

NATALIA SOKOŁOWSKA

AKADEMIA BEDEPROGRAMISTKA

Krótko o SCSS

BEDEPROGRAMISTKA.PL

Krótkie wprowadzenie do Sass i SCSS.	3
Instalacja Sass.....	5
Krok 1.....	5
Windows	5
MacOs	5
Krok 2.....	5
Kompilacja SCSS do CSS.....	7
Architektura plików	9
Importowanie plików	12
Zmienne.....	13
media queries	14
Zagnieżdżanie.....	15
Mixins (domieszka)	17
Przykład bardzo prostego mixin do tworzenia obramowania z wykorzystaniem parametru.....	19
Przykład mixin do centrowania pozycji bez parametru.....	19
Przykład mixin do centrowania pozycji z parametrami.....	20
Przykład mixin do tworzenia transition z trzema opcjonalnymi parametrami.....	22
Przykład mixin do tworzenia gradientu z trzema parametrami	22
Przykład mixin do zmiany px na rem	23
Przykład mixin do tworzenia strzałki o określonej wielkości (3 parametry).....	25
Extend	26
Wbudowane funkcje w SCSS	27
SCSS i Bootstrap	28
Pobieranie plików z Bootstrapa	28

Krótkie wprowadzenie do Sass i SCSS.

Sass to **preprocesor** *CSS*, który fantastycznie ułatwia pracę z *CSS*. Za jego pomocą możemy sobie tworzyć reguły, do których później będziemy odwoływać się krótkim poleceniem.

Praca z preprocesorem polega na tym, że tworzy się style w oddzielnym pliku. Następnie ten plik jest kompilowany tzn. przerabiany na zwykły *CSS*, który jest rozumiany przez przeglądarki.

Preprocesory powstały w myśl zasady *DRY* – *Don't Repeat Yourself*. Dzięki nim nie powtarzamy tego samego kodu, tylko raz go definiujemy, a następnie odwołujemy się do niego.

Istnieje kilka preprocesorów: *Less*, *Sass* czy np. *Stylus*.

My zajmiemy się *Sass*em.

Używając *Sass* masz do dyspozycji dwie składnie – *Sass* oraz *SCSS*. Różnią się one tym, że *SCSS* wymaga od nas, abyśmy używali średników i nawiasów. Aby wybrać składnię *SCSS* wystarczy, że zapiszemy nasz plik z rozszerzeniem *.scss*. Gdybyś jednak chciał pisać w *Sass* musiałbyś wybrać rozszerzenie *.sass*.

Tak może wyglądać kod napisany w *SCSS*:

```
// Variables

$mainColor: #cccccc;
$fontColor: #000000;
$blueColor: #7dc3cb;
$yellowColor: #d7b410;


// Extend

.underline-h2 {
  content: "";
  display: block;
  border: 1px solid $blueColor;
  width: 200px;
```

```

margin: 30px auto;
}

// Mixins

@mixin border($color) {
  border: 1px solid $color;
}

nav {
  color: $fontColor;
  background-color: $mainColor;

  ul {
    margin: 0;
    list-style: none;
  }

  li {
    text-align: center;
  }

  a {
    color: $yellowColor;
  }
}

.icon-fa {
  @include border($blueColor);
}

h2 {
  @extend .underline-h2;
}

```

Bardzo przydatnym narzędziem na początku drogi jest strona <https://www.sassmeister.com>

Strona ta kompiluje w obie strony wpisany kod i pokazuje Ci, w którym miejscu znajdują się ewentualne błędy.

Instalacja Sass

Krok 1

Windows

Pierwszą kwestią jest instalacja *Ruby*, o ile jeszcze nie masz go zainstalowanego.

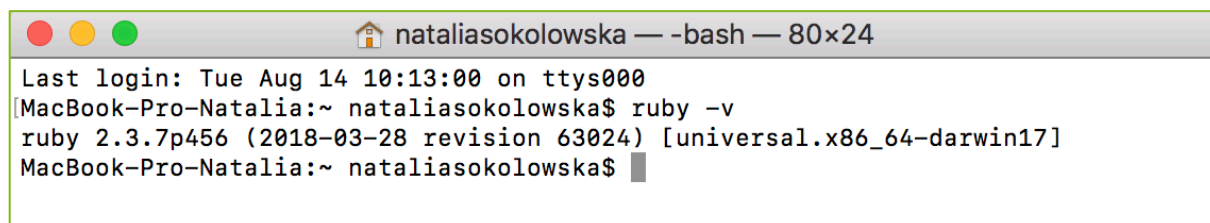
Wejdź na stronę <https://rubyinstaller.org/downloads/> i pobierz plik. Następnie zainstaluj go.

MacOs

Nie musisz nic instalować. Przejdź do kolejnego kroku.

Krok 2

Otwórz terminal i sprawdź czy Ruby jest zainstalowany. Wpisz komendę *ruby -v*

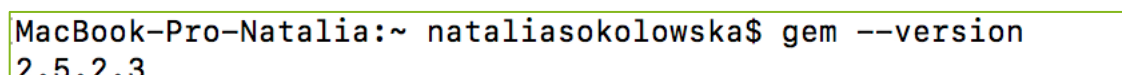


```
nataliasokolowska — -bash — 80x24
Last login: Tue Aug 14 10:13:00 on ttys000
[MacBook-Pro-Natalia:~ nataliasokolowska$ ruby -v
ruby 2.3.7p456 (2018-03-28 revision 63024) [universal.x86_64-darwin17]
MacBook-Pro-Natalia:~ nataliasokolowska$
```

Następnie musimy zainstalować Sassa. Najpierw sprawdzimy czy już nie jest zainstalowany komendą *gem --version*. Jeśli jest nie musisz nic robić. Jeśli nie ma:

W terminalu wpisz komendę:

- Windows - *gem install sass*
- Mac OS i Linux - *sudo gem install sass*



```
MacBook-Pro-Natalia:~ nataliasokolowska$ gem --version
2.5.2.3
```

Sprawdź czy udało się wszystko zainstalować komendą `sass -v`.

MacOs:

Gdyby pojawił Ci się błąd: `ERROR: Failed to build gem native extension` wpisz w terminalu komendę `xcode-select --install`. Zostanie pobrane oprogramowanie, dzięki któremu bez problem zainstalujesz `Sass`. Po instalacji wpisz w terminalu komendę `sudo gem install sass` a następnie sprawdź wersję zainstalowanego `Sassa` komendą `sass -v`.

```
MacBook-Pro-Natalia:~ nataliasokolowska$ xcode-select --install
xcode-select: note: install requested for command line developer tools
MacBook-Pro-Natalia:~ nataliasokolowska$
MacBook-Pro-Natalia:~ nataliasokolowska$ sudo gem install sass
Password:
Building native extensions. This could take a while...
Successfully installed ffi-1.9.25
Fetching: rb-inotify-0.9.10.gem (100%)
Successfully installed rb-inotify-0.9.10
Fetching: sass-listen-4.0.0.gem (100%)
Successfully installed sass-listen-4.0.0
Fetching: sass-3.5.7.gem (100%)

Ruby Sass is deprecated and will be unmaintained as of 26 March 2019.

* If you use Sass as a command-line tool, we recommend using Dart Sass, the new
  primary implementation: https://sass-lang.com/install

* If you use Sass as a plug-in for a Ruby web framework, we recommend using the
  sassc gem: https://github.com/sass/sassc-ruby#readme

* For more details, please refer to the Sass blog:
  http://sass.logdown.com/posts/7081811

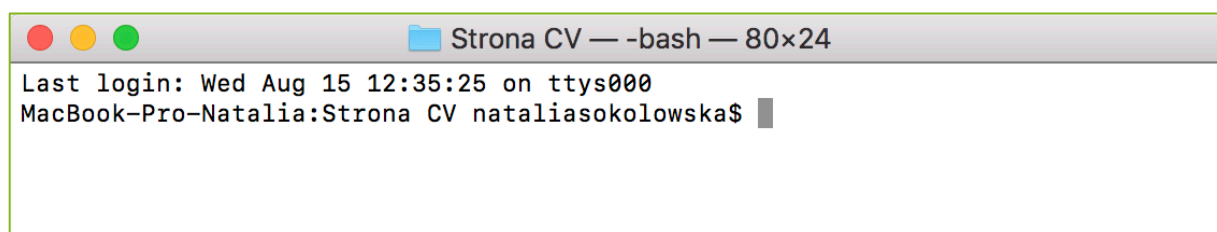
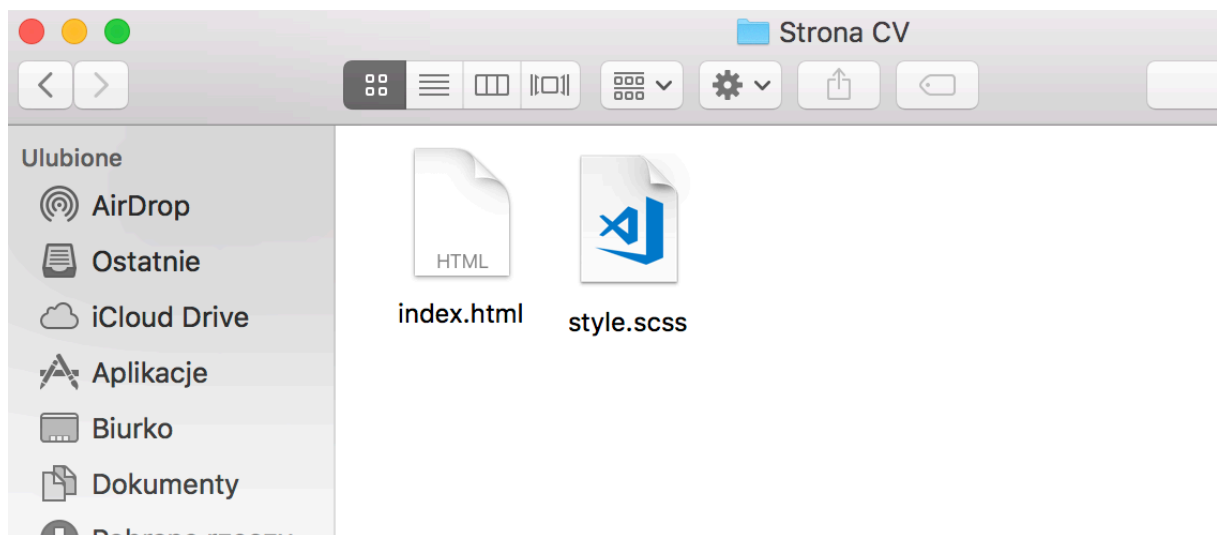
Successfully installed sass-3.5.7
Parsing documentation for ffi-1.9.25
Installing ri documentation for ffi-1.9.25
Parsing documentation for rb-inotify-0.9.10
Installing ri documentation for rb-inotify-0.9.10
Parsing documentation for sass-listen-4.0.0
Installing ri documentation for sass-listen-4.0.0
Parsing documentation for sass-3.5.7
Installing ri documentation for sass-3.5.7
Done installing documentation for ffi, rb-inotify, sass-listen, sass after 32 seconds
4 gems installed
MacBook-Pro-Natalia:~ nataliasokolowska$ sass -v
Ruby Sass 3.5.7
MacBook-Pro-Natalia:~ nataliasokolowska$
```

Kompilacja SCSS do CSS

Jest wiele metod kompilacji plików. Można to zrobić za pomocą programu Koala <http://koala-app.com>. Można też to robić za pomocą Gulpa, Webpacka czy innego task runnera.

Ja Ci pokażę metodę kompilacji plików za pomocą terminala, czyli nic nie trzeba instalować 😊.

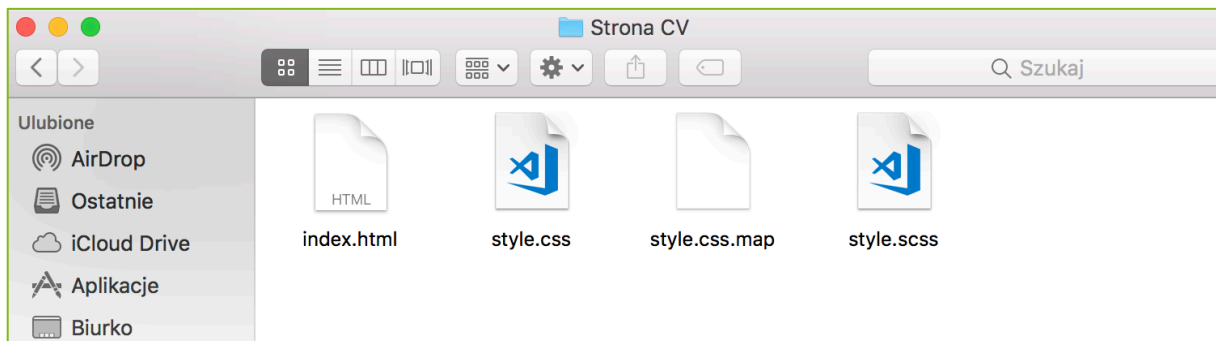
Na początek musimy otworzyć terminal i wejść do folderu, w którym mamy zapisane pliki *scss*.



Następnie wpisujemy komendę *sass --watch style.scss:style.css*

Od teraz pliki są śledzone przez komendę *--watch* i przy każdej zmianie w pliku *style.scss* są zmiany kompilowane do pliku *style.css*.

```
Strona CV — fsevent_watch ◀ sass --watch style.scss:style.css — 80x23
Last login: Wed Aug 15 13:21:24 on ttys001
MacBook-Pro-Natalia:Strona CV nataliasokolowska$ sass --watch style.scss:style.c
ss
>>> Sass is watching for changes. Press Ctrl-C to stop.
      write style.css
      write style.css.map
```



Aby zakończyć kompilację w terminalu użyj komendy **CTR C**.

Architektura plików

Jeśli tworzymy duży projekt, powinniśmy zachowywać czytelną i jasną strukturę plików. Za pomocą *SCSS* możemy stworzyć wiele małych plików, które będą oddzielnymi komponentami. Komponenty powinny być niezależne, mogą być użyte wielokrotnie w różnych miejscach projektu i powinny spełniać tylko jedno zadanie.

Istnieje podejście 7-1, które polega na tym, aby wszystkie pliki częściowe trzymać w 7 folderach + powinien istnieć jeden główny plik *main.scss*.

Do tego podejścia został stworzony *boilerplate*, który możesz pobrać stąd <https://github.com/HugoGiraudel/sass-boilerplate> bądź przynajmniej zaznajomić się z tym, jak wygląda struktura plików i na jakie komponenty jest podzielona.

W skrócie można powiedzieć, że struktura może wyglądać tak, jak na poniższym schemacie.

Mamy główny folder *scss*, w którym znajduje się główny plik *main.scss* i 7 folderów: *base*, *components*, *layout*, *pages*, *themes*, *utils*, *vendors*.

Każdy z tych folderów zawiera pliki, które dotyczą danej funkcjonalności z całego projektu.

- *base* – znajdują się wszystkie gotowe pliki, takie jak *reset/normalize*, reguły dotyczące typografii na stronie czy reguły dotyczące podstawowych stylów *CSS*.
- *components* – to folder dla mniejszych komponentów: przyciski, karuzele, miniatury.
- *layout* – w tym folderze znajdują się elementy, które służą do tego, aby розміścić dane elementy na stronie – aby stworzyć layout. Dlatego znajdują się tutaj takie elementy jak grid, header, footer, sidebar czy navigation.
- *pages* – tutaj umieszczamy style związane z konkretnymi podstronami naszego projektu.

- themes – folder jest wykorzystywany w przypadku dużych projektów i tworzenia osobnych motywów, dla różnych podstron/sekcji.
- utils – w tym miejscu trzymamy wszystkie pomocnicze funkcje *Sass*: zmienne, mixin, funkcje.
- vendors – tu się znajdują pliki gotowych bibliotek, jak np. pliki Bootstrapa, których nie modyfikujemy.

base

reset.scss / pliki normalizacyjne

typography.scss / cała typografia

component

buttons.scss / buttony

carousels.scss / karuzele

dropdown.scss / rozwijane menu

layout

navigation.scss / nawigacja

grid.scss / grid

header.scss / nagłówek

footer.scss / stopka

sidebar.scss / pasek boczny

forms.scss / formularze

pages

home.scss / strona główna

contact.scss / strona kontaktowa

themes

theme.scss

utils

variables.scss - zmienne

functions.scss - funkcje

mixins.scss - mixiny

vendors

bootstrap.scss

jquery.scss

Importowanie plików

W sytuacji, kiedy mamy stworzonych wiele plików *scss* wszystkie musimy je zaimportować do głównego pliku np. *main.scss*.

Aby zaimportować pliki należy je dołączyć za pomocą instrukcji *@import 'nazwapliku';*.

Należy pamiętać, aby w ścieżce uwzględnić folder, w którym dany plik się znajduje. Np.:

```
@import 'vendors/bootstrap';  
@import 'base/typography';  
@import 'layout/grid';
```

Następnie do dokumentu *html* dołączamy jedynie główny plik – *main.scss*.

Zmienne

Za pomocą zmiennych możemy na samym początku zdefiniować nasze właściwości i wartości.

Definiujemy zmienne od znaku *\$nazwazmienniej*: . Po *:* podajemy wartość zmiennej.

Możemy tak definiować kolory, fonty, wielkości (*width*, *height*, *font-size*, itp.) czy też reguły *media queries*.

Definiując zmienne np. w ten sposób:

```
// Variables

$mainColor: #cccccc;
$fontColor: #000000;
$blueColor: #7dc3cb;
$yellowColor: #d7b410;
```

możemy w późniejszej pracy nad kodem, bardzo łatwo odwoływać się do kolorów. Nie musisz pamiętać, że kod koloru żółtego to *#d7b410*. Wystarczy, że pisząc *color*: dodasz jego właściwość za pomocą nazwy, którą zdefiniowałeś w zmiennej. Poza tym w sytuacji, kiedy np. masz już skończony projekt, ale klient stwierdza, że jednak ten odcień żółtego to nie ten, nie musisz przeszukiwać całego kodu w poszukiwaniu koloru. Wystarczy, że zmienisz go w zdefiniowanych zmiennych, a on automatycznie zmieni się w każdym jego użyciu w kodzie.

```
a {
  color: $yellowColor;
}
```

media queries

Definiując w ten sposób zmienne, możemy w łatwy sposób używać *media queries*.

```
$small: "only screen and (max-width: 480px)";  
$medium: "only screen and (max-width: 768px)";  
$large: "only screen and (max-width: 1024px)";
```

Pisząc taki kod zmieniamy kolor tła w zależności od rozdzielczości ekranu:

```
body {  
  @media #{ $large } {  
    background: $yellowColor;  
  }  
  @media #{ $small } {  
    background: $blueColor;  
  }  
}
```

Kod po kompilacji będzie wyglądał tak:

```
@media only screen and (max-width: 1024px) {  
  body {  
    background: #d7b410;  
  }  
}  
  
@media only screen and (max-width: 480px) {  
  body {  
    background: #7dc3cb;  
  }  
}
```

Zagnieżdżanie

Zagnieżdżanie to nic innego, jak pisanie kodu „dziecka” wewnątrz kodu „rodzica”.

Tak wygląda kod napisany w *SCSS*. Wewnątrz *nav* znajduje się *ul*, *li* i *a* ze swoimi właściwościami.

```
nav {  
  color: $fontColor;  
  background-color: $mainColor;  
  
  ul {  
    margin: 0;  
    list-style: none;  
  }  
  
  li {  
    text-align: center;  
  }  
  
  a {  
    color: $yellowColor;  
  }  
}
```

A to jest ten sam kod napisany w *CSS*:

```
nav {  
  color: #000000;  
}  
  
nav ul {  
  margin: 0;  
  list-style: none;  
}  
  
nav ul li {  
  text-align: center;  
}
```

```
nav ul li a {  
  color: #d7b410;  
}
```

Oznacza to, że zamiast powtarzać ciągle ten sam selektor, możemy po prostu umieszczać wewnątrz niego poszczególne elementy.

Gdybyśmy chcieli umieścić np. *hover* w tagu *a* należy to zrobić za pomocą znaku *&*

```
a {  
  color: $yellowColor;  
  &:hover {  
    color: $blueColor;  
  }  
}
```

Ważne jest, aby w przypadku zagnieżdżania, pamiętać o odpowiednich wcięciach. Każdy kolejny poziom zagnieżdżenia to *1 tab*. Jeżeli chcemy dodać pseudoelement (*hover*, *active*, *after*, *before*, itd.) to robimy to za pomocą znaku *&* na tym samym poziomie co klasy elementu.

Mixins (domieszka)

Mixins to takie bardziej rozbudowane zmienne. To właściwie cały blok kodu, który raz definiujemy a następnie możemy go wielokrotnie używać za pomocą polecenia `@include`, nie przepisując go.

Aby stworzyć mixin używamy `@mixin` następnie wpisujemy jego *nazwę* oraz ewentualne *parametry* w nawiasie. Następnie tworzymy regułę i wywołujemy go w odpowiednim miejscu w kodzie projektu.

Mixin bez podanych parametrów nazywany jest mixin bezargumentowym.

```
@mixin overlay {  
  top: 0;  
  bottom: 0;  
  left: 0;  
  right: 0;  
  max-width: 100%;  
  position: absolute;  
}
```

Wywołując mixin używamy słowa `@include`.

Możemy też tworzyć mixin z parametrami. Wówczas po słowie `@mixin`, jego nazwie, dodajemy w nawiasach parametry. Parametry mogą być albo obowiązkowe albo opcjonalne. Różnią się one w zapisie tym, że parametrom opcjonalnym dodajemy wartość domyślną, którą możemy zmieniać już w momencie samego wykorzystywania mixin.

```
@mixin box($width, $height: $width) {  
  width: $width;  
  height: $height;  
}  
  
.box {  
  @include box(200px);  
}
```

W tym przypadku mamy parametr obowiązkowy *width*. Natomiast parametrem opcjonalnym jest *height*. Jeśli nie wpiszymy jego rozmiaru, przy wywołaniu mixin, jego rozmiar będzie domyślny, czyli taki jak rozmiar *width*.

Po kompilacji ten kod będzie wyglądał tak:

```
.box {  
  width: 200px;  
  height: 200px;  
}
```

W linku poniżej znajdziesz przykład dwóch mixins. Jeden jest bez parametrów – *centeredPosition*, drugi ma parametry opcjonalne – *opacity*.

Sprawdź co się zmieni, gdy wywołując *opacity* (*@include opacity*) dodasz nawiasy i wewnątrz nich ustawisz inną wartość niż domyślną (przypominam, że wartość *opacity* może być w przedziale 0-1 np. 0.2).

<https://codepen.io/BedeProgramistka/pen/XBwOLd>

W kodzie tym użyłam też wbudowanej funkcji w *SCSS* – *rgba*. O wbudowanych funkcjach przeczytasz niżej.

Przykład bardzo prostego mixin do tworzenia obramowania z wykorzystaniem parametru

Parametrem obowiązkowym w tym przypadku jest (*\$color*).

Tworzenie: *@mixin - naszaNazwa* (ewentualnie parametr obowiązkowy lub opcjonalny, w tym wypadku obowiązkowy) {nazwa reguły CSS: wartości - *1px solid \$color*}, gdzie pod *\$color* musimy wpisać nasz wybrany kolor, który ma mieć border.

```
@mixin border($color) {  
  border: 1px solid $color;  
}
```

I odwołujemy się do mixin w kodzie *SCSS*:

```
.icon-fa {  
  @include border($blueColor);  
}
```

Po kompilacji otrzymamy taki kod w *CSS*:

```
.icon-fa {  
  border: 1px solid #7dc3cb;  
}
```

Przykład mixin do centrowania pozycji bez parametru

```
@mixin centerPosition {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translateX(-50%) translateY(-50%);  
}
```

I jego użycie wewnątrz struktury *SCSS*:

```
.icon-head {  
  position: relative;  
}  
  
.icon-body {  
  @include centerPosition;  
}
```

Po kompilacji uzyskamy taki kod w *CSS*:

```
.icon-head {  
  position: relative;  
}  
  
.icon-body {  
  position: absolute;  
  top: 50%;  
  left: 50%;  
  transform: translateX(-50%) translateY(-50%);  
}
```

Przykład mixin do centrowania pozycji z parametrami

Możemy ten kod jeszcze bardziej rozbudować (przykład z <http://devcorner.pl>).

Tutaj nasz mixin musi przyjąć parametr – *vertical*, *horizontal* albo *both*. W zależności, jak parametr zastosujemy, taki kod zostanie skompilowany. W poniższym przykładzie używamy parametru *both*.

```
@mixin center($position) {  
  position: absolute;  
  @if $position=='vertical' {  
    top: 50%;  
  }
```

```

    -webkit-transform: translateY(-50%);
    -ms-transform: translateY(-50%);
    transform: translateY(-50%);
  }
  @else if $position=='horizontal' {
    left: 50%;
    -webkit-transform: translateX(-50%);
    -ms-transform: translateX(-50%);
    transform: translate(-50%);
  }
  @else if $position=='both' {
    top: 50%;
    left: 50%;
    -webkit-transform: translate(-50%, -50%);
    -ms-transform: translate(-50%, -50%);
    transform: translate(-50%, -50%);
  }
}

.parent {
  position: relative;
  .child {
    @include center(both);
  }
}

```

Po wywołaniu tak będzie wyglądał kod:

```

.parent {
  position: relative;
}

.parent .child {
  position: absolute;
  top: 50%;
  left: 50%;
  -webkit-transform: translate(-50%, -50%);
  -ms-transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
}

```

Przykład mixin do tworzenia transition z trzema opcjonalnymi parametrami

W tym przykładzie mamy trzy opcjonalne parametry.

```
@mixin transition($element: all, $time: .2s, $option: linear) {  
  -webkit-transition: $element $time $option;  
  transition: $element $time $option;  
}  
  
.box2 {  
  @include transition  
}
```

Jeśli przy wywołaniu tego mixin nie zmienimy wartości parametrów uzyskamy:

```
.box2 {  
  -webkit-transition: all 0.2s linear;  
  transition: all 0.2s linear;  
}
```

Przykład mixin do tworzenia gradientu z trzema parametrami

Innym fajnym mixin jest mixin do tworzenia gradientu (źródło: <https://devcorner.pl>)

```
@mixin background-gradient($start-color, $end-color, $orientation) {  
  background: $start-color;  
  @if $orientation=='vertical' {  
    background: -webkit-linear-gradient(top, $start-color, $end-color);  
    background: linear-gradient(to bottom, $start-color, $end-color);  
  }  
  @else if $orientation=='horizontal' {  
    background: -webkit-linear-gradient(left, $start-color, $end-color);  
  }  
}
```

```

    background: linear-gradient(to right, $start-color, $end-color);
  }
  @else {
    background: -webkit-radial-gradient(center, ellipse cover, $start-color, $end-color);
    background: radial-gradient(ellipse at center, $start-color, $end-color);
  }
}

.gradient {
  @include background-gradient(#3498db, #2c3e50, horizontal);
}

```

Przykład mixin do zmiany px na rem

W bardzo łatwy sposób możemy zamienić px na rem. Wystarczy, że użyjemy poniższego mixin i przy wywołaniu, w parametrze wpiszemy jednostki w px.

```

@mixin font-size($size, $base: 16) {
  font-size: $size; // fallback for old browsers
  font-size: ($size / $base) * 1rem;
}

p {
  @include font-size(12);
}

```

Po kompilacji uzyskamy taki kod:

```

p {
  font-size: 12;
  font-size: 0.75rem;
}

```


Przykład mixin do tworzenia strzałki o określonej wielkości (3 parametry)

Kolejny mixin pozwoli nam na stworzenie strzałek.

```
@mixin arrow($direction, $size, $color) {
  width: 0;
  height: 0;
  @if ($direction == left) {
    border-top: $size solid transparent;
    border-bottom: $size solid transparent;
    border-right: $size solid $color;
  }
  @else if ($direction == right) {
    border-top: $size solid transparent;
    border-bottom: $size solid transparent;
    border-left: $size solid $color;
  }
  @else if ($direction == down) {
    border-left: $size solid transparent;
    border-right: $size solid transparent;
    border-top: $size solid $color;
  }
  @else {
    border-left: $size solid transparent;
    border-right: $size solid transparent;
    border-bottom: $size solid $color;
  }
}

.arrow-black-left {
  @include arrow(left, 30px, #000000);
}
```

Po kompilacji uzyskamy taki kod:

```
.arrow-black-left {
  width: 0;
  height: 0;
  border-top: 30px solid transparent;
  border-bottom: 30px solid transparent;
  border-right: 30px solid #000;
```

```
}
```

Extend

Istnieje jeszcze coś takiego jak `extend`. `Extend` to takie trochę dziedziczenie parametrów. Gdy mamy jakąś klasę, której parametry chcemy użyć dla innej klasy i np. dopisać jakąś dodatkową właściwość możemy wykorzystać `extend`.

```
.icon {  
  color: $blueColor;  
  font-size: 2rem;  
  @include border($yellowColor);  
}  
  
.icon-2 {  
  @extend .icon;  
  padding: 0.2rem;  
}
```

Czyli w przypadku `.icon-2` kopiujemy niejako wszystkie parametry z klasy `.icon` – za pomocą `@extend` i dodajemy padding.

Nie musimy też dodawać innej właściwości. Możemy tylko skopiować właściwości za pomocą `@extend`.

Kod po kompilacji będzie wyglądał tak:

```
.icon, .icon-2 {  
  color: #7dc3cb;  
  font-size: 2rem;  
  border: 1px solid #d7b410;  
}  
  
.icon-2 {  
  padding: 0.2rem;  
}
```

Wbudowane funkcje w SCSS

Scss posiada wbudowane funkcje, za pomocą których możemy manipulować elementami na stronach, bez tworzenia nowych mixin.

Wszystkie wbudowane funkcje znajdziesz tu:

<http://sass-lang.com/documentation/Sass/Script/Functions.html>

Ja przygotowałam dla Ciebie małą ściągawkę z funkcji służących do manipulowania kolorami:

<https://codepen.io/BedeProgramistka/pen/RBzKbQ>

SCSS i Bootstrap

Gdy tworzymy projekt za pomocą stylów Bootstrapa istnieje czasem potrzeba ich nadpisania.

Dobrym podejściem jest praca na plikach Bootstrapa oraz na własnych plikach `.scss`, które są później kompilowane do jednego wyjściowego pliku `.css`. Należy przy tym pamiętać, żeby zmiany wprowadzać jedynie we własnych plikach. Plików Bootstrapa nie powinniśmy modyfikować.

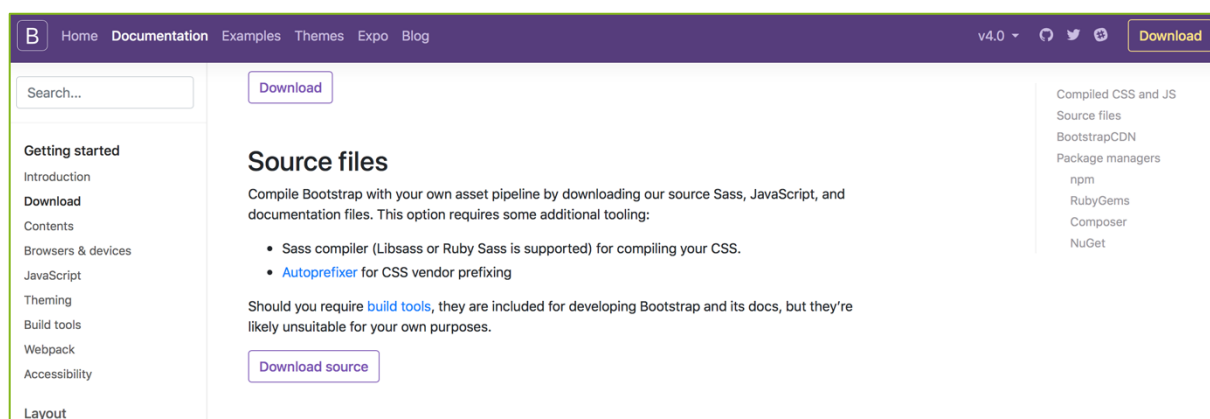
Jeśli chodzi o kolejność ładowania plików, to najpierw ładujemy pliki Bootstrapa, a dopiero później nasze.

Pobieranie plików z Bootstrapa

Aby pobrać pliki `SCSS` należy wejść na stronę Bootstrapa

<https://getbootstrap.com/docs/4.0/getting-started/download/>.

Wybierz wersję, na której będziesz pracować (w naszym przypadku jest to wersja 4.0), a następnie przejdź do zakładki *Source files*.

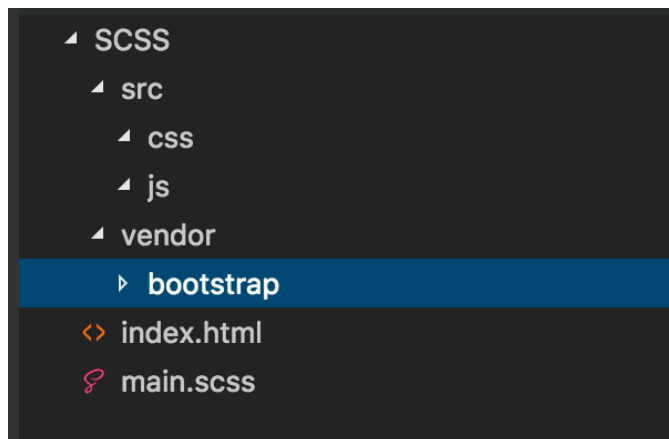


Kliknij na *download source* i pobierz paczkę plików.

Następnie musisz ją rozpakować.

W folderze z Twoim projektem utwórz dwa pliki – *index.html* i *main.scss*.

Do folderu z projektem wrzuć cały katalog *scss* (ten, który rozpakowałeś z pobranego pliku). Stwórz też katalog/katalogi dla stylów, które będziesz tworzyć. Nie zapomnij też o folderze *src/css*, w którym zostanie utworzony, w późniejszym etapie skompilowany plik *style.css*.

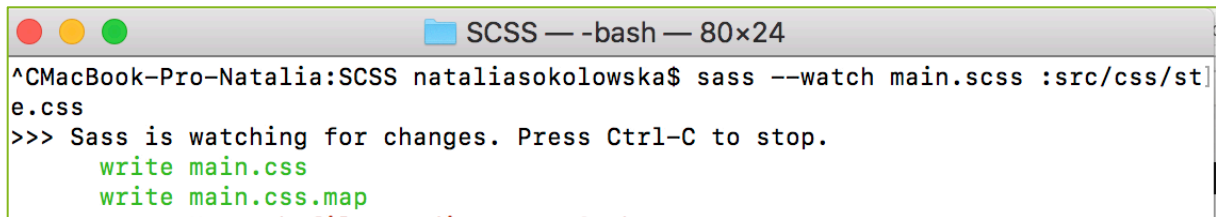


W folderze z plikami *scss* Bootstrapa znajduje się masa plików. Ale jest tam też plik *bootstrap.scss* i to on jest plikiem głównym, w którym są zaimportowane wszystkie pozostałe style *scss*. Dlatego, żeby mieć dostęp do plików Bootstrapa w Twoim projekcie musisz do głównego pliku *main.scss* zaimportować plik *bootstrap.scss*, a następnie do struktury podpiąć plik *style.css*.

```
@import 'vendor/bootstrap/bootstrap';
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Scss Bootstrap</title>
  <link rel="stylesheet" href="src/css/style.css">
</head>
```

Na koniec nie zostało nic innego, jak zacząć śledzić zmiany w pliku `scss` za pomocą terminala i komendy `sass --watch main.scss :src/css/style.css`.

A terminal window titled "SCSS — -bash — 80x24" with standard macOS window controls (red, yellow, green buttons). The terminal shows the command `sass --watch main.scss :src/css/style.css` being executed. The output indicates that Sass is watching for changes and provides instructions to press Ctrl-C to stop. It also shows two file write events: `write main.css` and `write main.css.map`.

```
^CMacBook-Pro-Natalia:SCSS nataliasokolowska$ sass --watch main.scss :src/css/style.css
e.css
>>> Sass is watching for changes. Press Ctrl-C to stop.
      write main.css
      write main.css.map
```

Uruchomienie śledzenia utworzy Twój skompilowany plik `style.css`.