

# Sass

v3.1

# Plan

1. Wstęp
2. Instalacja i uruchomienie Sassa
3. Formatowanie stylów wyjściowych
4. Mapowanie kodu źródłowego
5. Komentarze
6. Zmienne
7. Zagnieżdżanie właściwości i selektorów
8. Partials i importowanie
9. Mixins
10. Dziedziczenie
11. Listy i Mapy
12. Operatory i operacje
13. Interpolacja
14. Logika
15. Funkcje
16. Dlaczego warto pracować z Sassem?
17. Odpowiednie użycie Sassa

# Wstęp

# Czym jest Sass?

## Syntactically Awesome Style Sheets

**Sass** to język skryptowy rozszerzający CSS o dodatkowe możliwości.



# Preprocessing

## Czym jest preprocessing?

- **Sass** nie jest interpretowany przez przeglądarki, więc musi zostać przetworzony na CSS.
  - Umożliwia to **preprocessing**, czyli przetwarzanie kodu źródłowego według określonych reguł na kod wyjściowy.
  - Kodem źródłowym jest skrypt języka SCSS. Wynikiem – kod CSS.
- Domyślnie przetwarzanie **Sassa** zostało napisane w języku Ruby.
  - Dzięki bibliotekom **LibSass** można kompilować **Sass** w innych językach: Javie, PHP, .NET itp.

# Sass jest preprocesorem CSS



# Wynikowe pliki CSS

## Wczytujemy pliki CSS

W plikach HTML odwołujemy się do wygenerowanych plików **CSS** – nie do plików **Sassa**.

## Dlaczego?

- Przy kompilowaniu plików **Sassa** wszystkie zmiany wprowadzone bezpośrednio w plikach **CSS** zostaną nadpisane wynikiem kompilacji z plików **Sassa**.
- Zmiany z plików **CSS** nie są przenoszone do plików **Sassa**.

## Ważne!

- Do HTML-a dodajemy pliki **CSS** a nie pliki **Sassa**.
- Nie modyfikujemy plików **CSS**.

# .sass – składnia

## Pliki z rozszerzeniem .sass

**Sass** może być zapisany na dwa sposoby (pliki z rozszerzeniem **.sass** oraz **.scss**). Przykład pierwszego z nich widzimy po prawej.

- W pierwotnej składni **.sass** nie używano nawiasów klamrowych oraz średników.
- Zagnieżdżenia tworzone były przez wcięcia w liniach kodu, **tzw. indented syntax** zaczerpnięty z języka **Haml**.

## Przykład pliku z rozszerzeniem .sass

```
nav
  ul
    margin: 0
    padding: 0
  li
    display: block
```



# .scss – składnia

## Pliki z rozszerzeniem .scss

- Z czasem (wersja 3) składnię pierwotną zastąpiono składnią **SCSS** ( **Sassy CSS**), tak aby była jak najbardziej zbliżona do **CSS**.
- Dzięki temu każdy poprawny kod **CSS** może być wstawiony do plików **Sassa**.
- W przypadku starej składni z kodu **CSS** trzeba było usuwać zbędne znaki.

## Przykład pliku z rozszerzeniem .scss

```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
  }  
  li {  
    display: block;  
  }  
}
```

# Stosujemy SCSS – Sassy CSS

## Składnia sass

```
nav
  ul
    margin: 0
    padding: 0
  li
    display: block
```

## Składnia scss

```
nav {
  ul {
    margin: 0;
    padding: 0;
  }
  li {
    display: block;
  }
}
```

Sass zawiera też domyślnie narzędzie **Sass-convert** pozwalające zawsze szybko przejść z jednej składni na drugą niemal całkowicie automatycznie, a nawet pomóc przekonwertować kod CSS na SCSS.

# Instalacja i uruchomienie Sassa

# Rozpoczęcie pracy z Sassesem

Kompilacja Sass możliwa jest na różnych systemach operacyjnych na różne sposoby.

Przykład:

- przez aplikacje takie jak **npm**,
- przez wiersz poleceń – po zainstalowaniu środowiska **Rubiego** i biblioteki **Sass**.

# Kompilacja bez instalacji Rubiego

→ Wymaga instalacji modułu

```
npm install gulp-sass
```

→ Uruchomienie

```
gulp sass
```

Więcej:

<https://github.com/dlmanning/gulp-sass>

# Kompilacja bez instalacji Rubiego

## Przykład

```
var gulp = require('gulp');
var sass = require('gulp-sass');
gulp.task('sass', function() {
  return gulp.src('scss/style.scss')
    .pipe(sass({errLogToConsole: true}))
    .pipe(gulp.dest('css'))
});
gulp.task('watch', function(){
  gulp.watch('scss/*.scss', ['sass']);
})
```

# Kompilacja przy pomocy Ruby

## Instalacja Rubiego

- **Windows:** <http://rubyinstaller.org>
- **Linux:**
  - ➔ <http://bitnami.com/stack/ruby/installer>
  - ➔ RVM - <http://rvm.io/rvm/install>
  - ➔ `sudo apt-get install ruby`
- **OS X:**
  - ➔ Domyślnie istnieje wersja preinstalowana
  - ➔ <http://bitnami.com/stack/ruby/installer>

# Kompilacja przy pomocy Ruby

## Windows

Najprostszą metodą jest użycie instalatora:

<http://rubyinstaller.org>

Aby komendy Rubiego były dostępne z systemowego wiersza poleceń przy instalacji należy zaznaczyć opcję:

### **Add Ruby executables to your PATH**

Inaczej komendy będą dostępne tylko z wiersza poleceń uruchomionego z katalogu, w którym zainstalowany jest Ruby.

## OS X

System OS X ma preinstalowaną wersję Rubiego.

Można jednak zainstalować inną wersję, np.:

<http://bitnami.com/stack/ruby/installer>



# Instalacja Rubiego na Linuksie

- Najprostszą metodą jest użycie systemowego package managera.
- Można również użyć instalatora:  
<http://bitnami.com/stack/ruby/installer>  
lub zainstalować od razu manager wersji Ruby RVM: <http://rvm.io/rvm/install>

# Instalacja Sassa

Po zainstalowaniu Rubiego, instalacja Sassa sprowadza się do komendy w terminalu lub wierszu poleceń:

**Windows:**

```
gem install sass
```

**OS X, Linux:**

```
sudo gem install sass
```

Aby sprawdzić, czy instalacja się powiodła, możemy sprawdzić wersję:

```
sass -v
```

```
$ sass -v  
Sass 3.4.19 (Selective Steve)
```

# Kompilowanie plików Sassa przez wiersz poleceń

- Aby uruchomić kompilację, należy wpisać komendę:

```
sass input.scss output.css
```

- Bardziej przydatna jest jednak komenda, która obserwuje zapisywanie zmian w plikach Sass i automatycznie dokonuje kompilacji:

```
sass --watch  
input.scss:output.css
```

- Komendy można również stosować w odniesieniu do katalogów:

```
sass --watch  
app/sass:public/stylesheets
```

# Kompilacja przy pomocy Ruby'ego w Gulpie

Na niektórych systemach operacyjnych lub konfiguracjach moduł **gulp-sass** może nie działać (występują problemy z biblioteką LibSass).

Możemy wykorzystać kompilator Sass'a napisany w Ruby wewnątrz Gulpa.

- Wymagana instalacja modułu
  - ➔ **instalacja Ruby'ego oraz Sassa**  
(jest opisana na poprzednich slajdach)
  - ➔ **npm install gulp-ruby-sass**

- Uruchomienie

**gulp sass**

Więcej: <https://github.com/sindresorhus/gulp-ruby-sass>

# Kompilacja przy pomocy Ruby'ego w Gulpie

## Przykład

```
var gulp = require('gulp');
var sass = require('gulp-ruby-sass');
gulp.task('sass', function() {
  return sass('scss/style.scss',
    {style: 'expanded'})
    .on('error', sass.logError)
    .pipe(gulp.dest('css'));
});
gulp.task('watch', function(){
  gulp.watch('sass/*.scss', ['sass']);
})
```

# Kompilacja przy pomocy Ruby'ego w Gulpie

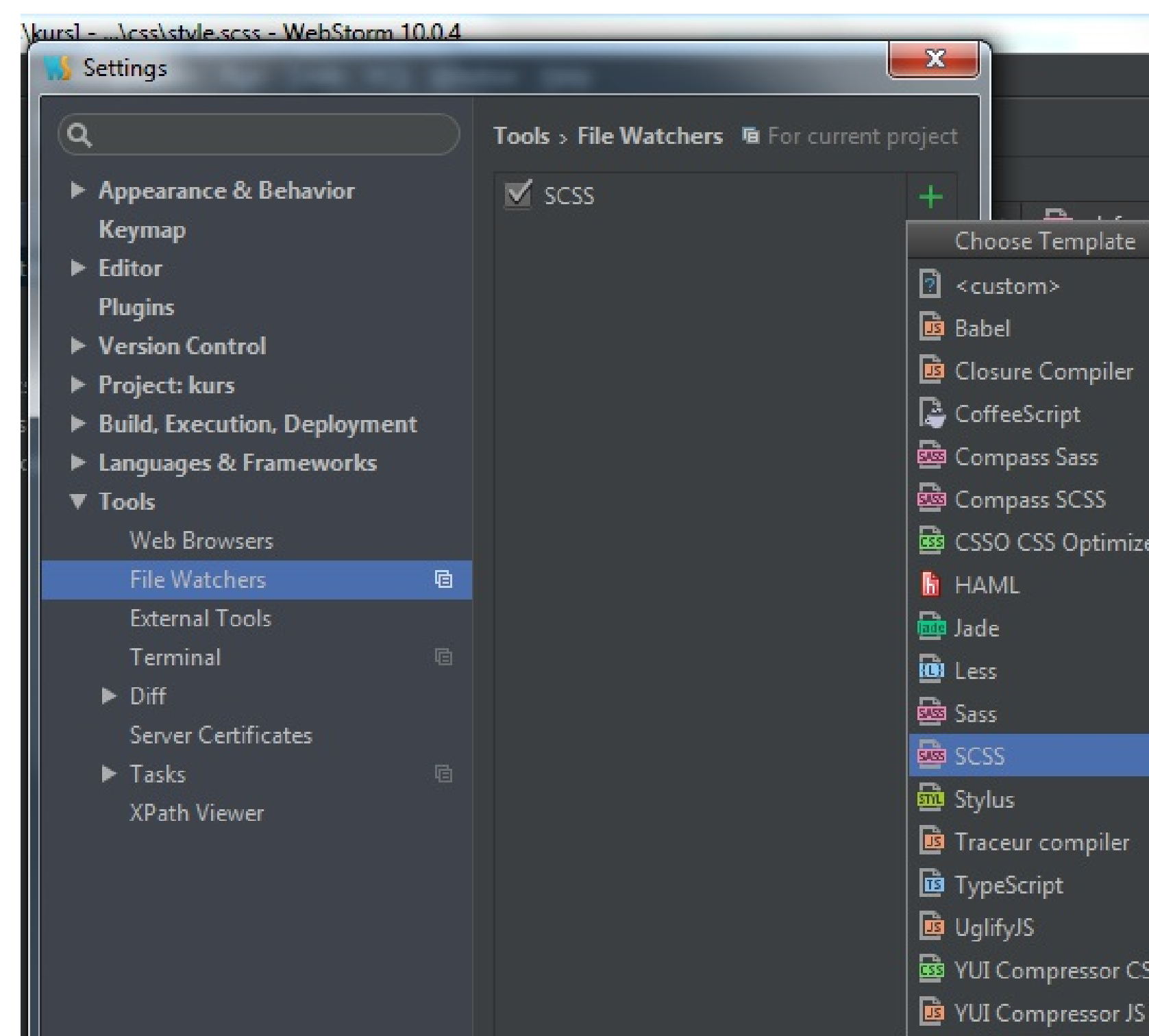
## Przykład

```
var gulp = require('gulp');
var sass = require('gulp-ruby-sass');
gulp.task('sass', function() {
  return sass('scss/style.scss',
    {style: 'expanded'})
    .on('error', sass.logError)
    .pipe(gulp.dest('css'));
});
gulp.task('watch', function(){
  gulp.watch('sass/*.scss', ['sass']);
})
```

**style: 'expanded'** - Zmiana stylu wyjściowego, więcej na kolejnych slajdach.

# Sass w edytorze WebStorm

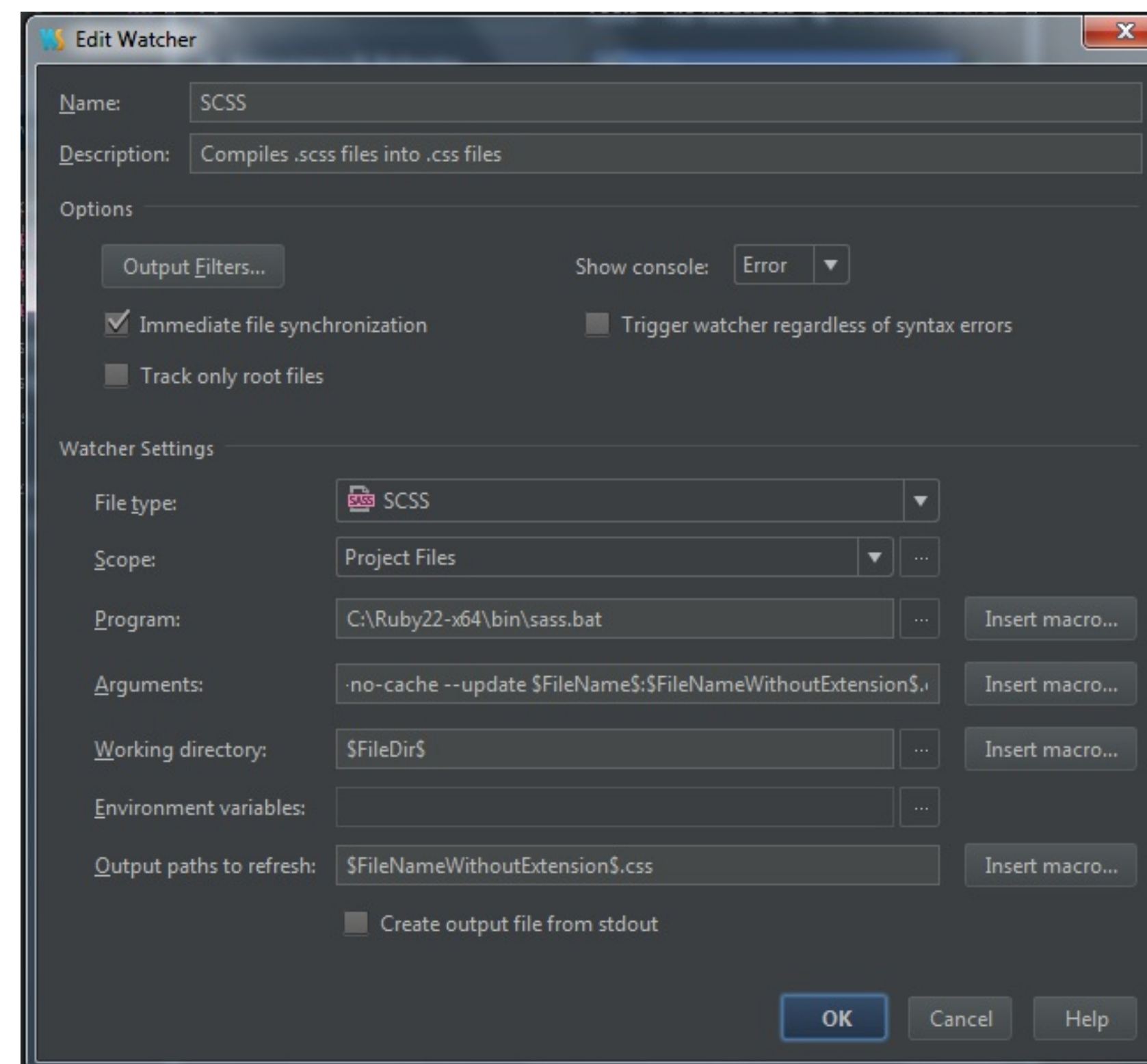
- **WebStorm** ma wsparcie dla Sassa. Rozpoznaje i uruchamia proces kompilacji plików **Sassa** (**.sass** i **.scss**) oraz koloruje składnię.
- WebStorm nie ma wbudowanego Rubiego, więc również wymagana jest jego instalacja.
- Potrzebne są również wtyczki **SASS support** oraz **File Watchers**, które są domyślnie zainstalowane i włączone.





# Dodanie plików Sass do File Watchers

- Aby pliki Sass były kompilowane, należy dodać je w ustawieniach File Watchers: **(File -> Settings -> Tools -> File Watchers)**
- Należy wybrać rodzaj pliku (.sass lub .scss), podać ścieżkę do lokalizacji Rubiego i określić ustawienia.
- Przydatna może okazać się pomoc **WebStorm**:  
<http://www.jetbrains.com/webstorm/help/transpiling-sass-less-and-scss-to-css.html>





Czas na zadania

# **Formatowanie stylu wyjściowych**

# Dostępne cztery style kodu

## Jak kod będzie wyglądał po kompilacji?

Kod CSS może być wygenerowany w jednym z czterech stylów:

- nested,
- expanded,
- compact,
- compressed.

# Dostępne cztery style kodu

Styl można określić na dwa sposoby:

- przez parametr **outputStyle** (gulp-sass) lub parametr **style** (gulp-ruby-sass) w funkcji **sass** dla **gulpfile** np.:

```
/* gulp-sass */  
.pipe(sass({  
  outputStyle: 'expanded',  
  sourceComments: 'map'  
}))  
/* gulp-ruby-sass */  
sass({  
  style: 'expanded'  
})
```

- przez flagę w komendzie (przy kompilacji przez linię poleceń):

```
sass input.scss output.css --style compressed
```

# :nested

## Domyślny

Oto styl domyślny, który przez wcięcia odzwierciedla strukturę klas.

```
#main {  
  color: #fff;  
  background-color: #000; }  
  #main p {  
    width: 10em; }  
.huge {  
  font-size: 10em;  
  font-weight: bold;  
  text-decoration: underline; }
```

# :expanded

## Najbardziej przyjazny

- Ten styl najłatwiej jest dla nas najbardziej czytelny.
- Wcięcia są stosowane tylko dla właściwości. Każda z nich wstawiana jest do osobnej linii.

```
#main {  
  color: #fff;  
  background-color: #000;  
}  
#main p {  
  width: 10em;  
}  
.huge {  
  font-size: 10em;  
  font-weight: bold;  
  text-decoration: underline;  
}
```

# :compact

## Reguły css w jednej linii

- Zajmuje mniej miejsca na ekranie i pozwala skupić większą uwagę na selektorach niż na ich właściwościach.
- Każda deklaracja zajmuje tylko jedną linię.

```
#main { color: #fff; background-color: #000; }  
#main p { width: 10em; }  
.huge { font-size: 10em; font-weight: bold;  
        text-decoration: underline; }
```

# :compressed

## Skompresowany

- Zajmuje najmniej miejsca, likwiduje możliwe spacje.
- Styl ma kilka metod kompresji.
- W tej wersji rozmiar pliku CSS jest najmniejszy.

```
#main{color:#fff;background-color:#000}#main p{width:10em}  
.huge{font-size:10em;font-weight:bold;text-decoration:underline}
```



# Mapowanie kodu źródłowego

# Mapowanie kodu źródłowego

- **Sourcemaps** czyli mapy kodu źródłowego informują przeglądarkę o tym, w której linii pliku **Sass** znajduje się źródło wygenerowanej deklaracji **CSS**.
  - Znacznie ułatwia to debugowanie i edycję plików, gdyż struktura plików i kodu z reguły różni się między **Sass** a **CSS**.
- Opcja ta jest domyślnie włączona.
  - Domyślna ścieżka URL do mapy jest relatywna do pliku **CSS**.
  - Flaga **--sourcemap none** wyłącza generowanie mapy.

# Włączenie sourcemaps w gulpie

## Dodanie pakietu gulp-sourcemaps

Po dodaniu pakietu **gulp-sourcemaps**, należy wewnątrz przetwarzania **pipe** zainicjować sourcemapę oraz przed zapisem pliku wynikowego css należy ją również zapisać. Po uruchomieniu gulpa w pliku css zostanie wówczas dodana podobna linijka kodu:

```
/*# sourceMappingURL=data:application/json;base64,
```

Natomiast w Web Developer Tools możemy podglądać kod scss następująco:

```
body {                                     main.scss:1  
  color: ■ red;  
  background-color: ■ blue;  
  margin: ► 0;  
}
```

# Włączenie sourcemaps w gulpie

## gulp-ruby-sass

```
/* ...wcześniejszy kod */
var sourcemaps = require('gulp-sourcemaps');
gulp.task('sass', function() {
  return sass('scss/style.scss',
    {style: 'expanded', sourcemap: true})
    .on('error', sass.logError)
    .pipe(sourcemaps.write())
    .pipe(gulp.dest('css'))
});
/* ...dalsza część kodu */
```

# Włączenie sourcemaps w gulpie

## gulp-ruby-sass

```
/* ...wcześniejszy kod */
var sourcemaps = require('gulp-sourcemaps');
gulp.task('sass', function() {
  return sass('scss/style.scss',
    {style: 'expanded', sourcemap: true})
    .on('error', sass.logError)
    .pipe(sourcemaps.write())
    .pipe(gulp.dest('css'))
});
/* ...dalsza część kodu */
```

Flaga **sourcemap** ustawiona na **true** działa identycznie jak **sourcemaps.init()**.

# Włączenie sourcemaps w gulpie

## gulp-sass

```
/* ...wcześniejszy kod */
var sourcemaps = require('gulp-sourcemaps');
gulp.task('sass', function() {
    return gulp.src('scss/style.scss')
        .pipe(sourcemaps.init())
        .pipe(sass({errLogToConsole: true}))
        .pipe(sourcemaps.write())
        .pipe(gulp.dest('css'));
});
/* ...dalsza część kodu */
```

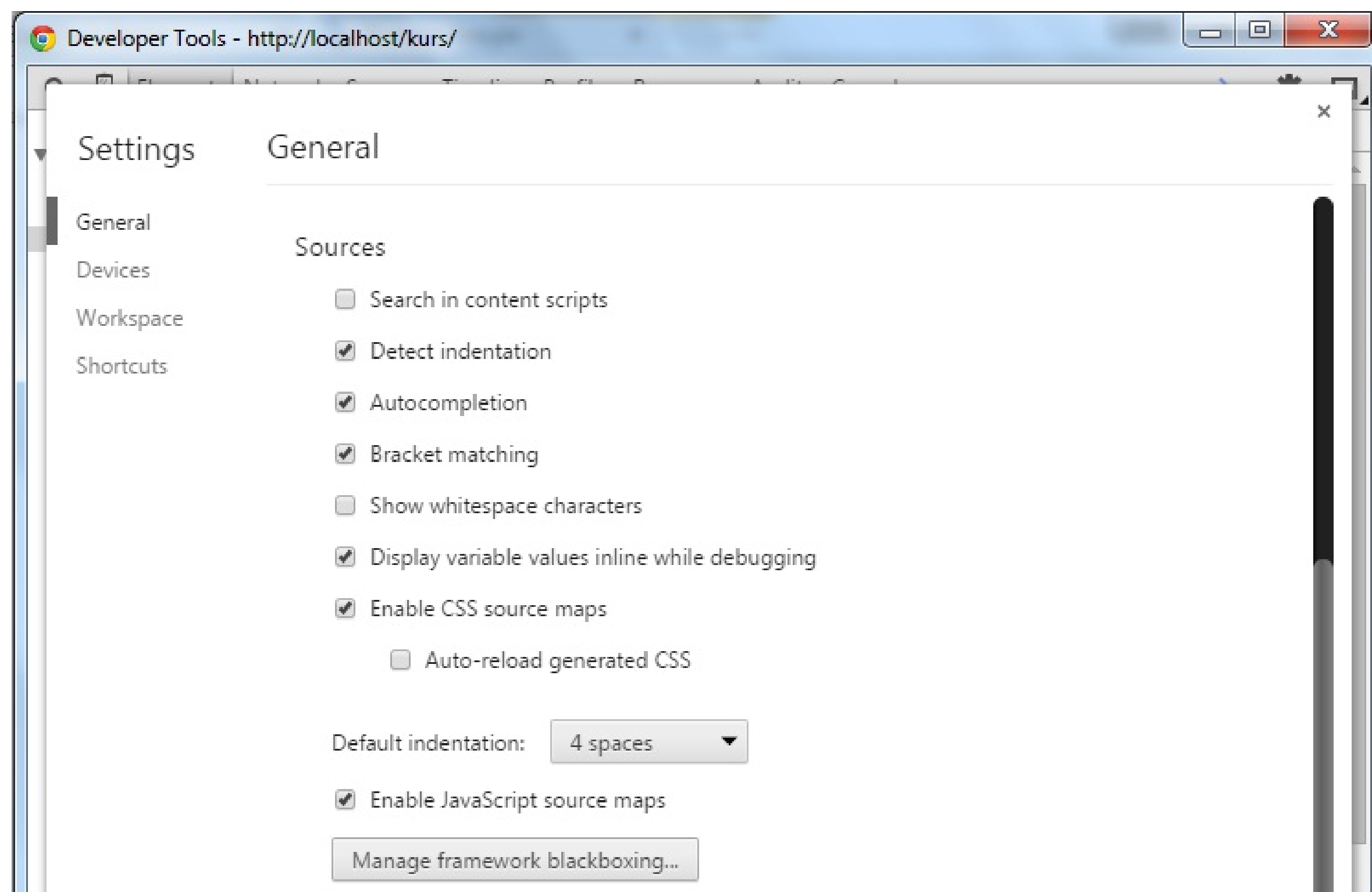
# Wsparcie przeglądarek

- Chrome, Safari, Firefox oraz Opera mają domyślne wsparcie dla **sourcemaps**.
- Również Internet Explorer 11 ma wsparcie dla map.

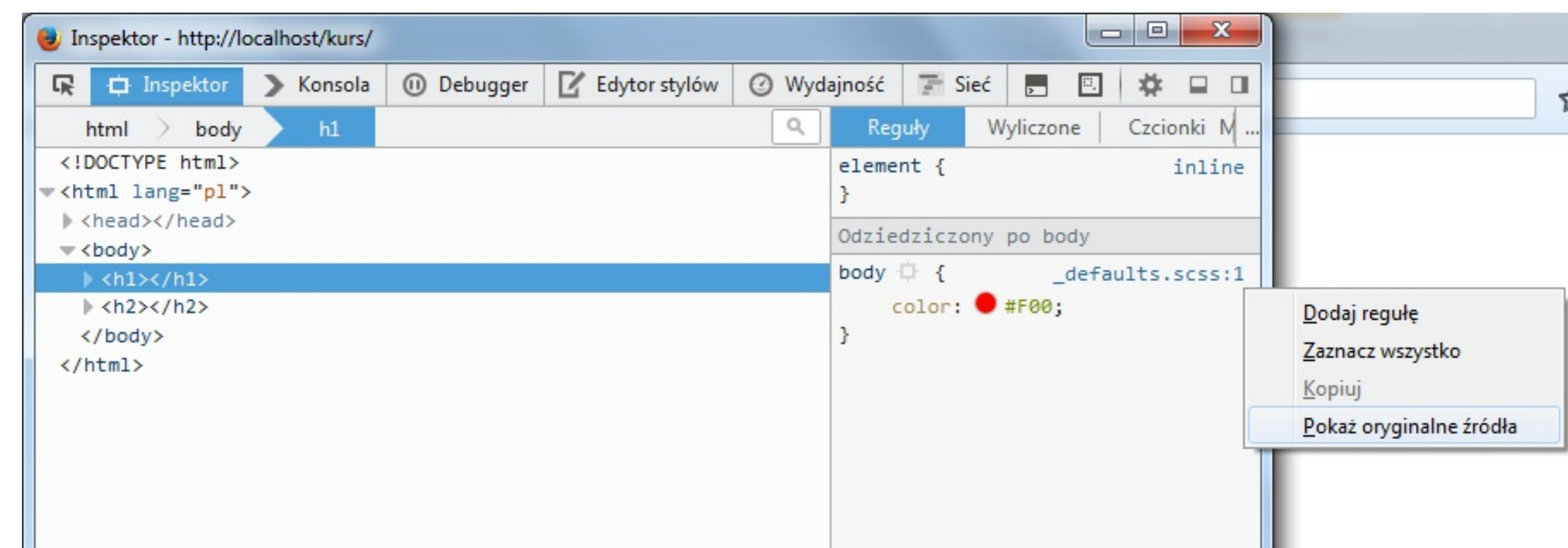


# Wsparcie przeglądarek

## Chrome



## Firefox





# Komentarze

# Dwa rodzaje komentarzy

W Sassie możliwe jest użycie dwóch typów komentarzy:

- `/* */` – ten komentarz zostanie umieszczony w wynikowym pliku CSS.
- `//` – tego komentarza nie będzie w wynikowym pliku CSS, jest przydatny głównie programiście pracującemu z danymi plikami Sass.

# Dwa rodzaje komentarzy

style.scss

```
// Ten komentarz nie  
// będzie widoczny  
// w pliku wynikowym CSS
```

```
/* Ten komentarz będzie */
```

style.css

```
/* Ten komentarz będzie */
```

# Zmienne

# Zmienne

## Pamiętaj o dolarze

- Zmienne w Sassie przechowują dowolną wartość CSS.
- Do definiowania zmiennych używamy symbolu \$.

## Przykład

```
$fontColor: blue;
```

# Przykład użycia zmiennej

## Składnia scss

```
$font-stack: Helvetica, sans-serif;  
$primary-color: #333;  
body {  
  font: 100% $font-stack;  
  color: $primary-color;  
}
```

## Rezultat w css

```
body {  
  font: 100% Helvetica, sans-serif;  
  color: #333;  
}
```

# Zmienne

## Zmienne boolowskie – logiczne

```
$border: false;  
$rounded: true;
```

## Liczby

```
$border: 1px;  
$rounded: 20px;
```

## Null

```
$border: null;
```

## Stringi

```
$title: 'Moja strona';  
$fontFamily: 'Helvetica';
```

## Listy

```
$users: john, alan, luke;  
$padding: 10px 2px 20px 0;
```

## Kolory

```
$mainColor: aqua;  
$textColor: rgb(255, 189, 2);  
$border: #444;
```

# Dobre rady

## Semantyczne nazwy

Używaj semantycznych nazw dla zmiennych.

### Dobrze

```
$main-color: red;  
$accent-color: yellow;
```

### Źle

```
$red: red;  
$yellow: yellow;
```



# Dobre rady

## Postfix lub prefix

Ustal konwencję nazewnictwa np. stosując **postfix** lub **prefix** dla określonych zmiennych.

### prefix

```
$main-color: red;  
$accent-color: yellow;  
$footer-height: 40px;  
$header-padding: 10px;
```

### postfix

```
$color-main: red;  
$color-accent: yellow;
```

## Osobny plik

Trzymaj zmienne w osobnym pliku np.: **\_variables.scss** lub **\_config.scss**

```
@import "variables";
```

Czas na zadania

# Zagnieżdżanie właściwości i selektorów

# Przykład zagnieżdżania selektorów

Przyjrzyj się HTML-owi obok. Na kolejnym slajdzie znajdziesz przykład kodu **scss** oraz przekompilowanego **css**.

## HTML

```
<div class="main-content">  
  <h3>Title</h3>  
  <span>23.05.2016</span>  
</div>
```

# Przykład zagnieżdżania selektorów

## Składnia scss

```
.main-content {  
  border: 1px solid red;  
  h3 {  
    margin: 10px 5px;  
    font-size: 3em;  
  }  
  span {  
    margin: 5px 0;  
    font-size: 1em;  
  }  
}
```

## Rezultat w css

```
.main-content {  
  border: 1px solid red;  
}  
.main-content h3 {  
  margin: 10px 5px;  
  font-size: 3em;  
}  
.main-content span {  
  margin: 5px 0;  
  font-size: 1em;  
}
```

# Przykład zagnieżdżania selektorów

## Składnia scss

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

## Rezultat w css

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

# Selektor rodzica &

## Ampersand – &

- W przypadku pseudoklas przydatny jest operator &.
- Jest on odwołaniem się do rodzica danego selektora.

## Składnia scss

```
a {  
  &:hover {  
    color: red;  
  }  
  &:visited {  
    color: black;  
  }  
}
```

- & w wygenerowanym pliku, jest powtórzeniem jego nazwy.

## Rezultat w css

```
a:hover {  
  color: red;  
}  
a:visited {  
  color: black;  
}
```



# Dobre rady

- Dobrą praktyką jest unikanie zbyt wielu poziomów zagnieżdżeń, gdyż taki kod jest trudny do utrzymania.
- Staraj się ograniczać zagnieżdżanie do 3–4 poziomów.





Czas na zadania

# Partials i importowanie

# Metoda @import i wiele plików Sass

## Struktura plików

Sass pozwala na rozdzielenie kodu na wiele plików i zaimportowanie ich do jednego pliku zbiorczego metodą **@import**.

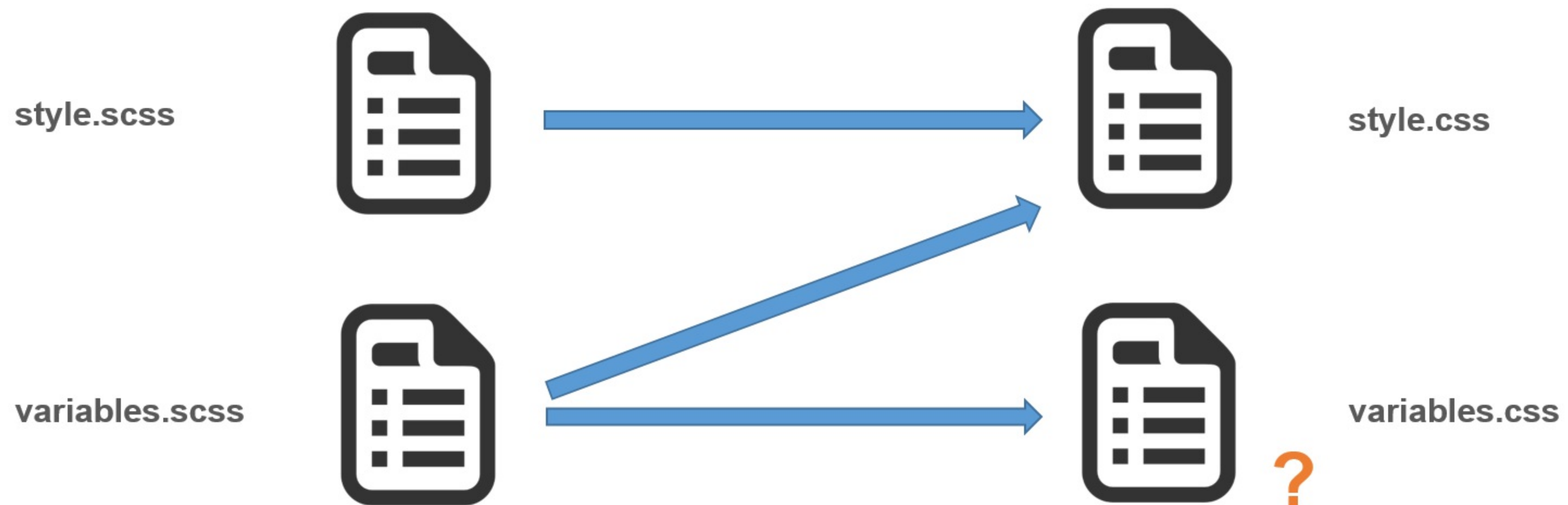
Dzięki temu struktura plików może być bardziej modułowa.

Np. pliki mogą być podzielone tematycznie:

- typografia,
- layout,
- tabele,
- listy,
- okna modalne.

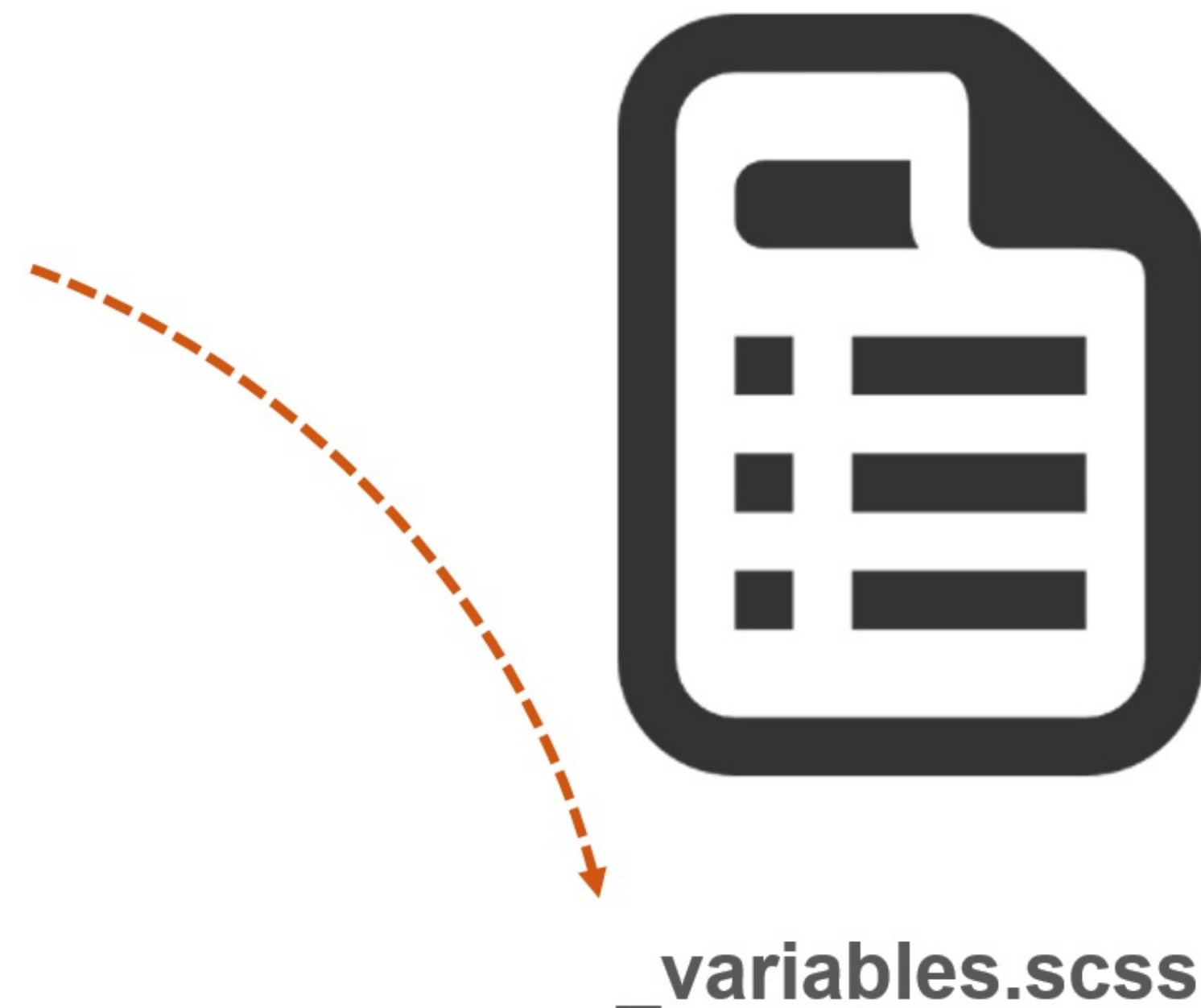
# Partials

Stworzenie nowego pliku **Scss** spowoduje, że zostanie on wygenerowany jako nowy plik **CSS**.

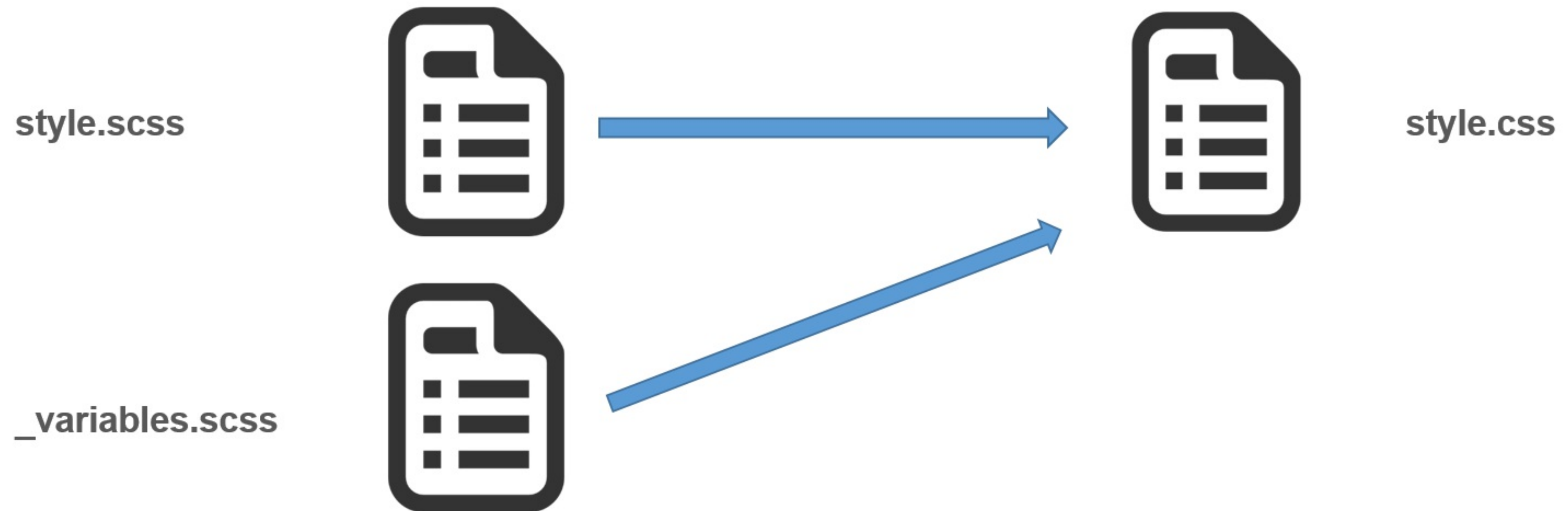


# Partials

- Aby uniknąć tworzenia niepotrzebnego pliku **css** wystarczy na początku nazwy wpisać znak **podkreślnika**.
- Pliki cząstkowe nie będą pojedynczo wczytywane przez przeglądarkę.
- Ich zawartość zostanie wstawiona bezpośrednio do jednego wynikowego pliku css razem z innymi plikami cząstkowymi.
- Pozwala to na lepszą organizację plików i pozbycie się zbędnych bajtów.



# Partials



# Importowanie plików Sassa

- Dobrą praktyką jest stworzenie jednego pliku **sass** składającego się jedynie z metod importowania.
  - Dzięki temu struktura kodu jest łatwiejsza do zrozumienia a zmienne globalne dostępne z poziomu różnych plików.
- Pamiętaj, że odwołanie się do zmiennej jest możliwe tylko wtedy, jeśli zmienna została zdefiniowana w miejscu poprzedzającym odwołanie.
  - Wynikiem będzie jeden plik css zawierający deklaracje ze wszystkich plików cząstkowych i to on zostanie wczytany przez przeglądarkę.

# Importowanie plików Sassa - partials

## Składnia SCSS

Mamy dwa pliki Sassa:

- `_reset.scss`,
- `style.scss`.

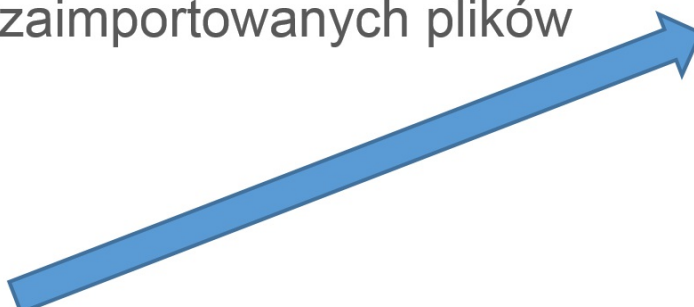
### `_reset.scss`

```
html, body, ul, ol {  
  margin: 0;  
  padding: 0;  
}
```

### `style.scss`

```
@import 'reset';  
body {  
  font: sans-serif;  
  color: #fff;  
}
```

Po kompilacji powstanie  
jeden plik `css`  
złożony ze wszystkich  
zaimportowanych plików



### `style.css`

```
html, body, ul, ol {  
  margin: 0;  
  padding: 0;  
}  
body {  
  font: sans-serif;  
  color: #fff;  
}
```



# Metoda @import oraz link

Jak wiemy, pliki dodawane do **HTML-a** są wczytywane przez osobne zapytanie do serwera, co może znacznie wydłużyć czas ładowania strony.

Różnica pomiędzy metodą **@import** w **Sassie** a natywną metodą **CSS** polega na tym, że wszystkie deklaracje w **Sassie** są kompilowane do jednego pliku **css**.

Dzięki czemu do serwera jest wysyłane **tylko jedno zapytanie** (HTTP request).

# Metoda @import oraz link

## Ładowanie kilku plików CSS

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

## Ładowanie jednego pliku CSS

### style.scss

```
@import 'buttons';  
body {  
  font: sans-serif;  
  color: #fff;  
}
```

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

# Struktura projektu Sass

## Struktura katalogu

```
sass/  
|-- modules/  
|   |-- _all.scss  
|   |-- _colors.scss  
|-- partials/  
|   |-- _base.scss  
|   |-- _reset.scss  
|   |-- _buttons.scss  
|-- vendor/  
|   |-- _jquery.ui.scss  
main.scss
```

## Jak wygląda plik main.scss

```
/* Modules */  
@import "modules/all";  
/* all.scss – import all  
other modules */  
/* Partials */  
@import "partials/reset";  
@import "partials/base";  
@import "partials/buttons";  
/* Vendor */  
@import "vendor/jquery.ui";
```

# Struktura projektu Sass

## Struktura katalogu

```
sass/  
|-- modules/  
|   |-- _all.scss  
|   |-- _colors.scss  
|-- partials/  
|   |-- _base.scss  
|   |-- _reset.scss  
|   |-- _buttons.scss  
|-- vendor/  
|   |-- _jquery.ui.scss  
main.scss
```

Tutaj przechowujemy wszelkiego rodzaju mixiny, funkcje i zmienne.

## Jak wygląda plik main.scss

```
/* Modules */  
@import "modules/all";  
/* all.scss – import all  
other modules */  
/* Partials */  
@import "partials/reset";  
@import "partials/base";  
@import "partials/buttons";  
/* Vendor */  
@import "vendor/jquery.ui";
```

# Struktura projektu Sass

## Struktura katalogu

```
sass/  
|-- modules/  
|   |-- _all.scss  
|   |-- _colors.scss  
|-- partials/  
|   |-- _base.scss  
|   |-- _reset.scss  
|   |-- _buttons.scss  
|-- vendor/  
|   |-- _jquery.ui.scss  
main.scss
```

Tutaj przechowujemy części naszego projektu.

## Jak wygląda plik main.scss

```
/* Modules */  
@import "modules/all";  
/* all.scss – import all  
other modules */  
/* Partials */  
@import "partials/reset";  
@import "partials/base";  
@import "partials/buttons";  
/* Vendor */  
@import "vendor/jquery.ui";
```

# Struktura projektu Sass

## Struktura katalogu

```
sass/  
|-- modules/  
|   |-- _all.scss  
|   |-- _colors.scss  
|-- partials/  
|   |-- _base.scss  
|   |-- _reset.scss  
|   |-- _buttons.scss  
|-- vendor/  
|   |-- _jquery.ui.scss  
main.scss
```

Tutaj przechowujemy kod z innych bibliotek, jeżeli z nich korzystamy.

## Jak wygląda plik main.scss

```
/* Modules */  
@import "modules/all";  
/* all.scss – import all  
other modules */  
/* Partials */  
@import "partials/reset";  
@import "partials/base";  
@import "partials/buttons";  
/* Vendor */  
@import "vendor/jquery.ui";
```

# Struktura projektu Sass

- **vendor** – tutaj przechowujemy kod z innych bibliotek, jeżeli z nich korzystamy.
- **main.scss** – to plik, który powinien składać się z metod **@import** importujących poszczególne elementy.
- **modules** – tutaj przechowujemy wszelkiego rodzaju mixiny, funkcje i zmienne.
- **partials** – tutaj przechowujemy części naszego projektu.

Sposobów na dzielenie jest wiele np.:

- header,
- content,
- footer.

Ale można też np.:

- buttons,
- typography,
- grid,
- forms,
- itp.

Czas na zadania



# Mixins

# Czym jest mixin?

- **Mixin** pozwala na zapisanie deklaracji CSS (właściwość i jej wartość) w postaci pewnego rodzaju zmiennej.
- Pozwala to na łatwiejsze wstawianie często powtarzającego się kodu np. prefiksy przeglądarek w niektórych właściwościach (**box-shadow**).
- **Mixin** może również przekazywać argumenty.

# Przykład użycia

## Składnia scss

```
@mixin boxShadow($color) {  
  -webkit-box-shadow: 10px 10px 5px 0 $color;  
  -moz-box-shadow: 10px 10px 5px 0 $color;  
  box-shadow: 10px 10px 5px 0 $color;  
}  
.box {  
  @include boxShadow(red);  
}
```

Mixin definiuje się metodą **@mixin**, a wstawia w odpowiednie miejsce za pomocą **@include**.

# Przykład użycia

## Składnia scss

```
@mixin boxShadow($color) {  
  -webkit-box-shadow: 10px 10px 5px 0 $color;  
  -moz-box-shadow: 10px 10px 5px 0 $color;  
  box-shadow: 10px 10px 5px 0 $color;  
}  
.box {  
  @include boxShadow(red);  
}
```

Mixin definiuje się metodą **@mixin**, a wstawia w odpowiednie miejsce za pomocą **@include**.

Argument, który tu ustawimy, będzie dostępny wewnątrz mixina.

# Przykład użycia

## Rezultat css

```
.box {  
  -webkit-box-shadow: 10px 10px 5px 0 red;  
  -moz-box-shadow: 10px 10px 5px 0 red;  
  box-shadow: 10px 10px 5px 0 red;  
}
```

# Mixin – argumenty

Dodanie trzech kropek do argumentu tworzy zmienną **vararg**, dzięki temu możemy rozdzielać własności CSS przecinkiem. Jest to przydatne np. przy tworzeniu m.in. animation czy transition.

## Składnia scss

```
@mixin animation($value...) {  
  -webkit-animation: $value;  
  -moz-animation: $value;  
  animation: $value;  
}  
.box {  
  @include animation(slide 1s infinity,  
                    move 0.5s infinity alternate);  
}
```

# Mixin – argumenty

Dodanie trzech kropek do argumentu tworzy zmienną **vararg**, dzięki temu możemy rozdzielać własności CSS przecinkiem. Jest to przydatne np. przy tworzeniu m.in. animation czy transition.

## Składnia scss

```
@mixin animation($value...) {  
  -webkit-animation: $value;  
  -moz-animation: $value;  
  animation: $value;  
}  
.box {  
  @include animation(slide 1s infinity,  
                    move 0.5s infinity alternate);  
}
```

Stworzenie **vararg**.

# Mixin

## Pamiętaj!

- **@include** – używamy mixina,
- **@import** – importujemy plik.



Czas na zadania

# Dziedziczenie

## Metoda `@extend`

# Czym jest extend?

Jeśli chcemy utworzyć styl zawierający wszystkie deklaracje innego stylu plus kilka dodatkowych możemy użyć dyrektywy **@extend**.

# @extend – przykład użycia

## Składnia scss

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

## Rezultat css

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

# @extend – niezamierzone style

## Niezamierzone dodatkowe style

### Składnia scss

```
a {  
  color: red;  
}  
.error {  
  @extend a;  
  border-color: red;  
}  
a:hover {  
  color: #333;  
  text-decoration: underline;  
}
```

### Rezultat css

```
a:hover, .error:hover {  
  color: #333;  
  text-decoration: underline;  
}  
a, .error {  
  color: red;  
}  
.error {  
  border-color: red;  
}
```

Efekt? Każdy **.error** będzie podkreślał się i zmieniał kolor przy najechaniu kursorem.

# @extend – zbędne selektory

## Wiele niezamierzonych i zbędnych selektorów

### W najlepszym przypadku

Plik wynikowy będzie zawierał dużo niepotrzebnego kodu i selektorów.

### Składnia scss

```
#admin .nav a {  
  color: #333;  
}  
a {  
  color: #666;  
}  
#demo .body .link {  
  @extend a;  
}
```

### W najgorszym przypadku

Nieoczekiwane nadpisanie stylów różnych elementów, trudne do wykrycia.

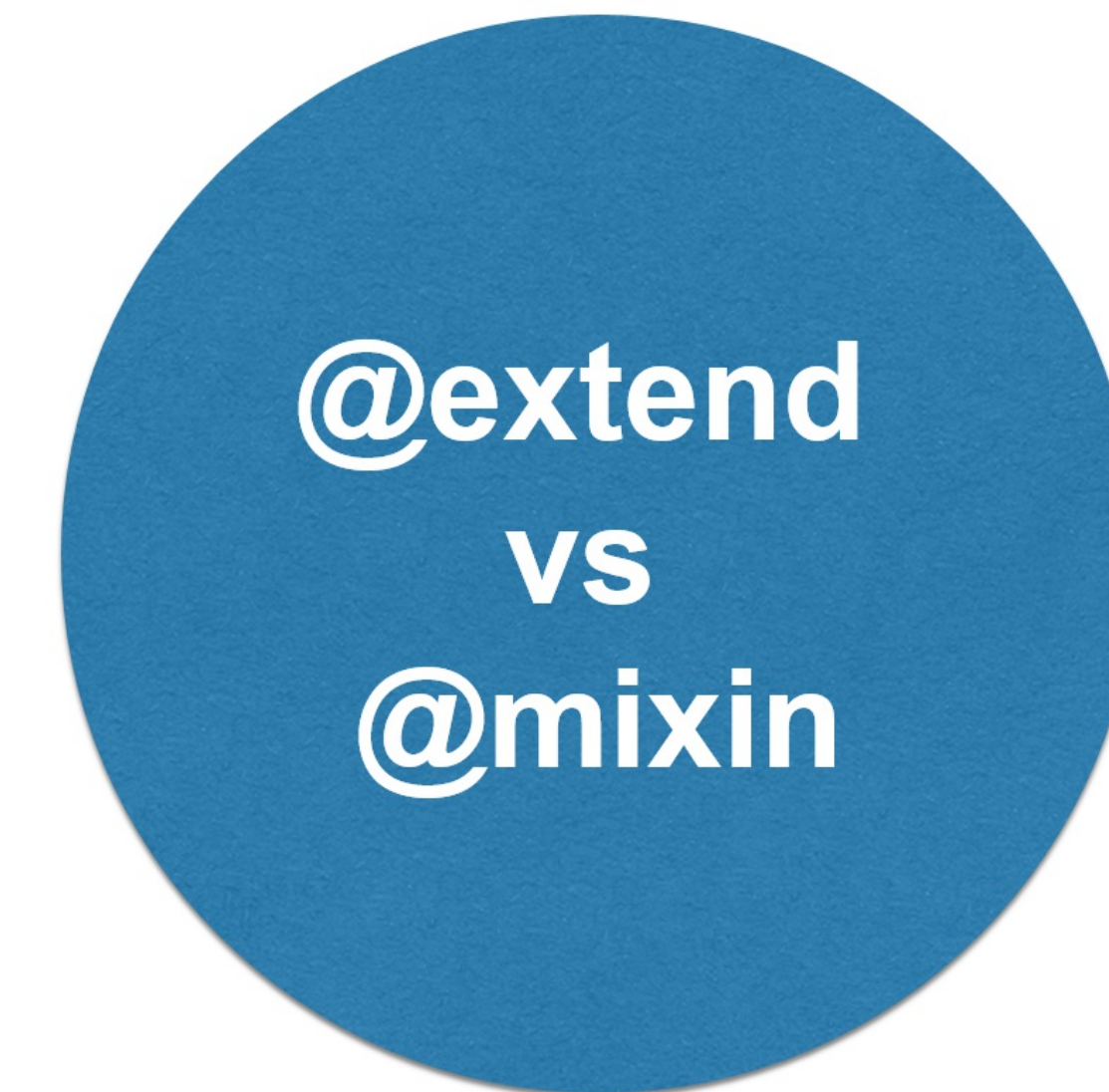
### Rezultat css

```
#admin .nav a,  
#admin .nav #demo .body .link,  
#demo .body #admin .nav .link {  
  color: #333;  
}  
a, #demo .body .link {  
  color: #666;  
}
```

# Różnica między @extend a @mixin

**@extend** i **@mixin** używane są w innych celach.

- **@extend** – służy do tego, aby wybrane selektory miały te same właściwości. Nie ma tutaj możliwości użycia argumentów.
- **@mixin** – pozwala na zagnieżdżenie dowolnego zestawu właściwości w różnych selektorach.
- **@extend** – może wykorzystać **@mixin**, jeśli został wstawiony dla selektora, który jest kopiowany.





# @extend i placeholder %

- W przypadku metody **@extend** przydatny jest tzw. **placeholder selector** %.
- Jeśli selektor z placeholderem nie zostanie nigdzie użyty za pomocą **@extend**, to nie zostanie wygenerowany w CSS jako właściwość klasy lub identyfikatora.



# @extend i placeholder %

## Składnia scss

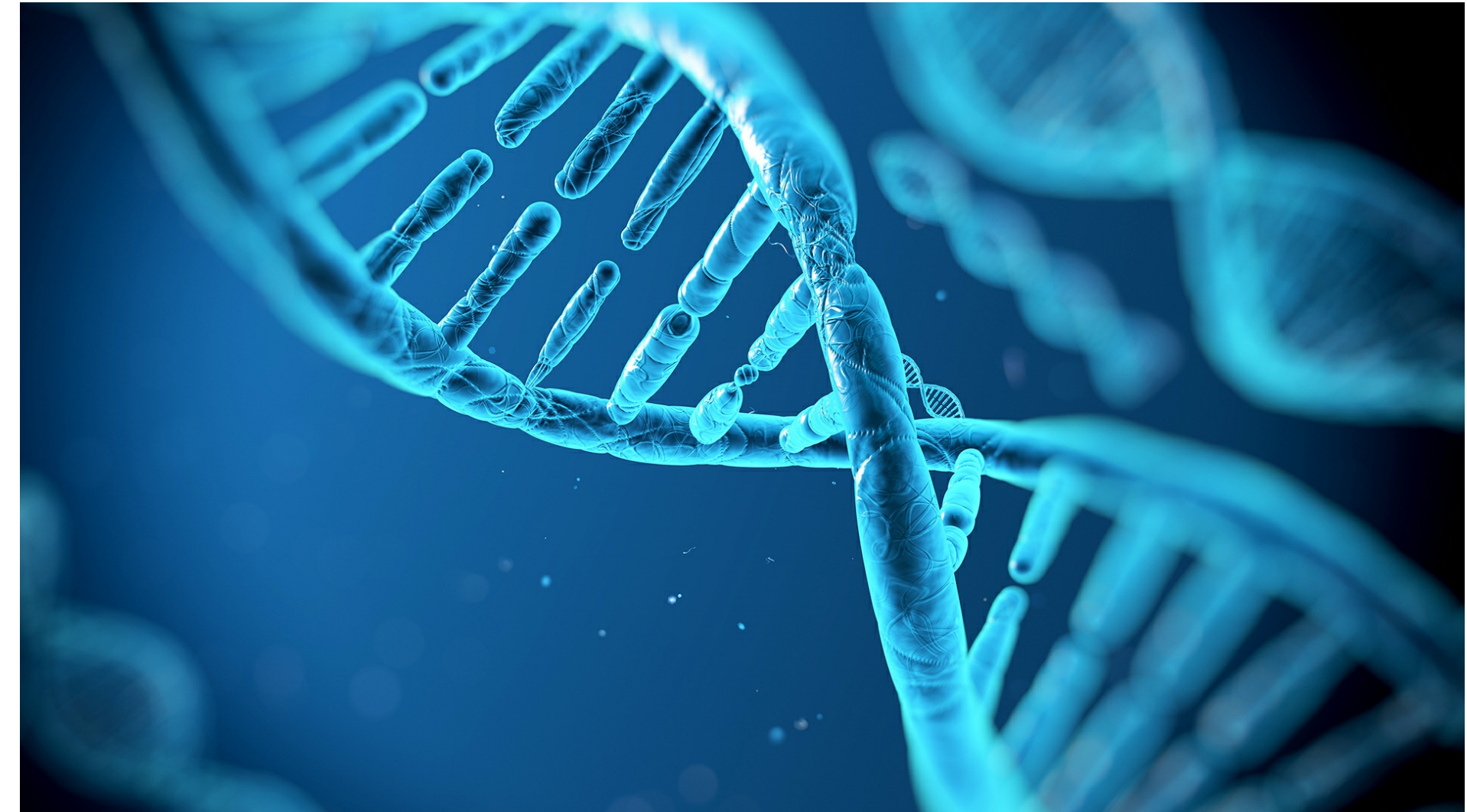
```
%message {  
  border: 1px solid #ccc;  
  padding: 10px;  
  color: #333;  
}  
.success {  
  @extend %message;  
  border-color: green;  
}  
.error {  
  @extend %message;  
  border-color: red;  
}
```

## Rezultat css

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

# Dziedziczenie

- Twórcy Sassa (Hampton Catlin, Natalie Weizenbaum, Chris Eppstein) twierdzą, że dziedziczenie to jedna z najbardziej przydatnych funkcji tego języka.
- W praktyce w Sassie właściwości zostaną wpisane tylko raz.
- Po skompilowaniu oba selektory będą miały współdzielone właściwości.
- Te, które się różnią, wygenerują osobne, specyficzne selektory.



Czas na zadania

# Listy i Mapy

# Listy

- Listy to zbiór dowolnych elementów.
  - Listy nie są przetwarzane na CSS, gdyż nie mają bezpośredniego przełożenia na ten język.
- Listy bywają trudne do zrozumienia, ponieważ w Sassie nie ma zbyt restrykcyjnych zasad podczas ich tworzenia np., elementy mogą być rozdzielone przecinkami ale nie muszą. Elementy również nie muszą mieć apostrofów/cudzysłowia, ale należy uważać przy definiowaniu dłuższych zdań. Zobacz przykład na następnej stronie.
  - Indeks zaczyna się od 1 nie od 0.



# Listy

## Przykład listy w Sassie

```
$list: #000, #f00, #0f0;
```

```
$myString: "Sass is awesome";  
length($myString); /* 1 */
```

A gdyby użytkownik usunął cudzysłów?

```
$myString: Sass is awesome;  
length($myString); /* 3 */
```

Zobacz na funkcje wbudowane dla list:

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

# Mapy (klucz / wartość)

- Mapy to zestaw kluczy i ich wartości, klucza używamy do odczytywania wartości.
- Mapy nie są przetwarzane na CSS, gdyż nie mają bezpośredniego przełożenia na ten język.
- W Sassie muszą być zapisane w nawiasach.

Zobacz na funkcje wbudowane dla map:

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

## Przykład mapy w Sassie

```
$map: (key1: value1,  
       key2: value2,  
       key3: value3);
```

# Stosowanie map

Mapy są przydatne do przechowywania zbiorów zmiennych, np. kolorów.

## Składnia scss

```
$colors: (  
  header: #b06,  
  text: #334,  
  footer: #666777  
);  
.header {  
  background-color: map-get($colors, header);  
}
```

## Rezultat css

```
.header {  
  background-color: #b06;  
}
```



Czas na zadania

# Operatory i operacje

# Operacje

## Operacje i operatory

Sass pozwala na wykonanie wielu operacji matematycznych:

- Dodawanie (  $+$  ),
- Odejmowanie (  $-$  ),
- Mnożenie (  $*$  ),
- Dzielenie (  $/$  ),
- Dzielenie modulo (  $\%$  ).

# Operatory + -

Operatory te mogą być użyte do obliczeń matematycznych (również na kolorach). Operator + może być również użyty do łączenia tekstu.

## Składnia scss

```
p {  
  margin: 7px - 4px auto;  
  color: #010203 + #040506;  
  cursor: e + -resize;  
}
```

## Rezultat css

```
p {  
  margin: 3px auto;  
  color: #050709;  
  cursor: e-resize;  
}
```

# Operator mnożenia \*

Sass pozwala mnożyć liczby i kolory.

## Składnia scss

```
div {  
  $baseline: 24px;  
  margin: 10px * 5;  
  padding: $baseline * 2;  
  color: #010203 * 2; /* 01 * 2, 02 * 2, 03 * 2 */  
}
```

## Rezultat css

```
div {  
  margin: 50px;  
  padding: 48px;  
  color: #020406;  
}
```

# Przykłady dzielenia

## Składnia scss

```
p {  
  /*Zwykły CSS, brak dzielenia*/  
  font: 10px/8px;  
  /*Występuje zmienna, wykona dzielenie*/  
  $width: 1000px;  
  width: $width/2;  
  /*Występuje funkcja, wykona dzielenie*/  
  width: round(1.5)/2;  
  /*Ujęte w nawiasy, wykona dzielenie*/  
  height: (500px/2);  
  height: (500px/2px);  
  /*Razem z +, wykona dzielenie*/  
  margin-left: 5px + 8px/2px;  
}
```

## Rezultat css

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

# Jednostki

- Sass zachowuje jednostki w trakcie operacji i podobnie jak w matematyce operacje muszą być przeprowadzone na zgodnych jednostkach.
- **Nie można dodać do siebie wartości określonych np. w pikselach i emach.**
- Wynikiem mnożenia jednostek będzie ich kwadrat, np.:

**10px \* 10px** zwróci wynik **100px \* px**, co nie jest prawidłową jednostką CSS.

- Jednostki w wyniku dzielenia skrócą się, np.:  
**10px/2px** zwróci **5**.

# Nawiasy okrągłe ( )

## Składnia scss

```
p {  
  width: 1em + (2em * 3);  
}
```

W celu określenia kolejności wykonywania operacji można użyć nawiasów okrągłych.

## Rezultat css

```
p {  
  width: 7em;  
}
```



# Interpolacja

# Interpolacja

Mamy wpływ na nazwy właściwości CSS przez wstawienie zmiennej w klamry i poprzedzenie jej znakiem #.

`#{$zmienna}`

## Składnia scss

```
@mixin border($color, $side) {  
  border-#{$side}-color:  
    $color;  
}  
.box {  
  @include border(red, bottom);  
}
```

## Rezultat css

```
.box {  
  border-bottom-color: red;  
}
```

# Logika

# @if

## Składnia scss

```
$type: monster;
p {
  @if $type == ocean {
    color: blue;
  } @else if $type == matador {
    color: red;
  } @else if $type == monster {
    color: green;
  } @else {
    color: black;
  }
}
```

## Rezultat css

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

- Sass pozwala na wstawianie definicji stylu na podstawie spełnionego warunku.
- Służy do tego metoda **@if**.
- Może być również połączona z **@else if**.

# @for

Metoda **@for** pozwala na powtórzenie stylu określoną liczbę razy.

```
@for $var from <start> through <end>
```

```
@for $var from <start> to <end>
```

## Składnia scss

```
@for $i from 1 through 3 {  
  .item-#{ $i } { width: 2em * $i; }  
}
```

- Jeśli użyjemy **through** – wartości i są wliczane.
- Jeśli użyjemy **to** – wartość jest pomijana.
- **\$var** może być dowolną zmienną.

## Rezultat css

```
.item-1 {  
  width: 2em;  
}  
.item-2 {  
  width: 4em;  
}  
.item-3 {  
  width: 6em;  
}
```

# @for

Metoda **@for** pozwala na powtórzenie stylu określoną liczbę razy.

```
@for $var from <start> through <end>
```

```
@for $var from <start> to <end>
```

## Składnia scss

```
@for $i from 1 through 3 {  
  .item-#{ $i } { width: 2em * $i; }  
}
```

**#{ \$i }** - Interpolacja.

- Jeśli użyjemy **through** – wartości i są wliczane.
- Jeśli użyjemy **to** – wartość jest pomijana.
- **\$var** może być dowolną zmienną.

## Rezultat css

```
.item-1 {  
  width: 2em;  
}  
.item-2 {  
  width: 4em;  
}  
.item-3 {  
  width: 6em;  
}
```

# @each

- Metoda **@each** jest podobna do metody **@for**.
- Różnicą jest zbiór, na którym jest wykonywana.
- W przypadku **@each** jest to lista.

Jej formuła to:

```
@each $var in <lista> {  
}
```

# Przykład @each

## Składnia scss

```
@each $animal in puma, sea-slug, egret, salamander {  
  .#{$animal}-icon {  
    background-image: url('/images/#{$animal}.png');  
  }  
}
```

## Rezultat css

```
.puma-icon {  
  background-image: url("/images/puma.png");  
.sea-slug-icon {  
  background-image: url("/images/sea-slug.png");  
.egret-icon {  
  background-image: url("/images/egret.png");  
.salamander-icon {  
  background-image: url("/images/salamander.png");  
}
```



# @while

**@while** podobnie jak w przypadku **@for** powtarza dany zestaw styli.

## Składnia scss

```
$i: 6;
@while $i > 0 {
  .item-#{ $i } {
    width: 2em * $i;
  }
  $i: $i - 2;
}
```

**@while** pozwala na zatrzymanie pętli dzięki zastosowaniu warunku.

## Rezultat css

```
.item-6 { width: 12em; }
.item-4 { width: 8em; }
.item-2 { width: 4em; }
```

A diagram showing the variable `$i` with a blue arrow pointing to a list of values: `1: 6`, `2: 4`, and `3: 2`. The numbers 6, 4, and 2 are in orange, while the others are in blue.

Czas na zadania

# Funkcje

# Funkcje w Sassie

Sass ma **gotowe funkcje** do wykonania na kolorach, transparentności, tekście, liczbach, listach, mapach i selektorach.

- Możliwe jest definiowanie własnych funkcji.
- Funkcje mogą być użyte dla dowolnej właściwości oraz w dowolnym kontekście.
- Funkcje mają dostęp do wszystkich globalnych zmiennych.
- Podobnie jak **mixin** mogą przyjmować argumenty.

Zbiór gotowych funkcji Sassa znajdziesz w dokumentacji:

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

# Definiowanie własnych funkcji

Zwrócenie wartości funkcji wymaga użycia **@return**.

## Składnia scss

```
@function test($width) {  
  @return $width/2;  
}  
#sidebar {  
  width: test(240px);  
}
```

## Rezultat css

```
#sidebar {  
  width: 120px;  
}
```

# Przykładowe funkcje kolorów

## Składnia scss

```
button {  
  $color: #5a57ff;  
  $color-alt: #ffc700;  
  color: $color-alt;  
  background: lighten($color, 20%);  
  &:hover {  
    color: lighten($color, 100%);  
    background: grayscale($color-alt);  
  }  
}
```

## Rezultat css

```
button {  
  color: #ffc700;  
  background: #bebdff;  
}  
button:hover {  
  color: white;  
  background: gray;  
}
```

Więcej przykładów znajdziesz tutaj:

<http://jackiebalzer.com/color>

Czas na zadania

**Dlaczego warto  
pracować z  
Sasseem?**



# Dlaczego warto pracować z Sassesem?

- Początki CSS-a sięgają połowy lat 90. Nie przewidziano wtedy pewnych zastosowań deklaracji, które są używane obecnie. Na przykład **float** miał wtedy służyć do pozycjonowania obrazków względem tekstu.
- Struktura CSS-a jest płaska i prosta.
- Sass rozszerza język CSS o możliwości dostępne w językach wykorzystujących paradygmat programowania obiektowego.
- Takie podejście ułatwia tworzenie oraz utrzymanie kodu.

## Możliwości Sassa

- Definiowanie zmiennych.
- Zagnieżdżanie właściwości i selektorów.
- **Mixins** – definiowanie grup deklaracji.
- Pliki cząstkowe ( **partials** ) i importowanie ( **@import** ).
- Dziedziczenie ( **@extend** ).
- Operacje i operatory **+**, **-**, **\***, **/**, **%**.
- Wbudowane funkcje np. manipulacja kolorami.
- Definiowanie własnych funkcji.
- Logika ( **@if**, **@for**, **@each**, **@while** ).

# Odpowiednie użycie Sassa

With great power comes great  
responsibility

# Pomocne wskazówki

- Dobrze przemyśl podejście do nazewnictwa klas oraz struktury kodu.
  - Np. użyj OOCSS (Object-Oriented CSS).
  - Odwzorowanie struktury kodu HTML nie zawsze jest najlepszym pomysłem.
  - Używaj mixinów z argumentem, np. mixin definiujący szerokość elementu na podstawie liczby kolumn, jeśli układ elementów jest oparty o grid.
- Zobacz, jak wygląda kod wynikowy – będzie Ci łatwiej uniknąć nadmiarowych linii kodu, np. powtarzania deklaracji stworzonych za pomocą **@extend**.
  - Pamiętaj - dla przeglądarki liczy się wydajność, a nie czytelność.
  - Sass jest tylko narzędziem, to Ty decydujesz o tym, jak wykorzystać je optymalnie.

# Przydatne linki

- <http://sass-lang.com/guide>
- [http://sass-lang.com/documentation/file.SASS\\_REFERENCE.html](http://sass-lang.com/documentation/file.SASS_REFERENCE.html)
- <http://abookapart.com/products/sass-for-web-designers>
- <http://thesassway.com>
- <http://jackiebalzer.com/color>

# Aplikacje

- Compass.app - OS X, Windows, Linux,
- CodeKit - OS X,
- Hammer - OS X,
- Koala - OS X, Windows, Linux,

- LiveReload - OS X, Windows (wersja alfa aplikacji),
- Prepros - OS X, Windows, Linux,
- Scout - OS X, Windows.