

# Kurs HTML5 i CSS3

Autor: Michał Kajpust

# Część X


SASS

# SASS:

Z powodu tworzenia coraz większych i coraz bardziej skomplikowanych arkuszy stylów coraz trudniej jest się w nich odnaleźć. W takiej sytuacji z pomocą przychodzą preprocesory CSS służące do wzbogacania języka CSS o niektóre mechanizmy znane z tradycyjnych języków programowania, które nie są dostępne w oficjalnym standardzie CSS3. Należą do nich m.in.: zmienne, funkcje, zagnieżdżanie, mixiny czy dziedziczenie dzięki, którym można znacząco zwiększyć efektywności podczas tworzenia stron internetowych. Zadaniem preprocesora jest przetworzenie nowej składni i przekompilowanie napisanego kodu do formatu zgodnego ze standardem CSS obsługiwanym przez przeglądarki. Po przekompilowaniu otrzymujemy natywny kod, który zapewnia zgodność ze wszystkimi znanymi przeglądarkami.

# SASS:

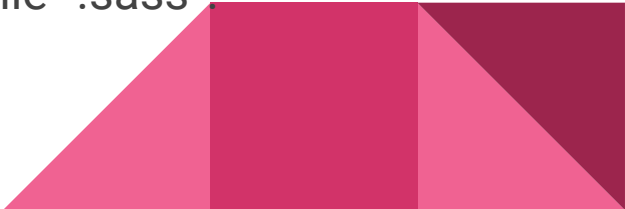
**Sass** (z ang. *Syntactically Awesome StyleSheets*) to najbardziej dojrzały i popularny ze wszystkich dotychczasowych preprocessorów CSS, który swoimi początkami sięga 2006 roku. Innymi graczami na rynku są np. Less (z ang. *Leaner CSS*) bądź *Stylus*. Dzięki Sass'owi można dobrze zorganizować duże arkusze stylów i szybko uruchomić małe. Niektóre zalety stosowania Sass:

- pełna kompatybilność z CSS;
  - rozszerzenia językowe takie jak zmienne, mixiny czy zagnieżdżanie;
  - wiele przydatnych funkcji do manipulowania kolorami i innymi wartościami;
  - dobrze sformatowane, konfigurowalne wyjście.
- 

# SASS:


Istnieją dwie składnie dostępne dla Sass, czyli: sass i scss, które różnią się między sobą sposobem formatowania kodu.

Składnia sass jest starsza i znana również jako “wcięta składnia”. Zapewnia bardziej zwężły sposób pisania CSS. Zamiast nawiasów klamrowych do definiowania bloków używa wcięć a nie nawiasów klamrowych, natomiast reguły oddziela od siebie znakiem nowej linii, a nie średnikiem jak ma to miejsce w klasycznym kodzie CSS. Dla niektórych będzie to łatwiejsze i szybsze podejście do pisania SCSS. Pliki używające tej składni mają rozszerzenie “.sass”



# SASS:

Składnia scss natomiast (z ang. *Sassy CSS*) jest zbliżona w założeniu do klasycznego kodu CSS będąc jednocześnie jego rozszerzeniem. Oznacza to, że każdy poprawny arkusz stylów CSS jest prawidłowym plikiem SCSS o tym samym znaczeniu (jednak nie odwrotnie). Ponadto SCSS rozumie większość *hacków* CSS i składni specyficznej dla określonych programów, jak np. składnia dla starszych wersji IE. Na przykład do definiowania bloków kodu wykorzystuje nawiasy klamrowe, a średniki do oddzielania poszczególnych reguł. Pliki używające tej składni mają rozszerzenie `".scss"`.



# SASS:

Każda składnia może importować pliki zapisane w drugiej. Pliki takie można automatycznie konwertować z jednej składni na drugą za pomocą wiersza poleceń i komendy *sass-convert*:

```
# Convert Sass to SCSS
$ sass-convert style.sass style.scss

# Convert SCSS to Sass
$ sass-convert style.scss style.sass
```

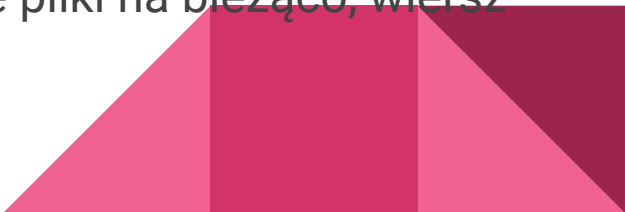
Należy jednak wiedzieć, że to polecenie nie generuje pliku CSS. W tym celu należy użyć innej komendy.

# SASS:

Sass może być używany na trzy sposoby:

- jako narzędzie wiersza poleceń;
- jako samodzielny moduł Ruby;
- jako wtyczka dla dowolnej platformy obsługującej Rack (w tym Ruby on Rails i Merb).

Pierwszym krokiem do korzystania z modułu Sass jest oczywiście jego zainstalowanie. Można tego dokonać za pomocą przynajmniej kilku metod w zależności od używanego systemu operacyjnego wykorzystując do tego np.: gotowe aplikacje i programy śledzące czy przetwarzające pliki na bieżąco, wiersz poleceń, pakiet npm, Chocolatey, Homebrew, Ruby i inne.





# SASS:

Po zainstalowaniu Sass'a można korzystać z listy gotowych komend wywoływanych bezpośrednio w wierszu poleceń. Do najważniejszych i najczęściej używanych należą:

- **sass input.scss output.css** - Sass jednorazowo przetwarza plik z rozszerzeniem .scss na plik .css;
- **sass --watch input.scss:output.css** - Sass na bieżąco obserwuje plik z rozszerzeniem .scss i jeśli pojawi się w nim jakaś zmiana to całość przetwarza i zapisuje do pliku .css;
- **sass --watch input-dir:output-dir** - w tym przypadku Sass będzie obserwował wszystkie pliki z rozszerzeniem .scss znajdujące się we wskazanym folderze i automatycznie kompilował je do plików znajdujących się w folderze zawierającym pliki .css;
- **sass --help** - powoduje wyświetlenie pełnej dokumentacji i listy komend wraz z ich tłumaczeniem;

# SASS:

Dodatkowo można używać kilku dodatkowych parametrów:

- **--style** nested (domyślny styl, w którym każda właściwość ma własną linię, bez stałego wcięcia - każda reguła jest wcięta na podstawie tego jak głęboko jest zagnieżdżona)/expanded (każda właściwość i reguła zajmuje jedną linię, właściwości są wcięte w ramach reguł ale reguły nie są wcięte w żaden szczególny sposób)/compact (zajmuje mniej miejsca niż styl *nested* i *expanded*, ponieważ każda reguła zajmuje tylko jeden wiersz)/compressed (zajmuje minimalną ilość miejsca bez białych znaków);
- **--no-cache** - wyłączenie buforowania plików .scss przyspieszającego kompilowanie (domyślnie włączone);
- **--update** - kompiluje pliki i foldery do .css (z dodatkowym parametrem **--force** zmienia każdy plik do .css nawet jeśli jest starszy od pliku wynikowego);
- **--scss** - kompilator użyje rozszerzenia pliku do określenia używanej składni;

# SASS:

- **--sourcemap** kontrolujący sposób generowania *sourcemap*. *Sourcemap*'y informują przeglądarkę jak odnaleźć style Sass, które spowodowały wygenerowanie stylów CSS. Przyjmuje 3 dodatkowe parametry: auto (używa względnych identyfikatorów URI, zakładając, że źródłowe arkusze stylów będą dostępne na dowolnym serwerze, który jest używany i, że ich względna lokalizacja będzie taka sama jak w lokalnym systemie plików)/inline (obejmuje pełny tekst źródłowy w *sourcemap*'ach, który jest maksymalnie przenośny - ale może przez to tworzyć bardzo duże pliki źródłowe)/none (nie powoduje tworzenia *sourcemap*).




# SASS - zmienne:

**Zmienne w Sass** podobnie jak w innych językach programowania służą do przechowywania informacji, które można następnie wykorzystać w innym miejscu arkusza stylów. Można za ich pomocą przechowywać takie rzeczy jak kolory, stopy czcionek lub dowolną wartość CSS. Sass umożliwia deklarowanie dwóch rodzajów zmiennych: takich z których można korzystać jedynie na poziomie zagnieżdżonych selektorów (w których są zdefiniowane) bądź takich, które są dostępne w całym arkuszu stylów (zdefiniowane poza zagnieżdżonymi selektorami lub z flagą “!global”). Aby utworzyć zmienną w Sass’ie należy jej nazwę poprzedzić znakiem “\$” a następnie użyć “:” i podać żądaną wartość CSS. Wszystkie nazwy zmiennych i identyfikatorów w Sass mogą używać myślników i podkreśleń zamiennie.

# SASS - zmienne:

## Typy zmiennych w Sass'ie:

- **liczby** (0, 1.2, 10px);
  - **ciągi znaków** - w cudzysłowie lub bez ('Helvetica', linear);
  - **kolory** (blue, #fff, rgb(0,0,0));
  - **wartości logiczne** (true, false);
  - **null**;
  - **listy wartości** oddzielone przecinkami (1.5em 1em 0 2em, Helvetica, Arial, sans-serif);
  - **mapy wartości** (key1: value1, key2: value2);
  - **funkcje**;
  - **wszystkie inne typy wartości** właściwości CSS traktowane jako ciągi znaków bez cudzysłowia.
- 

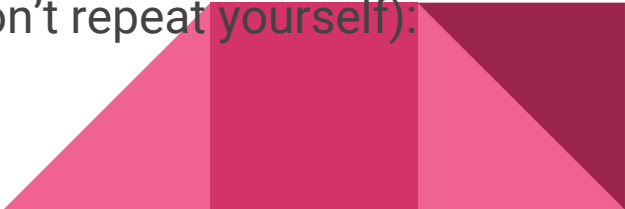
# SASS - zmienne:

```
$font-stack: Helvetica, sans-serif;  
$primary-color: #333;  
$primary-size: 48px;  
  
body {  
  font-family: $font-stack;  
  font-size: $primary-size;  
  color: $primary-color;  
}
```

```
body {  
  font-family: Helvetica, sans-serif;  
  font-size: 48px;  
  color: #333;  
}
```

# SASS - zagnieżdżanie:

Sass pozwala na **zagnieżdżanie selektorów** w sposób zbliżony do tego używanego w dokumencie HTML (należy jednak pamiętać, że nadmiernie zagnieżdżane reguły powodują komplikacje w CSS). Jest to świetny sposób na uporządkowanie kodu CSS i uczynienie go bardziej czytelnym. Ponadto Sass pozwala na wzajemne zagnieżdżanie się reguł - wewnętrzna reguła ma zastosowanie tylko w selektorze zewnętrznej reguły. Pomaga to unikać powtarzania selektorów macierzystych i znacznie upraszcza złożone układy CSS z dużą ilością zagnieżdżonych selektorów. Dzięki temu można dodatkowo trzymać się jednej z zasad pisania stylów CSS pod nazwą DRY (don't repeat yourself):



# SASS - zagnieżdżanie:

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
    li {  
      display: inline-block;  
      a {  
        display: block;  
        padding: 6px 12px;  
        text-decoration: none;  
      }  
    }  
  }  
}
```

```
nav ul {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
nav ul li {  
  display: inline-block;  
}  
nav ul li a {  
  display: block;  
  padding: 6px 12px;  
  text-decoration: none;  
}
```



# SASS - zagnieżdżanie:

W celu skonstruowania reguły pozwalającej na sformatowanie jakiegoś elementu zagnieżdżonego w inny sposób niż domyślny należy użyć znaku "&". Pozwala on odnieść się do elementu rodzica np. w sytuacji gdy posiada on określoną klasę lub w celu nadania określonego stanu (pseudoklasy). W takich sytuacjach za pomocą znaku "&" określa się gdzie selektor rodzica powinien zostać wstawiony. Podczas kompilacji kodu do CSS znak "&" zostanie zastąpiony selektorem nadrzędnym. Jedynym warunkiem takiej konstrukcji jest umieszczenie znaku "&" na początku selektora złożonego (z opcją poprzedzenia go odpowiednim sufixem). Podczas zagnieżdżania można również normalnie stosować selektory potomka (">", "+" i inne), pseudoklas (":"), pseudoelementów ("::") czy atrybutów ("[title='tytuł']"):

# SASS - zagnieżdżanie:

```
li {  
  display: inline-block;  
  &.lead {  
    background-color: red;  
  }  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
    &:hover {  
      text-decoration: underline;  
    }  
  }  
}
```

```
li {  
  display: inline-block;  
}  
li.lead {  
  background-color: red;  
}  
li a {  
  display: block;  
  padding: 6px 12px;  
  text-decoration: none;  
}  
li a:hover {  
  text-decoration: underline;  
}
```

# SASS - zagnieżdżanie:

W CSS jest wiele właściwości posiadających tą samą rodzinę nazw jak np. *font-family*, *font-size* i *font-weight*. Aby użyć ich wszystkich trzeba każdorazowo wpisywać konkretną nazwę co może niepotrzebnie wpływać na wydłużenie pisania kodu i jego poziom skomplikowania. Sass natomiast zapewnia łatwiejszy i bardziej intuicyjny sposób zapisywania konkretnej rodziny nazw polegający na jednorazowym napisaniu rodziny a następnie zagnieżdżeniu każdej właściwości podrzędnej:



## SASS - zagnieżdżanie:

```
.funky {  
  font: {  
    family: fantasy;  
    size: 30em;  
    weight: bold;  
  }  
}
```

```
.funky {  
  font-family: fantasy;  
  font-size: 30em;  
  font-weight: bold;  
}
```



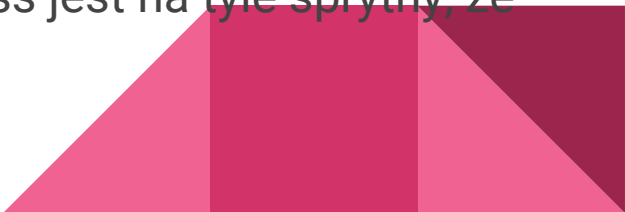
# SASS - dyrektywy:

Sass obsługuje wszystkie reguły CSS rozpoczynające się od znaku “@”, a także kilka dodatkowych zarezerwowanych tylko dla Sass zwanych **dyrektywami**. Można np. tworzyć częściowe pliki Sass (*partials*) zawierające małe fragmenty kodu importowane do innych plików Sass. Dzięki czemu można lepiej uporządkować kod i łatwiej go rozwijać w przyszłości. Pliki składowe to nic innego jak pliki z rozszerzeniem .scss (bądź .sass), których nazwa powinna rozpoczynać się od znaku podkreślenia (aby poinformować kompilator o tym, że jest to plik częściowy i nie powinien być od razu generowany do pliku .css). Następnie pliki takie można importować przy użyciu dyrektywy **@import** bez użycia podkreślenia i rozszerzenia. Wszystkie zaimportowane pliki .scss (lub .sass) zostaną połączone w jeden plik wyjściowy .css.

# SASS - dyrektywy:

Chociaż najlepiej posiadać dyrektywy *@import* na najwyższym poziomie dokumentu, to możliwe jest również uwzględnianie ich w regułach CSS i *@media*.

Reguły *@import* stosowane w Sass mają jeszcze jedną przewagę nad tymi wykorzystywanymi w CSS. Mianowicie używanie *@import* w CSS generuje kolejne żądania HTTP, natomiast Sass pobiera plik, który chce zaimportować i łączy go z docelowym plikiem (zwraca zawsze jeden plik), w taki sposób aby nie generować kolejnego żądania do serwera. Podczas importowania pliku nie trzeba uwzględniać rozszerzenia *.scss* (lub *.sass*), ponieważ Sass jest na tyle sprytny, że rozpoznaje typ dokumentu i odpowiednio go podłącza.



# SASS - dyrektywy:

```
p {  
  margin: 0;  
  padding: 0;  
  font-style: italic;  
  font-weight: bold;  
  font-size: 24px;  
  color: green;  
  background-color: #fff;  
}
```

```
@import 'base/reset';  
  
div {  
  @import 'base/reset';  
}
```

```
p {  
  margin: 0;  
  padding: 0;  
  font-style: italic;  
  font-weight: bold;  
  font-size: 24px;  
  color: green;  
  background-color: #fff;  
}
```

```
div p {  
  margin: 0;  
  padding: 0;  
  font-style: italic;  
  font-weight: bold;  
  font-size: 24px;  
  color: green;  
  background-color: #fff;  
}
```


# SASS - dyrektywy:

```
@import "functions";  
@import "variables";  
@import "mixins";  
@import "root";  
@import "reboot";  
@import "type";  
@import "images";  
@import "code";  
@import "grid";  
@import "tables";  
@import "forms";  
@import "buttons";  
@import "transitions";  
@import "dropdown";  
@import "button-group";  
@import "input-group";  
@import "custom-forms";  
@import "nav";  
@import "navbar";
```



# SASS - dyrektywy:

Dyrektywy **@media** w Sass'ie zachowują się tak samo jak w zwykłym CSS, z jedną dodatkową opcją - mogą być zagnieżdżone w regułach CSS. Jeśli w regule CSS pojawi się dyrektywa *@media* to zostanie ona przeniesiona (bąbelkując) do najwyższego poziomu arkusza stylów, umieszczając wszystkie selektory na drodze wewnątrz reguły. Ułatwia to znacząco dodawanie stylów związanych z nośnikami bez konieczności powtarzania selektorów lub przerywania przepływu arkusza stylów. Dodatkowo *@media queries* mogą być zagnieżdżane w sobie. Takie zapytania zostaną następnie połączone w jedno za pomocą operatora "and". Możliwe jest również wykorzystywanie przez *@media queries* funkcji i operatorów zamiast nazw funkcji i wartości funkcji.



# SASS - dyrektywy:

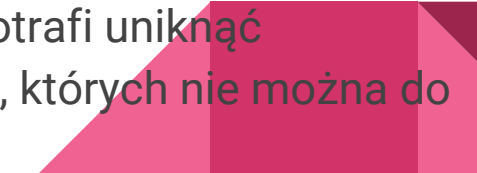
```
.sidebar {  
  width: 300px;  
  @media screen and (orientation: landscape) {  
    width: 500px;  
  }  
}  
  
@media screen {  
  .sidebar {  
    @media (orientation: landscape) {  
      width: 500px;  
    }  
  }  
}
```

```
.sidebar {  
  width: 300px;  
}  
  
@media screen and (orientation: landscape) {  
  .sidebar {  
    width: 500px;  
  }  
}  
  
@media screen and (orientation: landscape) {  
  .sidebar {  
    width: 500px;  
  }  
}
```

# SASS - dyrektywy:

Dyrektywa **@extend** jest jedną z najbardziej przydatnych funkcji w Sass, ponieważ pozwala udostępniać zestaw właściwości CSS z jednego selektora na drugi (jeden selektor dziedziczy style innego). Dzięki czemu w kodzie Sass dużo łatwiej zapanować nad zbędnym powtarzaniem selektorów (ponownie zasada DRY). Pomaga ona również unikać konieczności pisania wielu nazw klas dla elementów HTML (każdy element z użytą deklaracją *@extend* przyjmuje klasy podane po tej dyrektywie).

*@extend* działa poprzez wstawienie selektora rozszerzającego w dowolnym miejscu arkusza stylów, w którym pojawia się rozszerzony selektor. Podczas łączenia selektorów dyrektywa *@extend* jest na tyle inteligentna, że potrafi uniknąć niepotrzebnego duplikowania kodu i nie generuje selektorów, których nie można do niczego dopasować.



# SASS - dyrektywy:

```
.message {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
.  
success {  
  @extend .message;  
  border-color: green;  
}  
.  
error {  
  @extend .message;  
  border-color: red;  
}  
.  
warning {  
  @extend .message;  
  border-color: yellow;  
}
```

```
.message, .success, .error, .warning {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
.  
success {  
  border-color: green;  
}  
.  
error {  
  border-color: red;  
}  
.  
warning {  
  border-color: yellow;  
}
```

# SASS - dyrektywy:

Ponadto pojedynczy selektor może przyjmować więcej niż tylko jeden selektor rozszerzający (przy użyciu dyrektywy *@extend*).

Kilka rozszerzeń można również zapisać przy użyciu listy selektorów oddzielonych od siebie przecinkami. Np:

```
div {  
  @extend .error, .alert;  
}
```

```
div {  
  @extend .error;  
  @extend .alert;  
}
```

# SASS - dyrektywy:

```
.hoverlink {  
  @extend a:hover;  
}  
a:hover {  
  text-decoration: underline;  
}
```

```
a:hover, .hoverlink {  
  text-decoration: underline;  
}
```

```
p {  
  color: red;  
}  
div {  
  border: 1px solid green;  
  @extend p;  
}
```

```
p, div {  
  color: red;  
}  
  
div {  
  border: 1px solid green;  
}
```

# SASS - dyrektywy:

```
.error {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
.attention {  
  font-size: 3em;  
  background-color: #ff0;  
}  
.seriousError {  
  @extend .error;  
  @extend .attention;  
  border-width: 3px;  
}
```

```
.error, .seriousError {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
.attention, .seriousError {  
  font-size: 3em;  
  background-color: #ff0;  
}  
.seriousError {  
  border-width: 3px;  
}
```

# SASS - dyrektywy:

```
.error {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
.seriousError {  
  @extend .error;  
  border-width: 3px;  
}  
.criticalError {  
  @extend .seriousError;  
  position: fixed;  
  top: 10%;  
  bottom: 10%;  
  left: 10%;  
  right: 10%;  
}
```

```
.error, .seriousError, .criticalError {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
.seriousError, .criticalError {  
  border-width: 3px;  
}  
.criticalError {  
  position: fixed;  
  top: 10%;  
  bottom: 10%;  
  left: 10%;  
  right: 10%;  
}
```



# SASS - dyrektywy:

Istnieją również pewne ograniczenia dotyczące rozszerzania selektorów dotyczące np. reguł *@media*. Ponieważ Sass nie potrafi tworzyć reguł CSS poza blokiem *@media* aby zastosować selektory wewnątrz niego bez tworzenia ogromnej liczby nadpisań stylów poprzez kopiowanie w każdym miejscu. Oznacza to, że aby móc używać *@extend* wewnątrz dyrektywy *@media* (lub każdej innej) należy rozszerzać selektory tylko o te, które pojawiają się w tym samym bloku deklaracji:



# SASS - dyrektywy:

```
@media screen {  
  .error {  
    border: 1px #f00;  
    background-color: #fdd;  
  }  
  .seriousError {  
    @extend .error;  
    border-width: 3px;  
  }  
}
```


```
.error {  
  border: 1px #f00;  
  background-color: #fdd;  
}  
@media screen {  
  .seriousError {  
    @extend .error;  
    border-width: 3px;  
  }  
}
```

```
@media screen {  
  .error, .seriousError {  
    border: 1px #f00;  
    background-color: #fdd;  
  }  
  .seriousError {  
    border-width: 3px;  
  }  
}
```

```
/*  
Sass::SyntaxError: You may not @extend an outer selector from within @media.  
You may only @extend selectors within the same directive.  
From "@extend .error" on line 8 of scss/style.scss.  
*/
```

# SASS - dyrektywy:


Aby uniknąć żmudnego powtarzania kodu szczególnie w przypadku używania prefixów dla nowych właściwości CSS, moduł Sass pozwala tworzyć grupy deklaracji CSS, które można ponownie wykorzystywać - **@mixin** bo o nim mowa może również przyjmować dodatkowe parametry. Aby utworzyć nowego mixina należy użyć dyrektywy *@mixin* i nadać mu nazwę oraz opcjonalne argumenty i blok deklaracji. Po jego utworzeniu można go użyć w kodzie Sass zaczynając od słowa *@include*, po którym następuje nazwa mixina. Ze względów historycznych podobnie jak zmienne również mixiny mogą używać w swoich nazwach myślników i podkreśleń zamiennie bez wpływu na ich poprawne działanie. Definicje *@mixinów* mogą również zawierać inne mixiny:



# SASS - dyrektywy:

Mixiny mogą być zawarte poza dowolną regułą (o ile nie definiują bezpośrednio żadnych właściwości) i obejmować same siebie (nowość od wersji 3.3). Aby określać wartości domyślne dla argumentów, należy używać normalnej składni jak podczas definiowania zmiennej. Podczas użycia takiego mixina w przypadku nie podania jego argumentu zostanie użyta podana wcześniej wartość domyślna. Definiując mixiny argumenty są zapisywane wewnątrz nawiasu jako nazwy zmiennych oddzielone od siebie przecinkami. Nazwane argumenty mogą być przekazywane w dowolnej kolejności, a argumenty z wartościami domyślnymi można pomijać.

Mixiny, które definiują tylko wybrane selektory, można bez przeszkód umieszczać na najwyższym poziomie dokumentu.



# SASS - dyrektywy:

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}  
  
.box {  
  @include border-radius(10px);  
}
```

```
.box {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  -ms-border-radius: 10px;  
  border-radius: 10px;  
}
```



# SASS - dyrektywy:

```
@mixin border($color, $width) {  
  border: {  
    color: $color;  
    width: $width;  
    style: dashed;  
  }  
}  
  
p {  
  @include border(blue, 1rem);  
}
```

```
p {  
  border-color: blue;  
  border-width: 1rem;  
  border-style: dashed;  
}
```



# SASS - dyrektywy:


```
@mixin border($color, $width: 10px) {  
  border: {  
    color: $color;  
    width: $width;  
    style: dashed;  
  }  
}  
  
p {  
  @include border(blue);  
}  
  
h1 {  
  @include border(blue, 20px);  
}
```

```
p {  
  border-color: blue;  
  border-width: 10px;  
  border-style: dashed;  
}  
  
h1 {  
  border-color: blue;  
  border-width: 20px;  
  border-style: dashed;  
}
```

# SASS - operatory:

Sass obsługuje standardowe operacje arytmetyczne na liczbach: dodawanie "+", odejmowanie "-", mnożenie "\*", dzielenie "/" i moduł "%". Funkcje matematyczne zachowują jednostki podczas operacji arytmetycznych, dlatego nie można pracować z liczbami posiadającymi niekompatybilne jednostki (np. px + em) lub wykonywać mnożenia liczb z tą samą jednostką (jednostki kwadratowe nie są prawidłowymi jednostkami CSS).

Dodatkowo Sass obsługuje operatory porównania: "<", ">", "<=", ">=" (jedynie dla liczb) i operatory porównania "==", "!=" dla wszystkich typów jednostek.





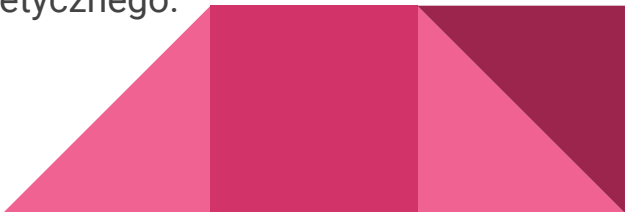
# SASS - operatory:

```
.container {  
  width: 100%;  
}  
article[role="main"] {  
  float: left;  
  width: 600px / 960px * 100%;  
}  
aside[role="complementary"] {  
  float: right;  
  width: 300px / 960px * 100%;  
}
```

```
.container {  
  width: 100%;  
}  
article[role="main"] {  
  float: left;  
  width: 62.5%;  
}  
aside[role="complementary"] {  
  float: right;  
  width: 31.25%;  
}
```

# SASS - operatory:

W przypadku dzielenia “/” sprawa jest nieco bardziej skomplikowana, ponieważ może się zdarzyć, że jakieś wartości zostaną uznane nie za działanie matematyczne ale za sposób przedstawienia wartości zmiennej. Co do poniższych można być jednak spokojnym:

- jeśli wartość lub jakakolwiek jej część jest przechowywana w zmiennej lub zwracana przez funkcję;
  - jeśli wartość jest otoczona nawiasami (chyba, że nawiasy znajdują się poza listą a wartość znajduje się w środku);
  - jeśli wartość jest używana jako część innego wyrażenia arytmetycznego.
- 

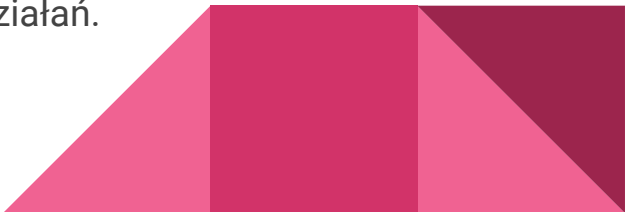
## SASS - operatory:

```
p {  
  font: 10px/8px;  
  $width: 1000px;  
  width: $width/2;  
  width: round(1.5)/2;  
  height: (500px/2);  
  margin-left: 5px + 8px/2px;  
  font: (italic bold 10px/8px);  
}
```

```
p {  
  font: 10px/8px;  
  width: 500px;  
  width: 1;  
  height: 250px;  
  margin-left: 9px;  
  font: italic bold 10px/8px;  
}
```

# SASS - operatory:

W związku z używaniem operatora odejmowania “-” podobnie jak w przypadku dzielenia “/” często zdarza się, że może on zostać niewłaściwie zinterpretowany. Do wyboru jest kilka opcji: jako operator odejmowania, określenie liczby ujemnej, operator negacji lub część identyfikatora. Aby zminimalizować ryzyko błędu należy trzymać się kilku zasad:

- w przypadku operatora odejmowania dodawać spacje po obu stronach znaku “-”;
  - w przypadku liczby ujemnej uwzględniać spację przed ale nie po znaku “-”;
  - w przypadku negacji okalać elementy wewnątrz nawiasów;
  - używać nawiasów w celu określenia prawidłowej kolejności działań.
- 

# SASS - komentarze:

Sass obsługuje standardowe wielowierszowe komentarze z CSS (rozpoczynające się od znaku `/*` i kończące się `*/`), a także komentarze jednowierszowe `//` (znane np z JavaScript). Co ważne to, komentarze wieloliniowe podczas kompilowania kodu są zachowywane w pliku wynikowym CSS, podczas gdy komentarze jednoliniowe są usuwane.

Podczas tworzenia komentarzy w SCSS możliwa jest również interpolacja zmiennych (ich konwertowanie na przypisane wartości).



## SASS - komentarze:

```
/*  
Komentarz umieszczony  
w  
wielu liniach  
kodu  
*/  
  
//Komentarz jednoliniowy
```

```
/*  
Komentarz umieszczony  
w  
wielu liniach  
kodu  
*/
```

# Ćwiczenia:

1. Stwórz hierarchię katalogów, która będzie najlepsza pod względem używania modułu Sass.
2. Zainstaluj moduł Sass na swoim komputerze wykorzystując którąś z dostępnych metod.
3. Sprawdź za pomocą konsoli poprawność zainstalowania modułu i jego wersję oraz wyświetl ekran pomocy.
4. Następnie utwórz plik index.html zawierający standardową strukturę i plik style.scss do którego będą importowane arkusze składowe.
5. Stwórz plik \_base.scss zawierający podstawowe reguły resetujące ustawienia przeglądarki i dołącz go do pliku style.scss we właściwy sposób.
6. Uruchom kompilator kodu i dołącz plik .css do dokumentu HTML.

# Ćwiczenia:

7. Następnie uruchom “strażnika” kodu scss aby czuwał nad zmianami dokonanymi w plikach .scss wewnątrz jednego folderu.
8. Stwórz dodatkowe pliki `_variables.scss` i `_typography.scss` w których umieść kilka zmiennych (kolory, tła, grubości, style) i klas służących do formatowania tekstu.
9. Następnie wewnątrz dokumentu HTML stwórz przykładowy nagłówek, akapit i kontener z umieszczonym wewnątrz mniejszym pudełkiem i sformatuj je za pomocą stworzonych reguł wykorzystując dyrektywę `@extend`.
10. Utwórz `@mixin` odpowiedzialnego za zmianę koloru placeholdera, zmieniającego sposób wyświetlania elementów na `display: flex`; (z właściwymi prefixami) i ustawiającego zaokrąglenie rogów.



# Ćwiczenia:

11. Dodaj do dokumentu HTML element input zawierający atrybut *placeholder* i dodaj mu stworzonego mixina.
12. Utwórz prostą nawigację przy użyciu listy nienumerowanej której linki podczas efektu hover będą zmieniały kolor i tło a do zaimplementowania przejścia wykorzystaj kolejnego *@mixin'a*.
13. Pobierz wersję źródłową bootstrapa4 ze strony internetowej i dołącz pliki .scss do stylu (sprawdź efekt w arkuszu .css) oraz skrypty do dokumentu HTML.
14. Odnajdź w dokumentacji bootstrapa4 moduł odpowiedzialny za nawigację i dołącz ją do dokumentu.
15. Sprawdź czy działa poprawnie w przeglądarce.