

# 6. Sass (SCSS), czyli CSS profesjonalnie



## Wyzwania:

- Dowiesz się, czym są preprocesory CSS i w jaki sposób rozszerzają funkcjonalność CSS.
- Poznasz jeden z najpopularniejszych preprocesorów, którym jest Sass.
- Nauczysz się tworzenia funkcji rodem z JS w stylach.
- Dowiesz się, jak rozpocząć pracę z Sassem na swoim środowisku developerskim.

Mixiny

## 6.1. Preprocesor CSS - co to jest?



### Preprocesory

Poznaliśmy już wiele reguł CSS i mogłoby się wydawać, że możliwości, jakie nam dają są ogromne. Jest to prawdą, jednak jeżeli się przyjrzymy (i popracujemy nad większymi projektami), to okaże się, że pewne rzeczy bywają żmudne i czasochłonne.

Wyobraźmy sobie, że w celu zapewnienia spójności wizualnej stosujemy w wielu miejscach ten sam kolor, a później zdecydujemy się go zmienić. Musimy odnaleźć wszystkie miejsca, gdzie ten kolor występuje, a następnie go podmienić.

Dużo prościej byłoby zdefiniować sobie zmienną, np. kodilla-green, i wstawiać ją jako wartość do każdej reguły, w której używamy tego koloru. Modyfikacja koloru odbywałaby się wtedy tylko w jednym miejscu.



Utrzymanie rosnącej liczby kodu CSS przy coraz większym projekcie bywa trudne. Łatwiej byłoby podzielić go na kilka plików, każdy odpowiedzialny np. za jeden komponent tak, by połączyć go w całość.

Możemy teoretycznie dodać kilka plików CSS do jednego dokumentu HTML, ma to jednak kiepski wpływ na prędkość ładowania strony.

Odpowiedzią na takie problemy (i wiele innych) jest właśnie **preprocesor**.

Rozszerza on CSS o nowe możliwości. Możemy korzystać z takich udogodnień jak np. operacje logiczne czy importowanie plików. Działa to w ten sposób, że tworzymy plik, w którym piszemy za pomocą składni danego preprocesora, a on go później **kompiluje**, tj. przerabia na zwykły CSS.

W ten sposób więcej czasu poświęcamy na właściwą pracę nad projektem, a mniej na walkę z czystym CSS :) Dzięki temu możemy też napisać znacznie mniej kodu, preprocesor zrobi brudną robotę za nas.

---

## Popularne preprocesory

---



Do najpopularniejszych preprocesorów CSS należą: Sass, Less oraz Stylus. Jeżeli patrzymy na same możliwości, jakie nam dają, to ciężko wskazać tu jednoznacznego zwycięzcę.

Różnice są nieznaczne i sprowadzają się głównie do preferencji odnośnie sposobu zapisu, chociaż bardzo często te preprocesory są prawie identyczne.

Jest jednak kilka argumentów przemawiających na korzyść Sassa, na podstawie których zdecydowaliśmy, że przeszkolimy Was z używania właśnie tego preprocesora.

Jednak zanim do tego przejdziemy, opiszemy krótko najpopularniejsze procesory.



# Less



Less to stosunkowo świeży preprocesor, którego twórcy od samego początku stawiali silny nacisk na związek z CSS i łatwość nauki. Niestety zrobili to kosztem rozbudowania - w porównaniu do Sassa, Less jest o wiele łatwiejszy do nauki, lecz oferuje mniejszą funkcjonalność i jego znajomość jest rzadziej wymagana przez pracodawców. Nie umożliwia też omijania nawiasów i średników, w przeciwieństwie do Sassa i Stylusa.

Poniżej przykład składni kodu przed kompilacją oraz po.

Kod Less przed kompilacją:

```
style.less
1  @green: #55ee70;
2
3  .border-radius(@n) {
4    -webkit-border-radius: @n;
5    -moz-border-radius: @n;
6    border-radius: @n;
7  }
8  form input[type=button] {
9    .border-radius(5px);
10 }
11
12 body {
13   color: @green;
14 }
```

Kod Less po kompilacji:



```
style.css
1  form input[type=button] {
2    -webkit-border-radius: 5px;
3    -moz-border-radius: 5px;
4    border-radius: 5px;
5  }
6
7  body {
8    color: #55ee70;
9  }
```

## Sass



**Sass** to preprocesor stawiający na nowoczesność i metodę DRY (Don't Repeat Yourself). Jest on bardziej skomplikowany i rozbudowany od Lessa, co z jednej strony oszczędza developerom sporo pracy, a z drugiej strony zapewnia im mnóstwo rzeczy do nauki.

Ogromną zaletą Sassa jest to, że obsługuje dwie wersje składni - składnia Sass oraz SCSS (Sassy CSS). Różnią się one tym, że SCSS wymaga używania nawiasów oraz średników, tak jak CSS - jest to ukłon w stronę osób przywiązanych do tradycyjnej składni.

Sass zawiera dużo wbudowanych funkcji, które umożliwiają chociażby korzystanie z podstawowych działań matematycznych czy konstrukcji warunkowych. Swoją składnią często przypominają syntax JavaScript.

Poniżej prezentacja kodu Sass przed i po kompilacji.

Kod Sass przed kompilacją:

```
style.sass
1 $green: #55ee70
2
3 =border-radius($n)
4   -webkit-border-radius: $n
5   -moz-border-radius: $n
6   border-radius: $n
7
8 form input[type=button]
9   +border-radius(5px)
10
11 body
12   color: $green
```

Kod Sass po kompilacji:

```
style.css
1 form input[type=button] {
2   -webkit-border-radius: 5px;
3   -moz-border-radius: 5px;
4   border-radius: 5px;
5 }
6
7 body {
8   color: #55ee70;
9 }
```

## Stylus



**Stylus** to najmłodszy z omawianych przez nas preprocesorów, a jednocześnie najbardziej skomplikowany i rozbudowany. Stylus zapewnia nam ponad 60 wbudowanych funkcji i wiele autorskich rozwiązań ułatwiających pracę.

Niestety, umożliwiając też maksymalne skracanie i upraszczanie wielu aspektów, jednocześnie pozwala na mieszanie różnych podejść, co czyni go nieodpowiednim i trudnym dla początkujących.

Przez zbyt dużą dowolność Stylus jest niespójny i mniej czytelny od Sassa, co jest ogromną wadą tego preprocesora z punktu widzenia początkującego koderka.



Dalej pokazujemy próbki kodu Stylusa przed i po kompilacji.

Kod Stylusa przed kompilacją:

```
style.styl
1  green = #55ee70
2
3  border-radius(n)
4    -webkit-border-radius n
5    -moz-border-radius n
6    border-radius n
7
8  form input[type=button]
9    border-radius(5px)
10
11 body
12   color green
```

Kod Stylus po kompilacji:

```
style.css
1  form input[type=button] {
2    -webkit-border-radius: 5px;
3    -moz-border-radius: 5px;
4    border-radius: 5px;
5  }
6
7  body {
8    color: #55ee70;
9  }
```

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

✓ Zapoznałem(a)m się!

## 6.2. Właściwości i składnia



### Podstawy

W przypadku **Sass** mamy do dyspozycji dwa rodzaje składni: **SCSS** oraz **Sass**. Aby wybrać jedną z nich, zapisujemy nasz kod w pliku z rozszerzeniem `.scss` lub `.sass`. Podstawowa różnica sprowadza się do tego, jak piszemy nasze reguły.



W przypadku składni Sass reguły budowane są z formatowaniem, jakiego nauczyliśmy się dotychczas, z tą jednak różnicą, że omijamy średniki oraz nawiasy klamrowe. Istnieją też drobne różnice przy korzystaniu z poszczególnych funkcji.

SCSS reprezentuje nieco odmienne podejście - jego składnia wygląda dokładnie tak, jak składnia CSS i nie obsługuje on pomijania średników oraz nawiasów klamrowych.

W dokumentacji Sass uwzględnione są obie składnie. Gdy korzystamy z preprocesora, musimy pamiętać o tych szczegółach - jeśli wykorzystywana przez nas funkcja nie działa, sprawdźmy, czy korzystamy z właściwej składni.

Na potrzeby modułu **będziemy korzystać ze składni SCSS**.

### Sam zdecyduj jaka składnia!

O tym jakiej składni użyjemy w naszym projekcie decyduje rozszerzenie pliku zawierającego nasze style. Odpowiednio:

- **SASS** - .sass
- **SCSS** - .scss

W treści modułu pojawi się jeszcze określenie "sass". Będzie oznaczać nazwę narzędzia (technologii), a nie składni. Wszędzie trzymamy się składni SCSS.

## Zmienne

Zmienne to nic innego jak pewne "pudełka", w których możemy zamknąć nasze dane. Podręcznikowym przykładem zastosowania zmiennych jest definiowanie kolorów i czcionek. Zobaczmy przykład poniżej:

```
$fonts: "Montserrat", sans-serif;
$kodilla-green: #53B863;

span {
  font: 100% $fonts;
  color: $kodilla-green;
}

h1 {
  font: 24px/36px $fonts;
  background-color: $kodilla-green;
}
```



Na początku pliku określamy np. kolor lub fonty, a potem możemy przywoływać je wielokrotnie. W każdym miejscu, w którym podaliśmy jako wartość `$kodilla-green`, podczas kompilacji zostanie podstawiona wartość `#53B863`. Kiedy chcemy zmienić wartość koloru, nie musimy tego robić w całym pliku - wystarczy zmienić wartość zmiennej w miejscu, gdzie ją zdefiniowaliśmy.

Kod, który otrzymamy po kompilacji, będzie wyglądać następująco:

```
span {  
  font: 100% "Montserrat", sans-serif;  
  color: #53B863;  
}  
  
h1 {  
  font: 24px/36px "Montserrat", sans-serif;  
  background-color: #53B863;  
}
```

Pewną przyjętą praktyką jest, że zmienne globalne definiuje się na początku pliku, by mieć do nich łatwy dostęp i w razie potrzeby móc je szybko modyfikować.

## Przykład

HTML	SCSS	Result
	<pre>1 @import url(https://fonts.googleapis.com    family=Montserrat&amp;subset=latin,latin-ext); 2 3 \$fonts:    "Montserrat",    sans-serif; 4 \$kodilla-green:    #53B863; 5 6 span { 7   font: 100%    \$fonts; 8   color:</pre>	<p>lorem</p> <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Rem, commodi asperiores adipisci, deleniti exercitationem a vel accusamus, aliquid illo officiis consectetur molestias recusandae nesciunt suscipit rerum eius</p>
Resources	1x 0.5x 0.25x	Rerun

## Zagnieżdżanie





Kolejną przydatną funkcją, jaką daje nam Sass, jest łatwiejsze zagnieżdżanie selektorów. W przypadku czystego CSS budowa wielu zagnieżdżonych selektorów bywa czasochłonna.

```
nav ul {  
  margin: 0;  
  padding: 0;  
  list-style: none;  
}  
  
nav li {  
  display: inline-block;  
  width: 100px;  
  height: 50px;  
  text-align: center;  
}  
  
nav a {  
  display: block;  
  padding: 6px 12px;  
  text-decoration: none;  
  background-color: #2c3e50;  
}  
nav ul a:hover,  
nav ul a:focus {  
  background-color: #34495e;  
}
```

Preprocesor może nam nieco uprościć pracę. W Sass, zamiast pisać każdorazowo nowy zagnieżdżony selektor, możemy umieszczać w sobie elementy w podobny sposób, jak w HTML. Zobaczmy to na przykładzie. Rezultat taki sam, jak wyżej, możemy otrzymać przy użyciu Sass w następujący sposób:



```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  li {  
    display: inline-block;  
    width: 100px;  
    height: 50px;  
    text-align: center;  
  }  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
    background-color: #2c3e50;  
    color: white;  
  }  
}
```

Zagnieżdżając selektory w powyższy sposób, musimy pilnować wcięć w kodzie. Każdy kolejny poziom zagnieżdżenia składa się z jednego tabulatora lub dwóch znaków spacji.

Jeżeli chcemy dodać do zagnieżdżonych w ten sposób elementów np. pseudoselektor stanu **hover**, robimy to za pomocą znaku **&**.

Po dodaniu kodu odpowiedzialnego za **hover** powyższy fragment wygląda tak:



```
nav {  
  ul {  
    margin: 0;  
    padding: 0;  
    list-style: none;  
  }  
  li {  
    display: inline-block;  
    width: 100px;  
    height: 50px;  
    text-align: center;  
  }  
  a {  
    display: block;  
    padding: 6px 12px;  
    text-decoration: none;  
    background-color: #2c3e50;  
    color: white;  
    &:hover,  
    &:focus {  
      background-color: #34495e;  
    }  
  }  
}
```

Jak widać, taki kod jest dużo bardziej przejrzysty. Ponadto jest on zgodny z zasadą **DRY** (**Don't repeat yourself**).

## Przykład





## Importowanie

Korzystaliśmy już z reguły `@import` przy korzystaniu z Google Fonts. Sass rozszerza możliwości tych reguły o importowanie plików z rozszerzeniem `.sass` oraz `.scss`.

### Uwaga dot. edytora projektów

Ze względów technicznych tymczasowo w naszym edytorze projektów online nie można korzystać z importowania innych plików `.sass` oraz `.scss`.

Przećwicz stosowanie `@import` po zainstalowaniu kompilatora Sass na swoim komputerze (omawiamy ten temat później w tym module).

Jeśli pracujesz lokalnie (na swoim komputerze, a nie w edytorze online) nad zadaniem, które ma być oddane w edytorze projektów, wystarczy że skopiujesz zawartość wszystkich plików `.sass` oraz `.scss` do jednego pliku, tak aby nie było potrzeby używania `@import`.

Aby stworzyć taki fragment gotowy do importu, tworzymy w folderze plik z podkreślnikiem na początku, np. `_partials.sass`. Tam umieszczamy kod, który chcemy zaimportować. By następnie zaimportować zawartość pliku `_partials.sass` w docelowym pliku, tworzymy regułę:

```
@import 'partials';
```



Zwróć uwagę, że nie dodajemy tutaj pełnej nazwy pliku: pomijamy podkreślnik oraz rozszerzenie. Jeżeli w regule `@import` dodamy `url("...")`, to zadziała on normalnie.

Jaki jest sens tworzenia takiego systemu? Dzięki niemu możemy podzielić duży plik CSS na kilka mniejszych modułów. Przy rozbudowanych projektach pozwala to na uporządkowanie naszego kodu. Ponadto, jeżeli mamy blok kodu, z którego często korzystamy, np. reset lub grid, wystarczy zapisać go do pliku i potem zaimportować.

Istnieje wiele dodatków do Sass, które możemy dodawać właśnie przez importowanie. Jednym z najpopularniejszych jest np. gotowy grid o nazwie Susy. Dodajemy go do projektu za pomocą jednej linii:

```
@import 'susy';
```

---

## Mixiny

---

Niektóre z właściwości CSS, zwłaszcza te korzystające z prefiksów, bywają bardzo rozległe. Często w ich przypadku bywa tak, że dla jednej właściwości musimy skopiować duży blok kodu, zmieniając tylko kilka parametrów. Przychodzi nam tutaj z pomocą Sass i mixiny.

Mixiny to tak jakby bardziej rozbudowane zmienne. Definiujemy je na początku, by następnie je przywoływać. Różnica jest taka, że możemy określić kilka wyjściowych parametrów. W przykładzie poniżej definiujemy właściwość `border-radius`, korzystając z prefiksów. Jak widzisz, korzystamy tam też z parametru `$radius`.

```
@mixin border-radius($radius) {  
  -webkit-border-radius: $radius;  
  -moz-border-radius: $radius;  
  -ms-border-radius: $radius;  
  border-radius: $radius;  
}
```

Po takim zdefiniowaniu mixinu przywołujemy go wewnątrz reguły, podając w nawiasie parametr, tak jak poniżej:

```
.box {  
  @include border-radius(10px);  
}  
.circle {  
  @include border-radius(50%);  
}
```

Po kompilacji otrzymamy zdefiniowane reguły **border-radius** wraz ze wszystkimi niezbędnymi prefiksami:

```
.box {  
  -webkit-border-radius: 10px;  
  -moz-border-radius: 10px;  
  -ms-border-radius: 10px;  
  border-radius: 10px;  
}  
.circle {  
  -webkit-border-radius: 50%;  
  -moz-border-radius: 50%;  
  -ms-border-radius: 50%;  
  border-radius: 50%;  
}
```

Dzięki takiemu zabiegowi nie musimy pisać aż tyle kodu, przez co zmniejszamy prawdopodobieństwo pomyłki i nie napracujemy się :)

## Przykład



HTML SCSS Result EDIT ON CODEPEN

```
1 @mixin border-radius($radius) {  
2   -webkit-border-radius:  
   $radius;  
3   -moz-border-radius: $radius;  
4   -ms-border-radius: $radius;  
5   border-radius: $radius;  
6 }  
7  
8 .box, .circle {  
9   width: 100px;  
10  height: 100px;  
11  background-color: #55ee70;  
12  margin: 10px auto;  
13 }  
14  
15 .box {  
16   @include border-radius(10px);  
17 }
```

Resources 1x 0.5x 0.25x Rerun

## Rozszerzanie reguł

Jednym z najbardziej przydatnych narzędzi, jakie daje nam Sass, jest rozszerzanie reguł za pomocą reguły `@extend`. Kiedy piszemy stronę internetową, często zdarza się że chcemy, by jedna klasa zawierała w sobie wszystkie reguły innej klasy oraz kilka unikalnych reguł dla niej samej.

Pracując w czystym CSS, rozwiązalibyśmy to, tworząc klasę ogólną oraz bardziej szczegółową. Wówczas musimy jednak pamiętać każdorazowo o dopisywaniu klasy ogólnej, co może czasami spowodować błędy. Rozwiązaniem jest właśnie reguła `@extend`. Dzięki takiemu rozwiązaniu rozszerzymy reguły jednej klasy o reguły innej. Zobaczmy to na przykładzie.





We fragmencie poniżej chcemy, aby klasy **success**, **error** oraz **warning** miały pewne reguły wspólne, zaś różniły się jedynie kolorem obramowania. W tym celu stworzyliśmy klasę wspólną klasę **.message**, zaś do poszczególnych klas dopisaliśmy regułę **@extend**.

```
.message {  
  border: 3px solid #ccc;  
  padding: 10px;  
  color: #333;  
  width: 200px;  
  text-align: center;  
  height: 50px;  
  margin: 0 auto;  
}  
  
.success {  
  @extend .message;  
  border-color: green;  
}  
  
.error {  
  @extend .message;  
  border-color: red;  
}  
  
.warning {  
  @extend .message;  
  border-color: yellow;  
}
```





W ten sposób tworzymy zrozumiały kod. Jeżeli chcemy stworzyć kolejne rodzaje wiadomości, to po prostu dodajemy nową klasę i rozszerzamy plik o regułę `@extend`. Preprocesor zrobi resztę.

---

## Operatory

---

Często podczas budowy layoutu musimy dokonać pewnych obliczeń matematycznych. W sytuacji gdy później ten layout modyfikujemy, obliczenia muszą być powtórzone, tylko z innymi danymi. Dzięki Sassowi możemy wykonywać operacje matematyczne wewnątrz pliku, co zapewne nie raz oszczędzi nam pomyłek. Do dyspozycji mamy pięć podstawowych operatorów matematycznych:

- dodawanie `+`
- odejmowanie `-`
- mnożenie `*`
- dzielenie `/`
- modulo `%`

Przy dokonywaniu działań matematycznych musimy pamiętać o odpowiednich jednostkach. Jeżeli dodajemy do siebie dwa składniki w pikselach, to ich suma również będzie wyrażona w pikselach, lecz jeżeli spróbujemy dodać do siebie składniki wyrażone w `px` i `em`, to spowoduje to błąd. Podobnie nie można też mnożyć przez siebie dwóch czynników wyrażonych w pikselach gdyż wynik będzie wyrażony w pikselach do kwadratu.

Rozważmy zastosowanie operatorów w kontekście budowy layoutu. Załóżmy, że dostaliśmy od grafika projekt z kontenerem o szerokości `100%` i dwóch kolumn w środku. Kolumny mają mieć szerokość zależną od szerokości kontenera i marginesów, dlatego też możemy zbudować nasze reguły w następujący sposób:



```
$columns: 2;
$margin: 2%;

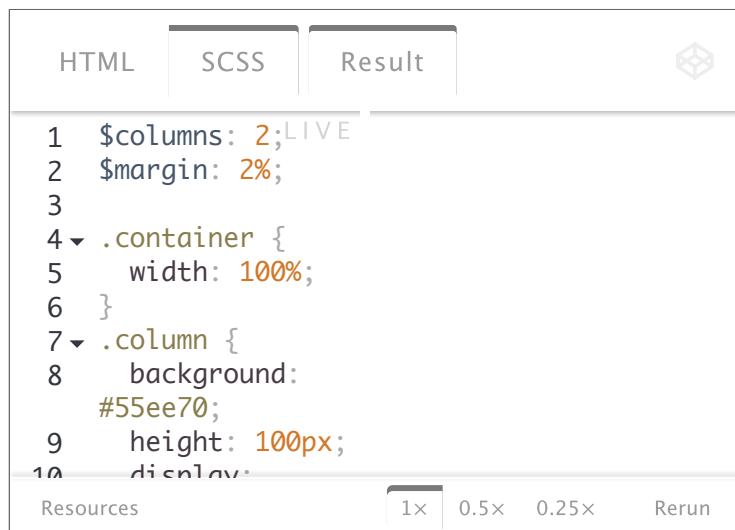
.container {
  width: 100%;
}
.column {
  background: #55ee70;
  height: 100px;
  display: block;
  float: left;
  width: (100% / $columns) - ($margin * 2);
  margin: $margin;
}
```

Po kompilacji otrzymamy następujący wynik:

```
.container {
  width: 100%;
}

.column {
  background: #55ee70;
  height: 100px;
  display: block;
  float: left;
  width: 46%;
  margin: 2%;
}
```

## Przykład



## Zadanie: Przeniesienie projektu na Sassa

W submodule 4.4 zajmowaliśmy się dodaniem responsywności do stworzonej wcześniej strony - pora wykorzystać poznanego właśnie Sassa i uporządkować swój kod z jego pomocą.

1. Stwórz pliki *style.scss*.
2. Przenieś kod HTML z wspomnianego wcześniej projektu do pliku *index.html*.
3. Skopiuj kod CSS do pliku *style.scss*.
4. Popraw kod tak, by był zgodny ze składnią SCSS.
5. Wykorzystaj zagnieżdżanie zamiast rozbudowanych selektorów CSS w miejscach, gdzie jest to możliwe, żeby Twój kod był bardziej czytelny.
6. Znajdź powtarzające się kolory i stwórz zmienne, którymi je zastąpisz.
7. Z pewnością masz w kodzie jakieś powtarzające się klasy - spróbuj uporządkować je z wykorzystaniem reguły **@extend** - krótszy kod jest bardziej czytelny dla developera.
8. Jeżeli wykorzystujesz reguły CSS mające swoje prefixy (np. **-webkit**, **-moz**), to zastąp je odpowiednimi mixinami.

Aby skorzystać z Sass w edytorze projektów Kodilla, wystarczy dodać nowy plik (*Plik > Nowy plik*). Przy zapisie pliku z rozszerzeniem **.scss** nastąpi jego automatyczna kompilacja do pliku o analogicznej nazwie z rozszerzeniem **.css** - przykładowo plik **style.scss**, przy zapisie, zostanie skompilowany do **style.css**.

Do zapisu konieczne jest jedynie, by plik **style.css** był już utworzony w katalogu projektu. Jeżeli plik taki nie zostanie odnaleziony, to zostaniemy o tym poinformowani alertem.

Możesz śmiało zapisywać swój plik **.scss** zanim jeszcze przerobisz go w całości - wyskoczy wtedy powiadomienie o błędzie, ale plik tak czy inaczej zostanie zapisany, więc nie utracisz swoich zmian :)

Podgląd zadania

Przejdź do projektu ✓



## 6.3. Instalacja Sass



Istnieje wiele różnych sposobów, dzięki którym można uruchomić Sassa i korzystać z niego. Do dyspozycji mamy programy z graficznym interfejsem użytkownika, oprogramowanie korzystające z wiersza poleceń oraz narzędzia do automatyzacji pracy takie jak Grunt lub Gulp.

W dokumentacji są przedstawione dwa podstawowe **sposoby instalacji Sass**. Jeżeli nie boisz się korzystania z wiersza poleceń (a nie ma się czego bać), możesz spróbować tej metody. Jeżeli nie jest to Twoje ulubione narzędzie lub nie czujesz się z nim pewnie, możesz skorzystać z darmowego programu Koala App, który umożliwi Ci kompilację kodu Sass (niezależnie od tego czy użyjesz składni sass czy scss).

Jednak z racji tego, że konsola bardzo się przydaje, polecamy oswoić się z tym narzędziem :)

Sass jest programem napisanym w języku Ruby, dlatego instalacja wymaga najpierw upewnienia się, że na naszym komputerze jest poprawnie zainstalowane i skonfigurowane środowisko tego języka - od tego zaczniemy ten fragment kursu.

---

## Windows

---

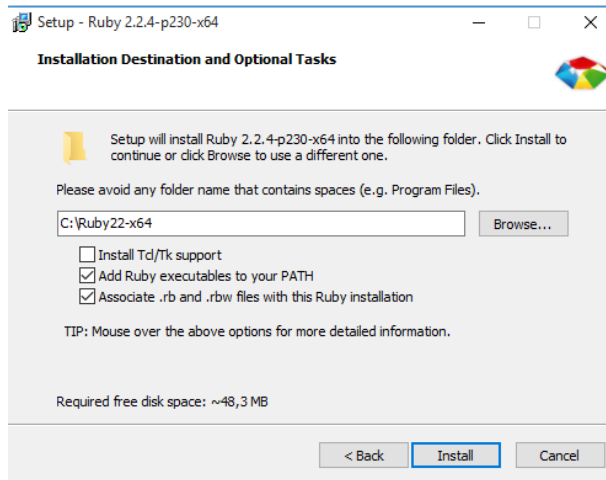
### Instalacja Ruby

W przypadku systemu Windows mamy do dyspozycji szybki instalator, który zrobi to automatycznie. Możemy go pobrać ze strony <http://rubyinstaller.org/>

Pobrany plik otwieramy, a następnie postępujemy zgodnie ze standardowymi krokami instalacji.

Jeżeli chcemy korzystać z Ruby przy użyciu wiersza poleceń, musimy pamiętać o **dodaniu Ruby do zmiennej systemowej PATH, więc zaznaczamy to podczas instalacji:**

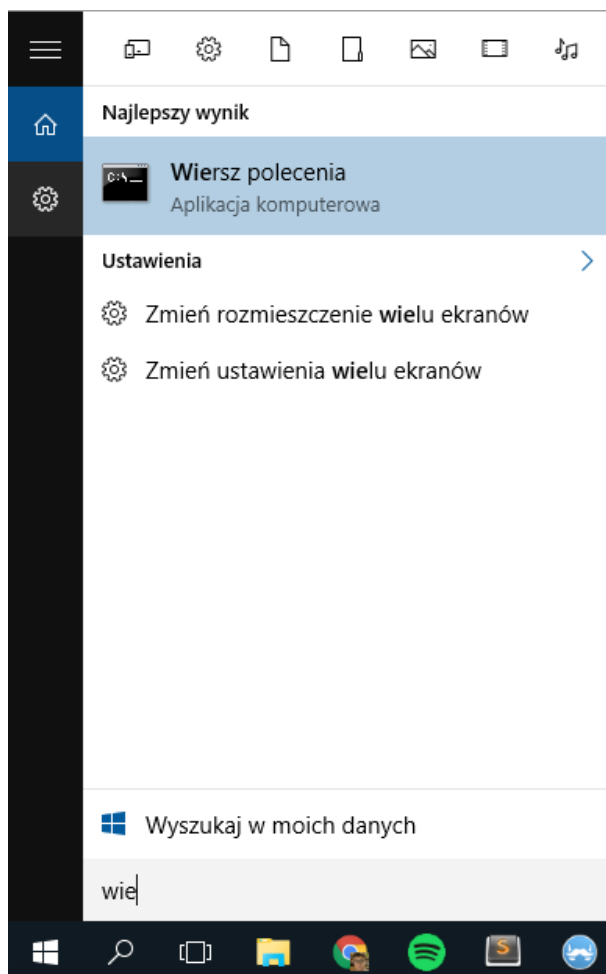





Po instalacji możemy skorzystać z menedżera pakietów RubyGems w wierszu poleceń.

## Instalacja Sassa - linia komend

Aby otworzyć wiersz poleceń, kliknij przycisk Start , a następnie wyszukaj **Wiersz poleceń**.




Uruchamiamy aplikację i wpisujemy polecenie: `gem install sass`

 C:\Windows\system32\cmd.exe


```
C:\Users\MB>gem install sass
```

To uruchomi automatyczną instalację.

 C:\Windows\system32\cmd.exe

```
C:\Users\MB>gem install sass
Fetching: sass-3.4.22.gem (100%)
Successfully installed sass-3.4.22
Parsing documentation for sass-3.4.22
Installing ri documentation for sass-3.4.22
Done installing documentation for sass after 11 seconds
1 gem installed
```

Jeśli wszystko się powiodło, możemy sprawdzić zainstalowaną wersję za pomocą polecenia **sass -v**:

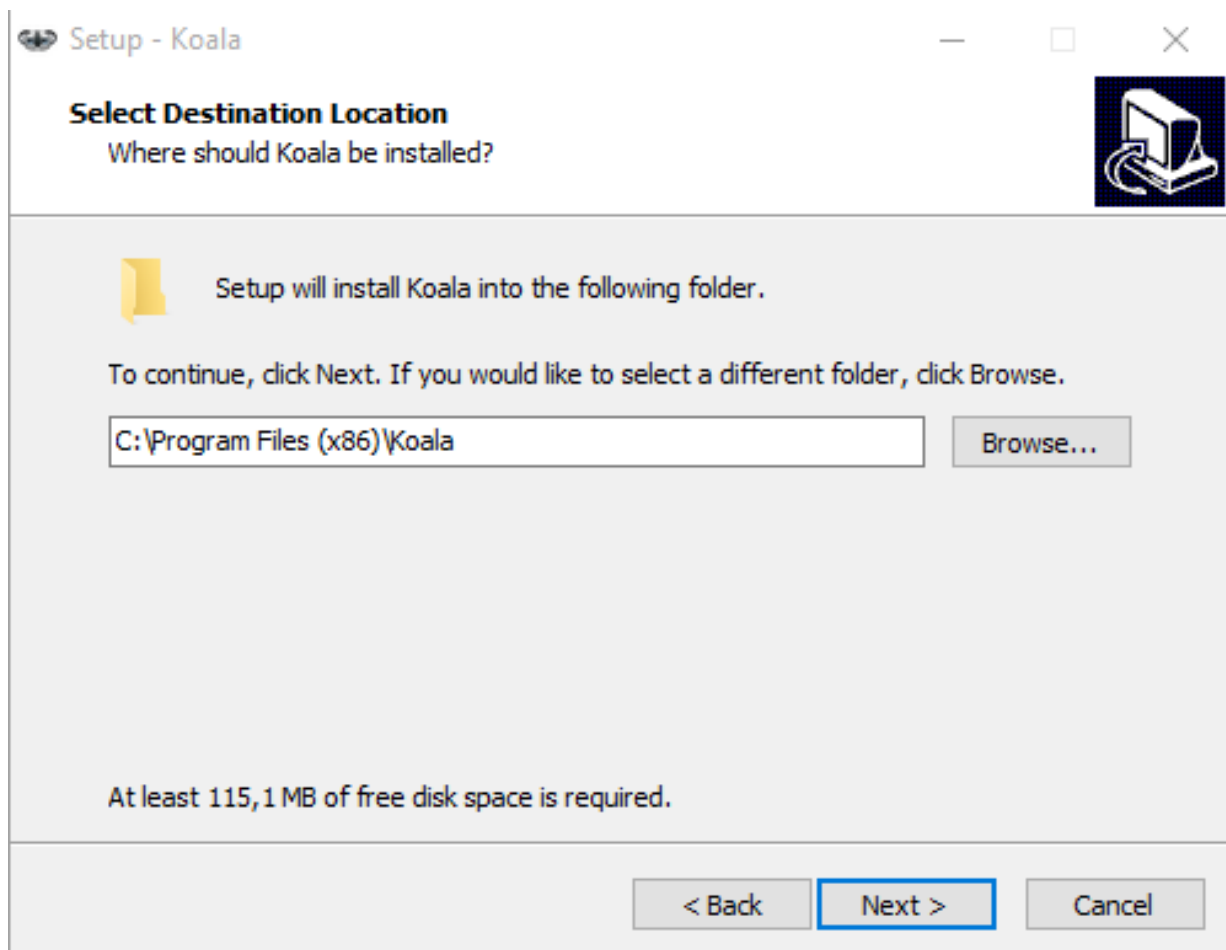
 C:\Windows\system32\cmd.exe

```
C:\Users\MB>sass -v
Sass 3.4.22 (Selective Steve)
```

## Instalacja Koala App

Jeśli nie lubisz korzystać z konsoli, dobrą alternatywą jest Koala App. Możesz ją pobrać za darmo pod adresem <http://koala-app.com>.

Proces instalacji nie jest skomplikowany - postępujemy tak, jak w przypadku każdego innego programu - uruchamiamy pobrany ze strony plik *koala\_2.0.4\_setup.exe* i postępujemy zgodnie z instrukcjami zawartymi w instalatorze.



---

## Mac OS X

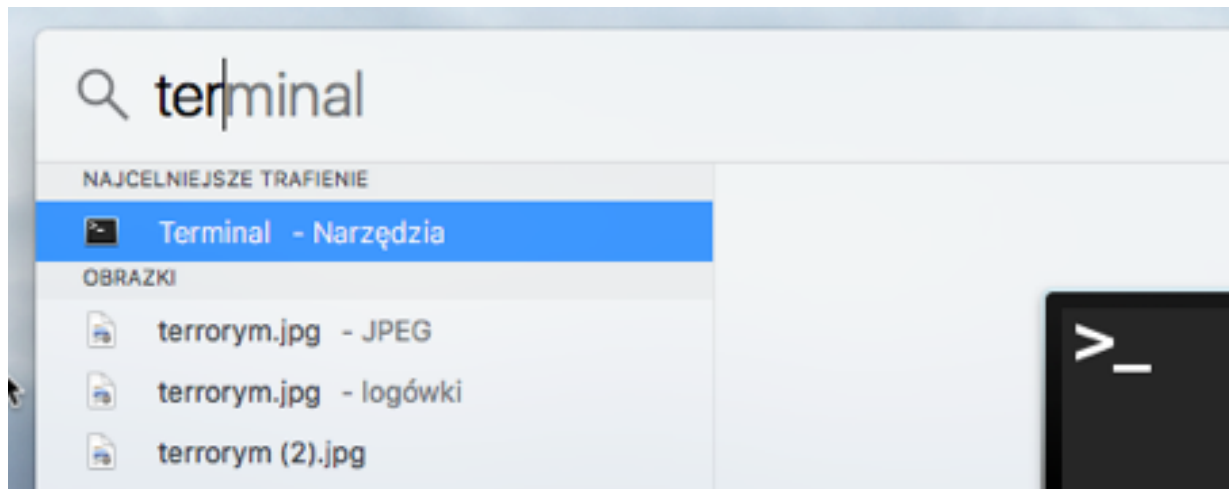
---

### Instalacja Ruby

Jeżeli korzystasz z systemu **OS X**, nie musisz się martwić - Ruby jest zainstalowany domyślnie.

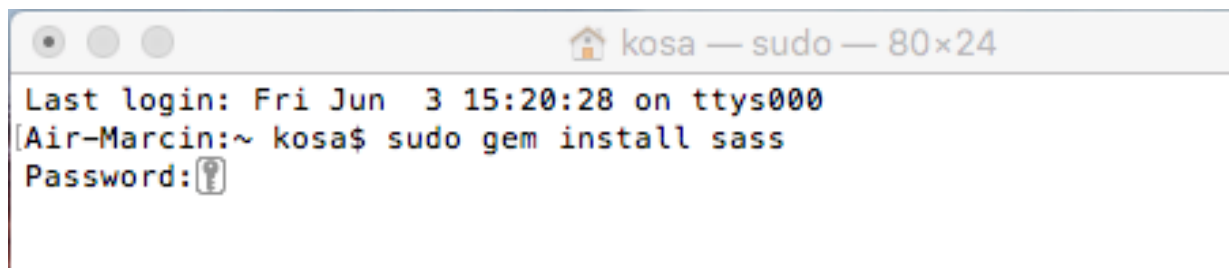
### Instalacja Sassa - linia komend

Pozostaje nam jedynie zainstalować Sass. W tym celu otwieramy Terminal. Jest to program *Terminal.app*, znajdziesz go w folderze *Programy/Narzędzia* lub za pomocą wyszukiwarki:

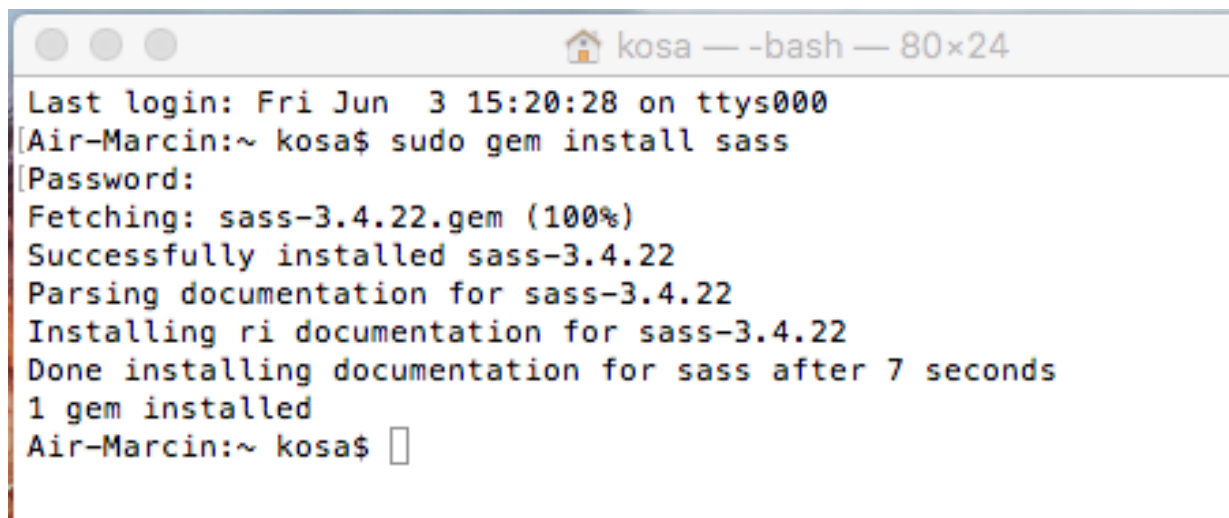


Następnie wpisujemy tam polecenie:

```
sudo gem install sass
```



Wymagane będzie podanie naszego hasła. Po jego podaniu instalacja odbędzie się automatycznie.



Możemy sprawdzić, czy instalacja się powiodła przez sprawdzenie wersji za pomocą polecenie **sass -v**:

```
Air-Marcin:~ kosa$ sass -v
Sass 3.4.22 (Selective Steve)
Air-Marcin:~ kosa$
```

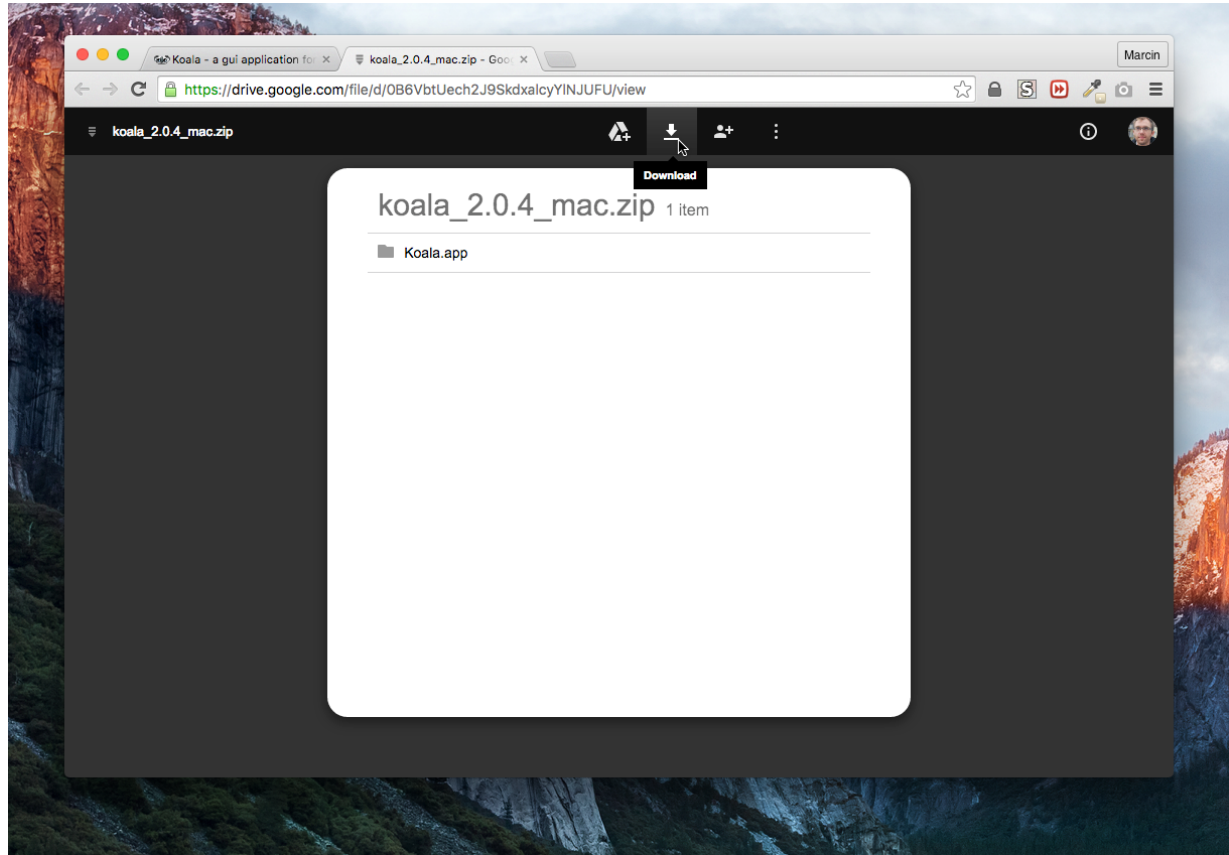




# Instalacja Koala App

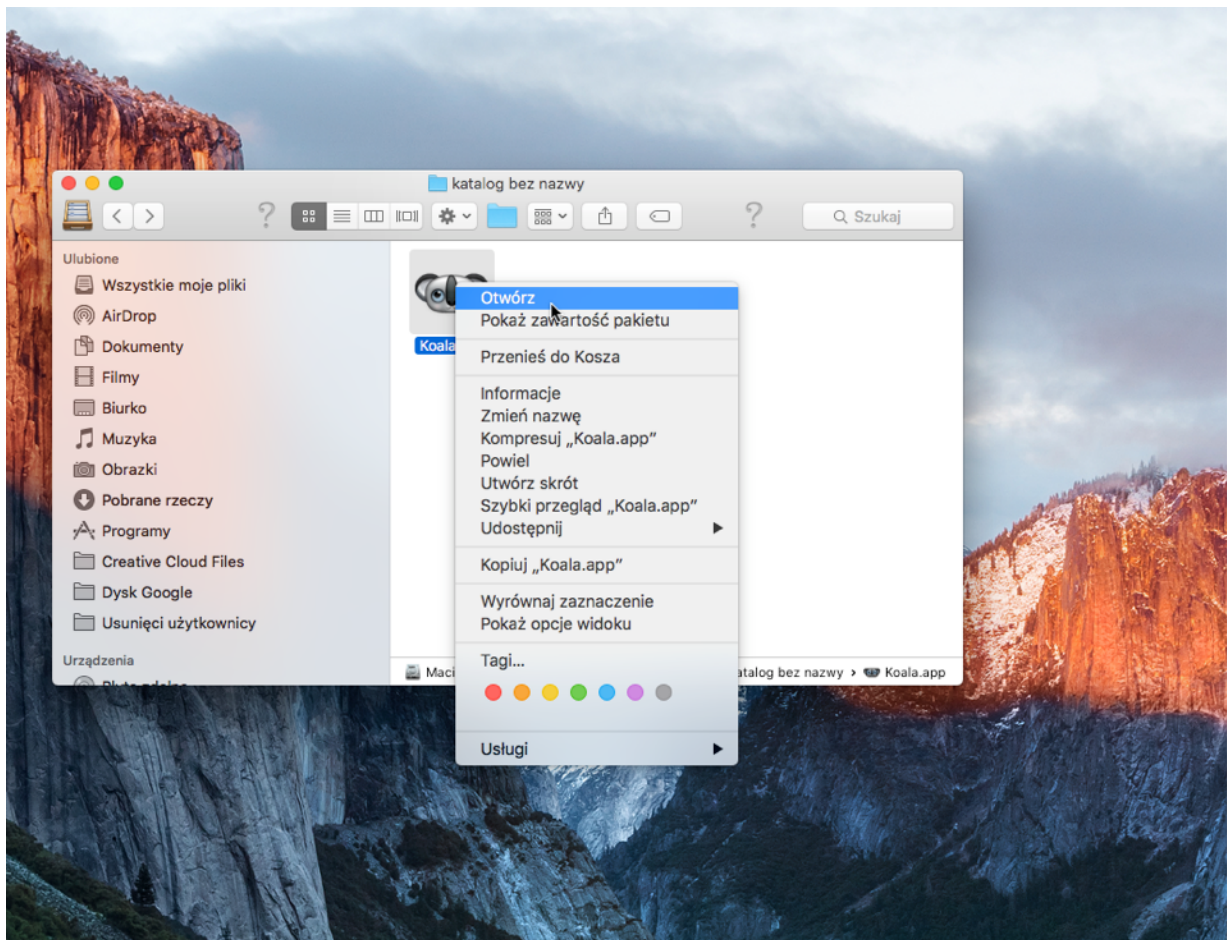
Pobieramy program Koala ze strony <http://koala-app.com/>

Po kliknięciu download zostaniemy przekierowani na stronę Google drive, gdzie jest do pobrania program Koala.app. Klikamy przycisk download na górze.



Następnie otwieramy katalog z pobranym programem.

Aby uruchomić aplikację, należy kliknąć prawym przyciskiem myszy (lub kliknąć przy wciśniętym klawiszu CMD) na ikonie aplikacji, wybrać Otwórz, a następnie potwierdzić chęć otwarcia aplikacji. Przy standardowym, dwukrotnym kliknięciu na ikonie aplikacji, system nie pozwoli na uruchomienie aplikacji, gdyż pochodzi ona z nieznanego źródła.



---

## Linux

---

### Instalacja Ruby

W przypadku systemów z rodziny **Linux** sprawa jest nieco skomplikowana, gdyż nie wszystkie dystrybucje mają domyślnie zainstalowany język Ruby.

Aby sprawdzić, czy język jest zainstalowany, otwórz terminal (np. za pomocą skrótu `ctrl+alt+t` lub znajdź go za pomocą wyszukiwarki wpisując "terminal") i wpisz:

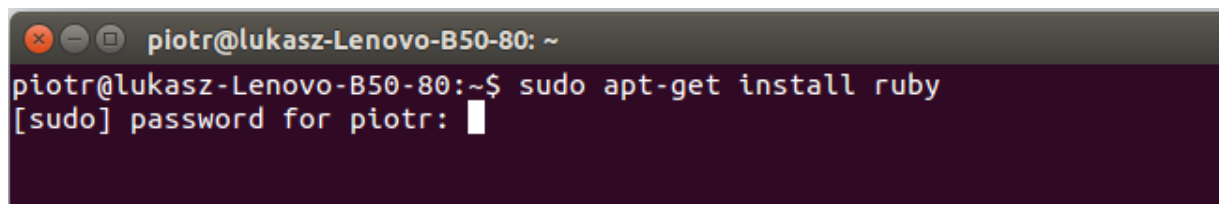
```
ruby -v
```

Naciśnij enter. Jeśli Ruby jest zainstalowany, to powinna zostać wyświetlona jego wersja:

```
piotr@lukasz-Lenovo-B50-80: ~  
piotr@lukasz-Lenovo-B50-80:~$ ruby -v  
ruby 1.9.3p484 (2013-11-22 revision 43786) [x86_64-linux]  
piotr@lukasz-Lenovo-B50-80:~$
```

Jeżeli nie zostanie wyświetlona wersja, możesz zainstalować Ruby za pomocą swojego systemu zarządzania pakietami, np. używając tego polecenia:

```
sudo apt-get install ruby
```



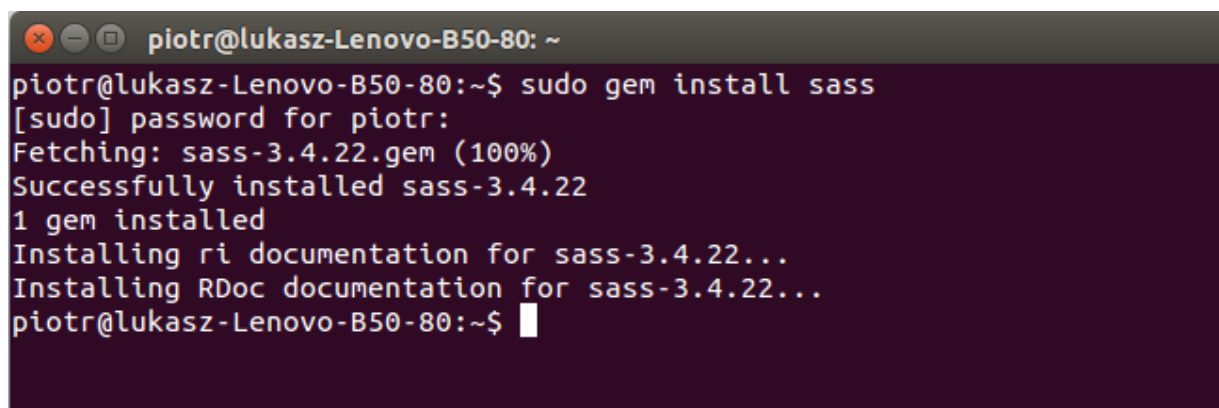
```
piotr@lukasz-Lenovo-B50-80: ~  
piotr@lukasz-Lenovo-B50-80:~$ sudo apt-get install ruby  
[sudo] password for piotr: █
```

Zostaniemy poproszeni o podanie hasła. Po jego wpisaniu instalacja przebiega samoczynnie.

## Instalacja Sassa - linia komend

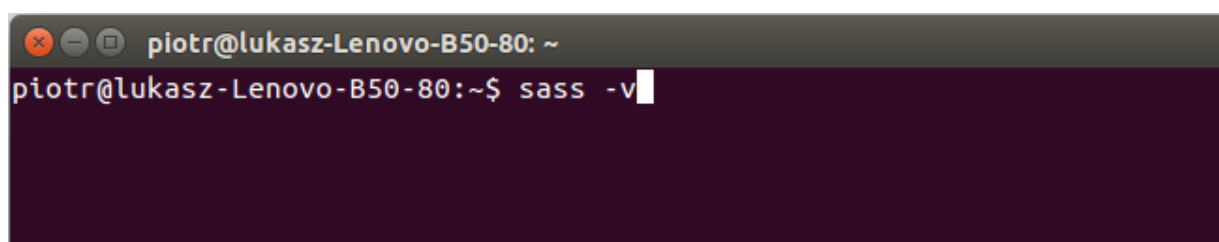
Kiedy mamy działające środowisko Ruby na naszym komputerze, możemy zainstalować Sass. Dokonujemy tego za pomocą wewnętrznego narzędzia do instalacji pakietów Ruby, czyli RubyGems. Aby to zrobić, wystarczy wpisać w terminalu polecenie:

```
sudo gem install sass
```



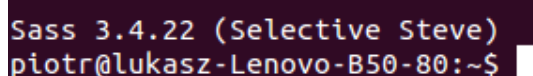
```
piotr@lukasz-Lenovo-B50-80: ~  
piotr@lukasz-Lenovo-B50-80:~$ sudo gem install sass  
[sudo] password for piotr:  
Fetching: sass-3.4.22.gem (100%)  
Successfully installed sass-3.4.22  
1 gem installed  
Installing ri documentation for sass-3.4.22...  
Installing RDoc documentation for sass-3.4.22...  
piotr@lukasz-Lenovo-B50-80:~$ █
```

Po zakończeniu procesu możemy sprawdzić, czy instalacja się powiodła przez wpisanie polecenia wyświetlającego wersję: **sass -v**



```
piotr@lukasz-Lenovo-B50-80: ~  
piotr@lukasz-Lenovo-B50-80:~$ sass -v █
```

Powinniśmy otrzymać komunikat z wersją: **Sass 3.4.22 (Selective Steve)**



```
Sass 3.4.22 (Selective Steve)  
piotr@lukasz-Lenovo-B50-80:~$ █
```

Jeżeli z jakiegoś powodu instalacja się nie powiedzie, zostaniemy o tym poinformowani:

```
bash: /usr/local/bin/sass: No such file or directory
```

```
piotr@lukasz-Lenovo-B50-80: ~  
piotr@lukasz-Lenovo-B50-80:~$ sass -v  
bash: /usr/local/bin/sass: No such file or directory
```

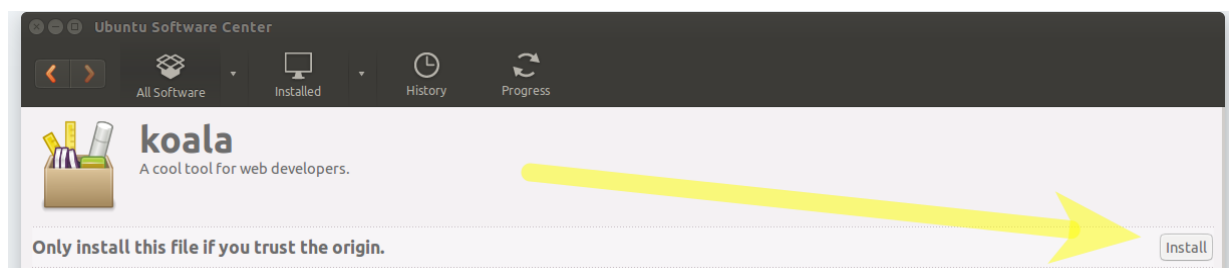
## Instalacja Koala App

Instalacja Koala App na systemach Linux jest prosta. Zademonstrujemy ją na systemie Ubuntu.

Pobieramy plik ze strony <http://koala-app.com>. Będzie to plik z rozszerzeniem .deb.

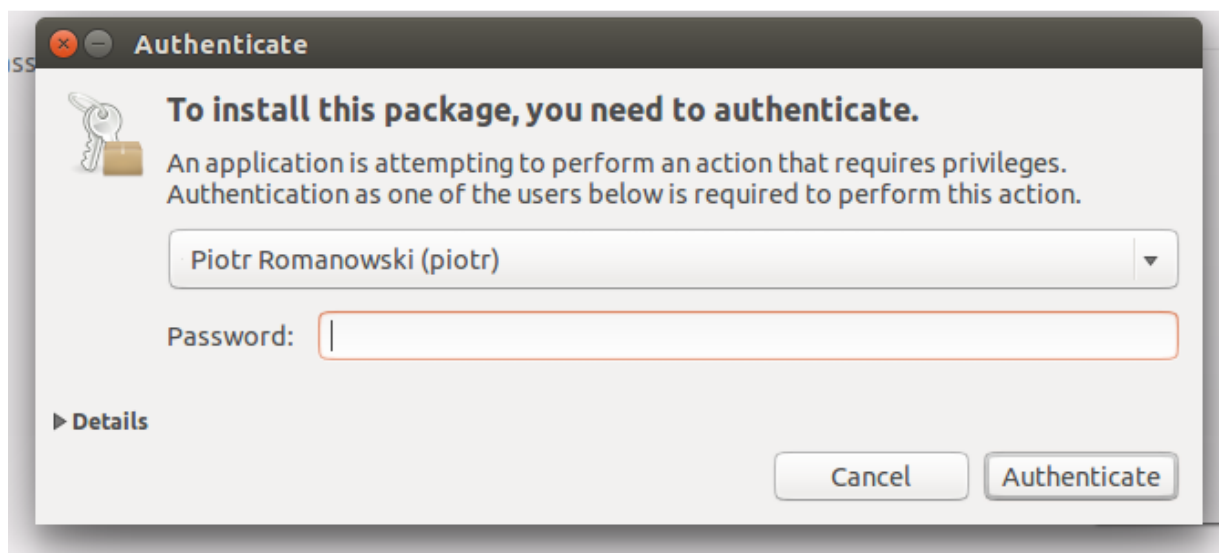


Klikamy go prawym przyciskiem myszy i wybieramy opcję "Otwórz w Centrum Oprogramowania Ubuntu". Następnie klikamy przycisk instaluj.



Zostaniemy poproszeni o podanie hasła.





Po jego podaniu program zostanie zainstalowany. Następnie możemy uruchomić program przez otwarcie terminala i wpisanie tam polecenia **koala**

Jeżeli otrzymamy komunikat błędu dotyczący braku biblioteki:

```
/usr/share/koala/koala: error while loading shared libraries:  
libudev.so.0: cannot open shared object file: No such file or  
directory,
```

wystarczy pobrać brakującą bibliotekę libudev.so.0 z poniższej lokalizacji i zainstalować ją w analogiczny sposób za pomocą Centrum Oprogramowania.

<http://packages.ubuntu.com/precise/i386/libudev0/download>

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

✓ Zapoznałem się!

## 6.4. Kompilacja



Proces kompilacji przedstawimy sobie na przykładzie Windowsa, ale jeśli korzystasz z innego systemu, to nie przejmuj się, kompilacja wygląda dokładnie tak samo.

---

### Linia komend



Najprostszym sposobem na kompilację pliku Sass lub SCSS do pliku CSS jest użycie polecenia **watch**. Aby to zrobić, otwieramy wiersz poleceń, a następnie przechodzimy do folderu, w którym znajduje się nasz projekt.


#### Wskazówka

Wiersz poleceń w Windows możemy też uruchomić już w konkretnym katalogu. Będąc w nim, wystarczy kliknąć prawym przyciskiem myszy w pustą przestrzeń trzymając jednocześnie Shift (Shift + PPM) i z menu kontekstowego wybrać opcję "Otwórz okno polecenia tutaj". Wtedy otworzy nam się terminal już w konkretnym katalogu.

W linii komend poruszamy się po katalogach za pomocą polecenia **cd**, niezależnie od systemu. Jeżeli chcemy przejść do katalogu, który znajduje się wewnątrz tego, w którym się znajdujemy, wpisujemy po prostu **cd nazwa\_katalogu**. By wyświetlić wszystkie pliki i katalogi, wpisujemy polecenie **ls** na systemach OS X i Linux oraz **dir** na systemie windows.

Aby przejść do katalogu wyżej, wpisujemy **cd ..**

W przykładzie poniżej wyświetliliśmy pliki, następnie przeszliśmy do folderu *sass* i wróciliśmy z powrotem do katalogu *documents*.

 C:\Windows\system32\cmd.exe

```
C:\Users\MB\Documents\Preprocesory>dir
Volume in drive C is Windows
Volume Serial Number is 1CAB-E7C8

Directory of C:\Users\MB\Documents\Preprocesory

06.06.2016  12:50    <DIR>          .
06.06.2016  12:50    <DIR>          ..
03.06.2016  16:45    <DIR>          admin
06.06.2016  12:50         1 492 output.css
06.06.2016  12:50    <DIR>          Sass
02.06.2016  16:20         142 style.css
02.06.2016  16:22         207 style.less
02.06.2016  16:20         188 style.sass
02.06.2016  16:20         223 style.scss
02.06.2016  16:20         174 style.styl
                6 File(s)                2 426 bytes
                4 Dir(s)  85 005 234 176 bytes free

C:\Users\MB\Documents\Preprocesory>cd sass
C:\Users\MB\Documents\Preprocesory\Sass>cd ..
C:\Users\MB\Documents\Preprocesory>_
```

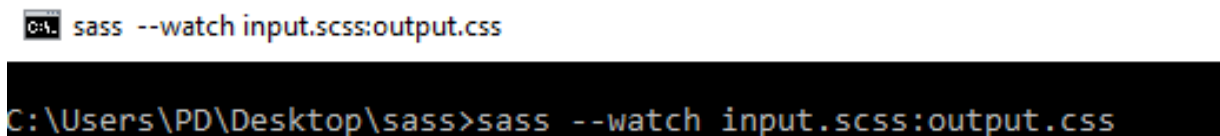




Kiedy znajdziemy się w folderze z naszym projektem, możemy uruchomić Sassa. Polecenie **watch** sprawia że Sass będzie "obserwować" plik `.sass` lub `.scss` i przy każdorazowym zapisie przekompiluje go do pliku `.css`. W poleceniu podajemy nazwę pliku, który Sass ma obserwować oraz nazwę pliku, do którego ma być on skompilowany.

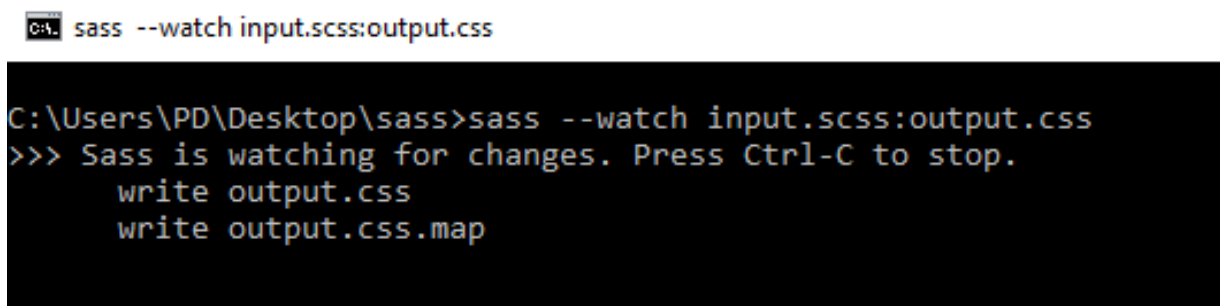
Po przejściu do folderu wydajemy następujące polecenie:

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



```
C:\Users\PD\Desktop\sass>sass --watch input.scss:output.css
```

Teraz każdorazowe zapisanie zmian pliku `.scss` spowoduje skompilowanie go do `.css`, który następnie możemy podpiąć do naszej strony internetowej.



```
C:\Users\PD\Desktop\sass>sass --watch input.scss:output.css
>>> Sass is watching for changes. Press Ctrl-C to stop.
    write output.css
    write output.css.map
```

Jeżeli chcemy zakończyć działanie programu, wystarczy skorzystać ze skrótu klawiszowego `ctrl+c`.

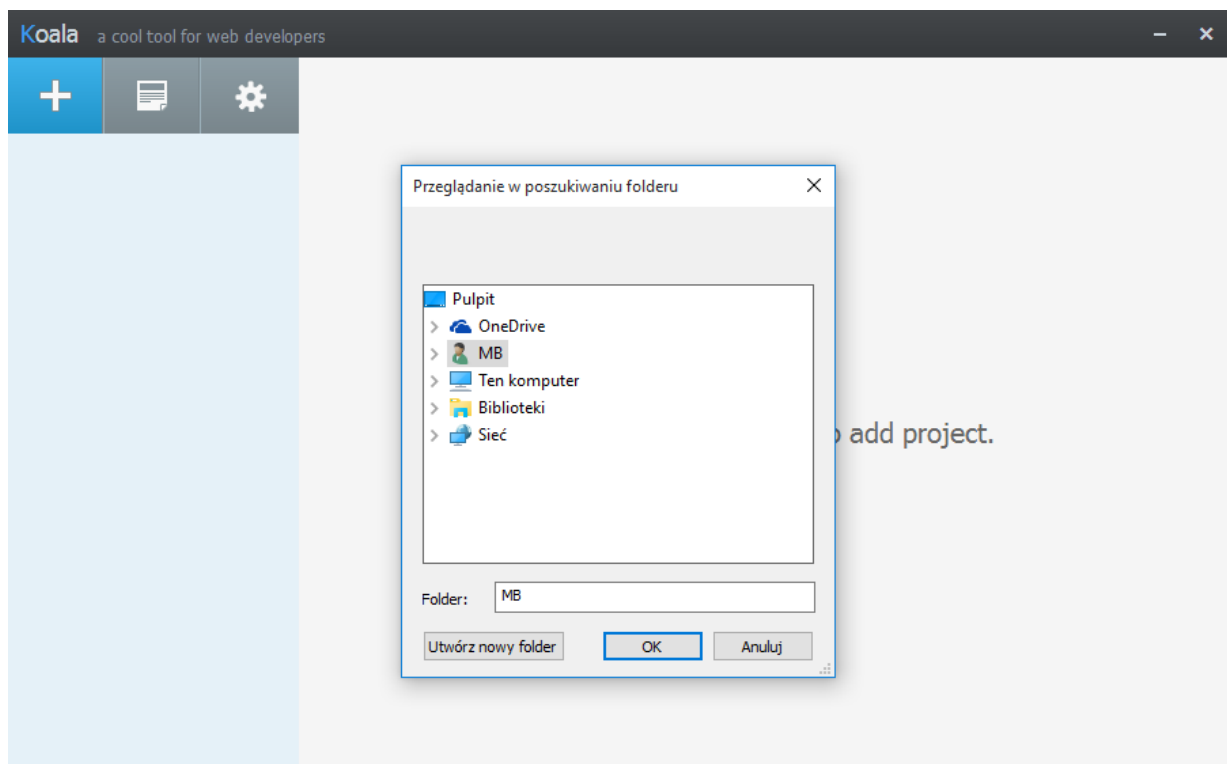
#### Wskazówka

[Kliknij tutaj by zobaczyć zbiór komend których można używać w terminalu Windowsa](#)

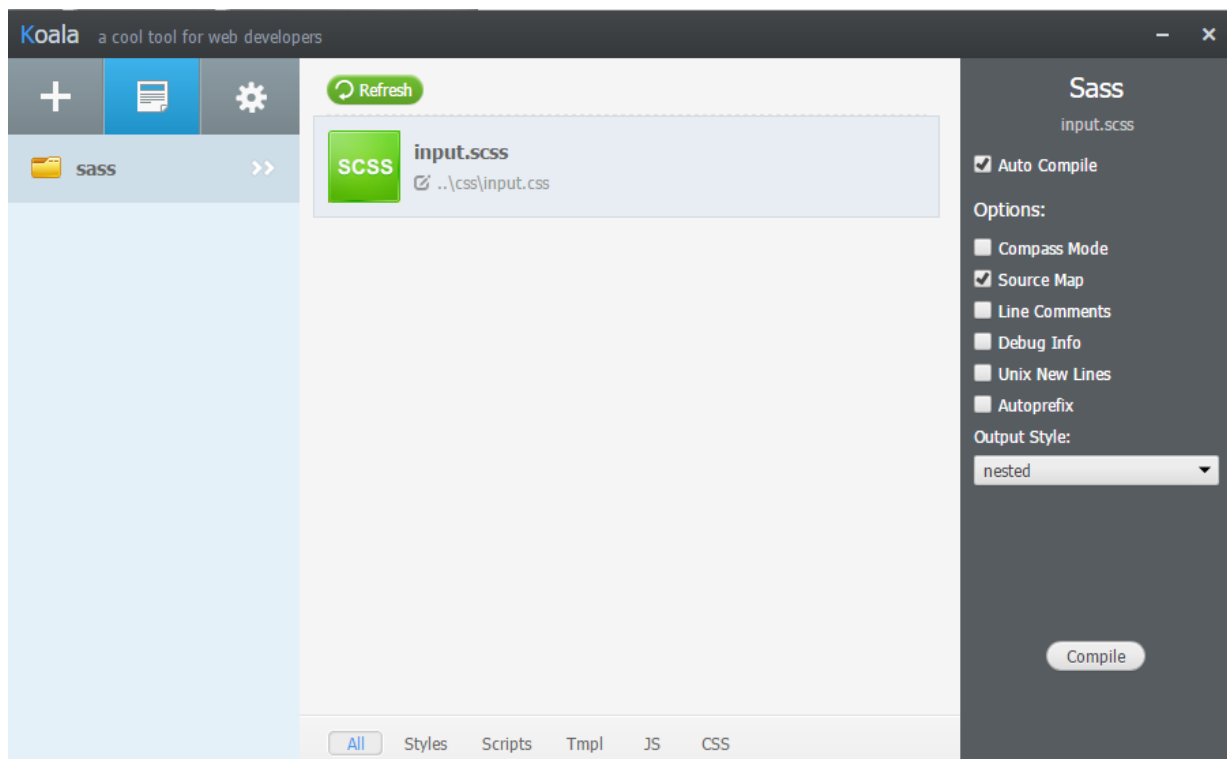
## Koala

Jeżeli zdecydowaliśmy się korzystać z aplikacji Koala, to kompilacja plików Sass będzie banalnie prosta. Dodajemy folder z projektem:





A następnie klikamy **compile**:



Gotowe! :)

Jedyną wadą tego podejścia jest to, że trzeba je każdorazowo ręcznie kompilować, natomiast przy użyciu linii komend kompilacja następuje automatycznie przy zapisie.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

✓ Zapoznałem(a)m się!



Regulamin

Polityka prywatności

© 2019 Kodilla

