Helvetica, sans-serif;
 color: darkgray;
}

.footer__copyright {
 font-size: 22px;
}
```

instrukcja @Extend

```
.text {
   font-size: 20px;
   font-family: Arial, Helvetica, sans-serif;
   color: darkgray;
}


.footer__copyright {
// extend oznacza instrukcję: “kompilatorze dodaj ten selektor do reguły w której jest użyty selektor text”
   @extend .text;
   font-size: 22px;
   text-transform: uppercase;
}
```

```
.text, .footer__copyright {

}

.footer__copyright {
 font-size: 22px;
 text-transform: uppercase;
}
```

instrukcja @Extend - problemy

rozszerzenie wszystkie selektory które zawierają dany dany selektor

```
.text {
   font-size: 20px;
   font-family: Arial, Helvetica, sans-serif;
}

header .text {
   font-weight: bold;
}

.footer__copyright {
   @extend .text;
   text-transform: uppercase;
}
```

```
.text, .footer__copyright {
   font-size: 20px;
   font-family: Arial, Helvetica, sans-serif;
}

header .text, header .footer__copyright {
   font-weight: bold;
}

.footer__copyright {
   text-transform: uppercase;
}
```

instrukcja @Extend - problemy 2

rozszerzenie zagnieżdżeń. Wyobraź sobie duży projekt.

```
.box {
   color: red;
   .article {font-size: 12px;}
   .title {text-decoration: underline;}
}

.intro {
   @extend .box;
   text-transform: uppercase;
}
```

```
.box, .intro {
 color: red;
}

.box .article, .intro .article {
 font-size: 12px;
}

.box .title, .intro .title {
 text-decoration: underline;
}

.intro {
 text-transform: uppercase;
}
```

Domieszki vs. Rozszerzenia

Domieszka “wstrzykuje” kod domieszki w miejscu gdzie jest użyta. Generuje dodatkowy kod w CSS (powtórzenia)

Instrukcja @extend rozszerza każdą regułę zawierająca wskazany selektor o inny selektor. Po kompilacji w CSS nie powiela właściwości.

Domieszki vs. Rozszerzenia

Domieszka “wstrzykuje” kod domieszki w miejscu gdzie jest użyta

```scss
@mixin example ($color, $size) {
   color: $color;
   font-size: $size * 1px;
}

p {
@include example(red, 20);
   line-height: 150%;
}
```

```scss
p {
   color: red;
   font-size: 20px;
   line-height: 150%;
}

// Jeśli użyjemy w innych
miejscach to znajdzie się w nich
kopia tych właściwości (kopia
zawartości domieszki).
```

Domieszki vs. Rozszerzenia

```
.selector1 {
   color: red;
}

.selector2 {
 @extend .selector1;
   font-size: 1.2em;
}

.selector3 {
 @extend .selector1;
 @extend .selector2;
   background: blue;
}
```

Instrukcja extend rozszerza każdą regułę zawierająca wskazany selektor o inny selektor.

```
.selector1, .selector2, .selector3 {
 color: red;
}

.selector2, .selector3 {
 font-size: 1.2em;
}

.selector3 {
 background: blue;
}
```

zasady @extend

1. miejsc wrzucenia @extend w regule nie ma znaczenia

2. @extend można użyć nawet przed miejscem w którym pojawia się dany selektor

```
.intro {
   color: blue;
   //2. odwołanie do .box możliwe zanim pojawia się w kodzie.
   //Jest ok, bo to nie jest zmienna
   @extend .box;
   //1. można tu, ale można na początku reguły
}
.box {
    color: red;
    font-size: 12px;
}
```

zasady @extend

3. znaczenie ma pozycja i siła selektora (kaskadowość)

```
.intro {
   @extend .box;
   color: blue;
}
.box {
    //3. kaskadowość wskaże na color red dla .intro
    color: red;
    font-size: 12px;
}
```

```
.intro {
 color: blue;
}

.box, .intro {
 color: red;
 font-size: 12px;
}
```

zasady @extend

3. znaczenie ma pozycja i siła selektora (kaskadowość)

```
.box {
    color: red;
    font-size: 12px;
}

.intro {
    //3. tym razem kolor blue dla właściwości intro
    color: blue;
    @extend .box;
}
```

```
.box, .intro {
   color: red;
   font-size: 12px;
}

.intro {
   color: blue;
}
```

Selektory zastępcze (placeholders, ciche klasa)

- selektor wirtualny, tworzy wirtualną regułę, do której można dodawać (poprzez @extend) kolejne elementy.
- selektor zastępczy sam w sobie nie jest kompilowany, więc jeśli nie zostanie do niego dodany inny selektor za pomocą extend to z punktu widzenia CSS nie istnieje.
- świetnie się sprawdza jako baza do rozszerzeń
- na fakt, że mamy do czynienia z selektorem zastępczym (placeholderem) wskazuje nam symbol procentu na początku.

```
%nazwa-selektora-zastępczego {
    deklaracje
}
```

Selektory zastępcze - przykład z rozszerzeniem

```scss
%flex-list {
   display: -webkit-box;
   display: -ms-flexbox;
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
}
ul {
   @extend %flex-list;
   list-style: none;
}
.news {
   border: 2px solid gray;
   flex-direction: column;
   @extend %flex-list;
}
```

```scss
ul, .news {
   display: -webkit-box;
   display: -ms-flexbox;
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
}

ul {
   list-style: none;
}

.news {
   border: 2px solid gray;
   flex-direction: column;
}
```

Selektory zastępcze - przykład bez rozszerzenia

```
%flex-list {
   display: -webkit-box;
   display: -ms-flexbox;
   display: flex;
   flex-direction: row;
   flex-wrap: nowrap;
}
ul {
   list-style: none;
}
.news {
   border: 2px solid gray;
   flex-direction: column;
}
```

```
ul {
   list-style: none;
}

.news {
   border: 2px solid gray;
   flex-direction: column;
}
```

Korzystać z @extend czy @mixin/@include?

#extend

```
%red {
   color: red;
}
p {
   @extend %red;
}
div {
   @extend %red;
}
```

#po kompilacji

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

#mixin

```
@mixin red {
   color: red;
}

p {
   @include red;
}

div {
   @include red;
}
```

#po kompilacji

```
p {
 color: red;
}

div {
 color: red;
}
```

Korzystać z @extend?

- najczęściej ma to sens dla prostych selektorów (selektorów, które nie są zagnieżdżane) lub jeszcze lepiej placeholderów
- część osób w ogóle z nich nie korzysta (mixiny im wystarczają), a część osób bardzo lubi rozszerzenia. Wybór należy do Ciebie.
- grupowanie selektorów, to na pewno ciekawe rozwiązanie, ale wiele też zależy od tego jako konwencję wybierzesz (BEM, modyfikacje BEM, czy inny sposób).
- zazwyczaj lepszym rozwiązaniem są, w mojej opinii, domieszki. Generują więcej kodu po stronie CSS, ale są bardziej elastyczne (parametry) i łatwiej utrzymać mi strukturę projektu (również w konwencji BEM).

Porozmawiamy o operacjach matematycznych

Obliczenia podstawowe

```
width: 20px + 200px; // 220px po kompilacji
margin: 40px - 30; // 10px po kompilacji
height: 10vh + 20px; // błąd Error: Incompatible units: 'vh' and 'px'.
//Przed wyświetlenie strony w przeglądarce wartość jednostki vh/vw nie jest znana.

height: calc(10vh + 20px); // w sytuacji jak wyżej powinniśmy skorzystać z funkcji calc.
Kompilacja nic tutaj nie zmieni i w takiej wersji ten kod znajdzie się w przeglądarce.
```

- pamiętaj o spacji między znakami np. 120px -5 kompilator traktuje , to jako dwie wartości a nie działanie i skompiluje do np. width: 120px -5;. To nie to samo co 120px - 5 czyli działanie matematyczne 120px minus 5 i kompilacja do 115px.

Obliczenia podstawowe - zmienne i mnożenie

```
$page-margin: 20px;
$margin-increase: 1.2;

.menu__item {
   margin: $page-margin * $margin-increase;  //po kompilacji 24px
}
```

!Jednostki przy mnożeniu używamy tylko przy jednej jednostce.

Obliczenia podstawowe - dzielenie

```
$page-margin: 20px;
$margin-increase: 1.2;
.menu__item {
   margin: $page-margin / $margin-increase;  //po kompilacji 16.66667px
}
```

!Jeśli dzielimy liczby bezpośrednio to powinniśmy użyć nawiasu

```
width: 20px/2;  // po kompilacji 20px/2
width: (20px/2); // po kompilacji 10px
width: (20px/2px); // po kompilacji 10
width: ($page-margin + 20px/2 * $margin-increase); // 32px
```

Porozmawiajmy o interpolacja

Interpolacja - #{$zmienna}

Zawartość zmiennej możemy umieścić w miejsce oczekiwanej właściwości - to wiemy do tej pory. Jednak dzięki interpolacji możemy wykorzystywać zmienne także w innych miejscach kodu. Interpolacja to wstawienie zawartości zmiennej w miejscu gdzie używamy składni z hashem i nawiasami klamrowymi.

```
$default-size: 10;
$class-large-font: 'xxl';

p.#{$class-large-font} {
   font-size: $default-size * 2px;
}
```

```
p.xxl {
 font-size: 20px;
}
```

Wbudowane funkcje kolorów - przykłady

Funkcja w Sass - darken i lighten

- funkcja w Sass przetwarza dane wejściowe i coś zwraca
- funkcje dla kolorów zwracają nowe kolory

```
$font-color: rgb(12,122,222);

.box {
  color:  $font-color;
  h1 {
   color: lighten($font-color, 25%);
   border-bottom: 2px solid darken($font-color, 10%)
  }
}
```

```
.box {
  color: #0c7ade;
}

.box h1 {
  color: #72b8f7;
  border-bottom: 2px solid #095fae;

}
```

#0c7ade;

#72b8f7;

#095fae;

Funkcja mix - przyjmuje dwa argumenty

```scss
$font-color: rgb(12,122,222);
$mix-color: #ee44ee;

.box {
  color: mix($font-color, red);
  h1 {
    color: mix($font-color,
    $mix-color);
  }
}
```

```scss
.box {
  color: #863d6f;
}

.box h1 {
  color: #7d5fe6;
}
```

Funkcja w Sass - funkcja w funkcji

Przy tej okazji warto wiedzieć, że jako argument w funkcjimoże być przekazana inna funkcja. Oczywiście w taki sposób jak poniżej nie ma sensu ich stosować (dobrze to jednak obrazuje o co chodzi), ale sytuacje gdzie jedna funkcja wykorzystuje jako argument coś zwracane z innej funkcji są w Sass obecne.

```
$font-color: rgb(12,122,222);

.box {
   color:  $font-color;
   h1 {
      color: darken(lighten($font-color, 25%), 25%);
   }
}
```

```
.box {
   color: #0c7ade;
}

.box h1 {
   color: #0c7ade;
}
```

Porozmawiajmy o instrukcjach warunkowych

instrukcje warunkowe - @if

- @if generowanie kodu tylko po spełnianiu warunku

```
$box-width: 180px;

p {
   @if ($box-width < 200) {
     border-color: red;
   }
   color: red;
}
```

```
p {
 border-color: red;
 color: red;
}
```

instrukcje warunkowe - @if @else

- @else - co, gdy warunek nie jest prawidłowy

```scss
$box-width: null;
p {
   @if ($box-width) {
      border-color: red;
   }
   @else {
      border-color: yellow;
   }
}
```

```scss
p {
 border-color: yellow;
}

/*
warunek w środku jest zamieniany
na false/true. Null, jest
konwertowany* na false.
* konwersja - zamiana typu
*/
```

instrukcje warunkowe - @if @else if

- @else if - dodatkowe sprawdzenie warunku

```scss
$box-width: 90px;
$border-size: medium;
p {
  @if ($box-width == 100) {
     border-color: red;
  }
  @else if ($box-width != 100 and $border-size == large) {
     border-color: yellow;
  }
  @else {
     border-color: gray;
  }
}
```

```scss
p {
    border-color: gray;
}
```

Domieszki i @if

```scss
@mixin box-margin($top: 0, $right: $top, $bottom:$top, $left:$right) {
   @if($top != $bottom or $right != $left) {
       margin: $top $right $bottom $left;
   }
   @else if ($top == $bottom and $right == $left and $top == $left) {
       margin: $top;
   }
   @else {
       margin: $top $right;
   }
}
div {
   @include box-margin(10px)
}
.article {
   @include box-margin(20px, 30px, 40px);
}
p {
   @include box-margin(2em, 1em, 3em, 1.5em);
}
```

```scss
div {
  margin: 10px;
}

.article {
  margin: 20px 30px 40px 30px;
}

p {
  margin: 2em 1em 3em 1.5em;
}
```

Porozmawiajmy o tworzenie i wykorzystywanie funkcji

Własne funkcje

- Zadaniem funkcji jest przetworzyć dane wejściowe i zwrócić wartość, która będzie użyta jako wartość właściwości. Dotyczy to zarówno funkcji kolorów, które poznaliśmy jak i funkcji, które możemy zadeklarować (stworzyć) sami.

```
@function nazwa-funkcji($parametr1, $parametr2) {
      //ciało funkcji
      @return zwracana-wartość;
};

div {
    nazwa-właściwości: nazwa-funkcji() //po kompilacji wartość
}
```

Własne funkcje - przykład

```scss
$border-width: 2px;

@function border-example() {
    @return 20 * $border-width;
}

button {
    border-width: border-example();
}
```

```scss
button {
    border-width: 40px;
}
```

Własne funkcje

```scss
$border-width: 2px;

@function multi-border($multiply: null, $width: $border-width,) {
  @if($multiply == null) {
    @return $width;
 }
 @else if ($multiply == 0) {
   @return 0;
 }
 @else if (type-of($multiply) == number and $multiply > 0) {
   @return $width * $multiply;
 }
 @else {
   @return null
   //null przypisany do właściwości powoduje,
   //że nie będzie ona wyświetlana
 }
};
```

```scss
button {
 border-width: multi-border($width: 10px); //10px
 border-width: multi-border(-2, 10px); //brak
 border-width: multi-border(0.2, 5px); //1px
 border-width: multi-border(4); //8px
}
```

Media Queries (instrukcja @media)

Kilka zdań o Media Queries i Sass, choć temat responsywności szerzej omówimy w dalszej części kursu.

Mnóstwo technik MQ & Sass (& BEM)

Jak używać (np. w zagnieżdżonych czy na końcu kodu), jakich technik (mapy, domieszki, instrukcje warunkowe).

Przykład 1 - z zagnieżdżeniem,
Przykład 2 - z mapami, interpolacją i domieszkami.

Zagnieżdżanie Media Queries

```scss
.menu__item {
   padding:20px;
   font-size: 1.6rem;
   &--active {
      background-color: #fff;
      @media (min-width: 640px) {
         background-color: #ddd;
      }
   }
   @media (min-width: 640px) {
      font-size: 1.8rem;
   }
}
```

```scss
.menu__item {
 padding: 20px;
 font-size: 1.6rem;
}
.menu__item--active {
 background-color: #fff;
}
@media (min-width: 640px) {
 .menu__item--active {
   background-color: #ddd;
 }
}
@media (min-width: 640px) {
 .menu__item {
   font-size: 1.8rem;
 }
}
```

Breakpoints

Tworzymy mapę, do której przyporządkowujemy wartości. To poniżej jest jedynie propozycją.

```
$breakpoints: (
  xsmall: (min-width: 460px),
  small: (min-width: 640px),
  medium: (min-width: 800px),
  large: (min-width: 1024px),
  xlarge: (min-width: 1200px),
  xxlarge: (min-width: 1440px)
);
```

Breakpoints - użycie z domieszką

```scss
$breakpoints: (
  xsmall: (min-width: 460px),
  small: (min-width: 640px),
  medium: (min-width: 800px),
  large: (min-width: 1024px),
  xlarge: (min-width: 1200px),
  xxlarge: (min-width: 1440px),
);
```

```scss
@mixin mq($breakpoint) {
    $size: map-get($breakpoints, $breakpoint);
    @if($size) {
        @media #{$size} {
            @content;
        }
    }
    @else {
        @error '"#{$breakpoint}" - Nie rozpoznaje tej wielkosci';
    }
}
```

Breakpoints - użycie z domieszką

```
$breakpoints: (
  xsmall: 'min-width: 460px',
  small: 'min-width: 640px',
  medium: 'min-width: 800px',
  large: 'min-width: 1024px',
  xlarge: 'min-width: 1200px',
  xxlarge: 'min-width: 1440px'
);
```

```
@mixin mq($breakpoint) {
   $size: map-get($breakpoints, $breakpoint);
   @if($size {
      @media (#{$size}) {
         @content;
      }
   }
   @else {
      @error '"#{$breakpoint}" - Nie rozpoznaje tej wielkości';
   }
}
```

```scss
//1. zdefiniowanie rozdzielczości kluczowych
$breakpoints: (xsmall: (min-width: 460px), small:
(min-width: 640px), medium: (min-width: 800px),
large: (min-width: 1024px),  xlarge: (min-width:
1200px), xxlarge: (min-width: 1440px));

//2. definicja domieszki do zapytań o media
@mixin mq($breakpoint) {
 $size: map-get($breakpoints, $breakpoint);
 @if($size) {
   @media #{$size} {
     @content;
   }
 } @else {
   @error '"#{$breakpoint}" - błąd';
 }
}
```

```scss
.box {
   .title {
      font-size: 20px;
      //3. użycie domieszki do MQ
      @include mq(xsmall) {
         font-size:18px;
      }
   }
}

//4. Po kompilacji
.box .title {
   font-size: 20px;
}
@media (min-width: 460px) {
 .box .title {
   font-size: 18px;
 }
}
```

```
//1. zdefiniowanie rozdzielczości kluczowych
$breakpoints: (xsmall: (min-width: 460px), small:
(min-width: 640px), medium: (min-width: 800px),
large: (min-width: 1024px),  xlarge: (min-width:
1200px), xxlarge: (min-width: 1440px));

//2. definicja domieszki do zapytań o media
@mixin mq($breakpoint) {
 $size: map-get($breakpoints, $breakpoint);
 @if($size) {
   @media #{$size} {
     @content;
   }
 } @else {
   @error '"#{$breakpoint}" - błąd, nie wiem co to';
 }
}
```

```
.box {
   .title {
      font-size: 20px;
      //3. użycie domieszki do MQ z BŁĘDEM
      @include mq(xxsmall) {
         font-size:18px;
      }
   }
}

//4. Błąd kompilacji z informacją jaką podaliśmy
w instrukcji.

Compilation Error
Error: "xxsmall" - błąd, nie wiem
co to
```

Co dalej?

Zdaje sobie sprawę, że potrzebujesz praktyki by te rzeczy Ci się utrwaliły! Pojawi się więc praktyczny przykład w którym wykorzystamy wiedzę z BEM i Sass oraz innych technik, narzędzi i technologii, których uczysz się ze mną w tym kursie. Będzie to siódmy, ostatni projekt w tym kursie.