

Bartosz Chucherko

Sass

Nowoczesne
arkusze stylów

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Małgorzata Kulik

Projekt okładki: Bartosz Chucherko

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/sasspp>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-2619-4

Copyright © Helion 2017

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Wstęp	9
Część I. Podstawy	13
Wprowadzenie	15
Czym jest Sass?	15
Dlaczego Sass?	15
Inne preprocesory	16
Sass i CSS	17
Sass i SCSS — różne formaty	17
Sass w 10 minut — przegląd możliwości	18
Zmienne	18
Zagnieżdżanie	19
Pliki cząstkowe i importowanie	20
Domieszki (Mixins)	20
Dziedziczenie (dyrektywa @extend)	21
Biblioteki i narzędzia rozszerzające Sass	24
Konwersja Sass do CSS — narzędzia	25
Instalacja Ruby Sass przy użyciu wiersza poleceń	27
Instalacja kompilatora w środowisku node	29
Edytory tekstu i Sass	31
Uwagi ogólne	32
Pierwszy kod	33
Zmienne	36
Nazwy zmiennych	38
Zasięg zmiennych	40
Określanie wartości domyślnej i zmiennej globalnej	42
Zagnieżdżanie	44
Selektor rodzica	48
Zagnieżdżanie właściwości CSS	51
Zagnieżdżanie media queries	52

Komentowanie kodu	54
Ustawianie formatu wyjściowego CSS	54
Rodzaje komentarzy	56
Garść praktycznych porad	57
Dyrektywa @import — importowanie plików	59
Importowanie plików cząstkowych	60
Importowanie zagnieżdżone	60
Podstawowa struktura projektu	61
Pliki cząstkowe, zmienne i flaga !default	63
Proste operacje arytmetyczne	65
Praca z kolorami	68
Modele HSL, RGB, RGBA	69
Funkcje działające w oparciu o HSL	70
Funkcje działające w oparciu o RGB(A)	72
Inne funkcje do pracy z kolorami	72
Użycie funkcji do pracy z kolorami w kodzie	74
Znajdowanie błędów w kodzie	75
Dyrektywa @mixin — domieszki	76
Przekazywanie argumentów do domieszki	81
Wartości domyślne	82
Argumenty nazwane	83
Nieokreślona liczba argumentów	83
Przekazywanie zawartości do domieszek (dyrektywa @content)	84
Przykłady użycia domieszek	86
Kiedy używać domieszek	88
Dyrektywa @extend — dziedziczenie i rozszerzenia	90
Kiedy używać dziedziczenia	100
Placeholders — selektory zastępcze	101
Domieszki, rozszerzenia, selektory zastępcze — kiedy stosować?	104
Podsumowanie — co powinienś wiedzieć	108

Część II. Techniki zaawansowane 111

Wprowadzenie	113
Logika programistyczna w Sass	114
Typy danych	115
Wartości numeryczne	115
Łańcuchy tekstowe	116
Kolory	117
Wartości logiczne (typ boolowski)	117

Wartość nieokreślona null	118
Listy	119
Mapy	120
Interpolacja	122
Powłoka interaktywna Sass	124
Instrukcja warunkowa if	126
Pętle	129
Pętla @for	129
Pętla @while	131
Pętla @each	132
Jak wybrać odpowiedni typ pętli?	135
Funkcje wbudowane Sass	136
Funkcje do przetwarzania łańcuchów tekstowych	137
Funkcje do pracy z liczbami	137
Funkcje do przetwarzania list	139
Funkcje do przetwarzania map	140
Funkcje introspekcyjne	145
Funkcje do operacji na selektorach	148
Funkcje niesklasyfikowane	148
Tworzenie własnych funkcji	150
Własna funkcja czy domieszka?	151
Podsumowanie — co powinieneś wiedzieć	152

Część III. Dobre praktyki i optymalizacja projektów Sass155

Wprowadzenie	157
Optymalizacja technik Sass	159
Stosowanie zmiennych	159
Zagnieżdżanie selektorów	161
Domieszki, rozszerzenia i selektory zastępcze	162
Funkcje i pętle	164
Importowanie	165
Instrukcja warunkowa	166
Debugowanie kodu przy użyciu map źródłowych (source maps)	166
Architektura projektów	169
Pliki cząstkowe i importowanie	170
Struktura i podział	170
Dzielenie interfejsu na komponenty	172
Przegląd architektur CSS i systemów organizacji kodu	173

Konwencje i styl kodu	193
Styl, składnia i formatowanie	193
Komentarze	197
Konwencje nazw elementów składni Sass	199
Przedrostki producentów	201
Tworzenie kodu produkcyjnego	202
Podsumowanie — co powinieneś wiedzieć	204
.....	204

Część IV. Sass w praktyce 207

Praktyczne przykłady zastosowania	209
Zerowanie stylów	209
Zapytania medialne i strony responsywne	209
Technika clearfix	211
Zamiana jednostek	212
Centrowanie elementów	213
Generowanie przycisków	213
Określanie wyglądu linków	215
Szybkie określanie pozycji elementu	215
Szybkie określanie rozmiaru elementu	216
Przycinanie zbyt długiego tekstu	216
Tworzenie układu strony i siatek	216
Typografia w Sass	218
Zastosowanie list i map	220
Tworzenie szablonów tematycznych (themes)	224
Pozostałe techniki	230
Sass i Bootstrap	231

Część V. Biblioteki i dodatki do Sass 243

Wprowadzenie	245
Omówienie wybranych narzędzi i bibliotek	246
Bourbon — przydatna kolekcja domieszek	249
Instalacja	250
Przegląd praktycznych dodatków biblioteki Bourbon	251
Przegląd innych domieszek biblioteki Bourbon	255
Przegląd funkcji biblioteki Bourbon	257
Neat	260
Instalacja	260
Konfiguracja siatki	261

Określanie rozpiętości elementu	263
Zagnieżdżanie siatki	264
Przesuwanie elementów na siatce	266
Dodawanie elementów responsywnych	266
Definiowanie kontekstów	268
Breakpoint	270
Tworzenie prostych zapytań medialnych (media queries)	270
Określanie docelowego typu mediów	272
Określanie właściwości	272
Określanie docelowej rozdzielczości	273
Sass-mq	275
Narzędzia Sass do typografii	277
Uwagi końcowe	281
DODATEK A. Ściąga	283
Zmienne	283
Zagnieżdżanie	284
Komentarze	285
Importowanie plików częściowych	285
Domieszki	285
Dyrektywa @extend	287
Pętle i instrukcje warunkowe	287
Typy danych	288
Funkcje do przetwarzania łańcuchów tekstowych	289
Funkcje do pracy z kolorami	290
Funkcje do pracy z liczbami	291
Funkcje do przetwarzania list	292
Funkcje do przetwarzania map	293
Funkcje introspekcyjne	294
DODATEK B. Przydatne materiały	297
Skorowidz	298

Optimalizacja technik Sass

Jeśli uważnie czytałeś poprzednie dwie części książki, zapewne zwróciłeś uwagę, że w wielu miejscach pojawiły się już liczne wskazówki dotyczące najlepszych praktyk. Tak było np. w przypadku zasad stosowania komentarzy, zmiennych, domieszek, rozszerzeń, importowania plików czy kwestii wyboru pętli. W tym rozdziale podsumujemy i uzupełnimy tę wiedzę.

Swego czasu w Internecie pojawiło się wiele dyskusji i krytyki na temat tego, że preprocesory takie jak Sass generują kod słabej jakości, gorszy od arkuszy CSS pisanych „ręcznie”. Wraz z upływem czasu, popularyzacją preprocesorów, a co za tym idzie wypracowaniem dobrych praktyk sprawa nieco się rozjaśniła. Niezmienna jednak pozostaje stara prawda — **słaby kod Sass wygeneruje słaby kod CSS**. Nadużywanie i nieumiejętne stosowanie którejkolwiek z omawianych dotąd technik, takiej jak na przykład rozszerzanie, może szybko doprowadzić do powstania rozдутego kodu CSS i problemów ze specyficznością selektorów.

Celem optymalizacji kodu Sass jest otrzymanie wydajnego, przejrzystego, łatwego w modyfikacji i utrzymaniu kodu, zgodnego z zasadą DRY („bez zbędnych powtórzeń”). Jeśli piszesz swój kod uważnie od samego początku, prawdopodobnie nie będziesz musiał wiele optymalizować. Często jednak kod pisze się szybko, zostawiając optymalizację na koniec, a w rezultacie nigdy nie ma ona miejsca. Warto więc pamiętać o dobrych praktykach i od początku tworzyć wydajny i schludny kod.

Prawdopodobnie najlepszą, ogólną radą, jaką mogę przekazać wszystkim osobom piszącym w Sass (a szczególnie tym początkującym), jest **zachowanie prostoty i umiaru**. Być może brzmi to banalnie, ale wielu deweloperów próbuje niekiedy „na siłę” stosować wymyślne i skomplikowane techniki kodowania w Sass tylko dlatego, że są one dostępne, tracąc przy tym czas i niepotrzebnie komplikując kod. Patrząc na kod, powinieneś od razy być w stanie zrozumieć jego działanie i przewidzieć, co zostanie wygenerowane do postaci CSS. **Pamiętaj, że celem Sass jako narzędzia jest umożliwienie twórcom arkuszy stylów pisania czystszej, bardziej czytelnej i łatwiejszej w utrzymaniu oraz modyfikacji kodu CSS.** Nieumiejętne i zbyt częste wykorzystywanie elementów składni Sass, takich jak domieszki, rozszerzenia czy zagnieżdżanie, może doprowadzić do zbędnego skomplikowania Twojego kodu. **Wszystkie elementy i techniki języka Sass, które dotąd poznałeś, powinny być używane zgodnie z przeznaczeniem, odpowiednio do sytuacji, a ich zastosowanie powinno mieć logiczne uzasadnienie i przynosić wymierną korzyść.** Pamiętaj również, że napisany przez Ciebie kod może kiedyś (mimo że teraz tego nie zakładasz) trafić w inne ręce, a im prostszy i bardziej czytelny kod, tym łatwiej będzie nowej osobie rozpocząć z nim pracę.

Przypomnijmy zatem dobre praktyki stosowania rozmaitych technik w Sass.

Stosowanie zmiennych

Zmienne z pewnością są jednym z największych dobrodziejstw Sass i elementem wykorzystywanym niemal w każdym projekcie opartym na Sass. **Użycie zmiennej ma sens wtedy, kiedy wiemy, że określona wartość ma istotne znaczenie, będzie wykorzystywana wielokrotnie**

lub jej wartość będzie często zmieniana. Przykładowo idealnymi kandydatami są wartości określające wielkości w projekcie (marginesy, rozmiary czcionek itp.), definiujące siatkę (ilość i szerokość kolumn, wielkość odstępów) bądź kolory tworzące schemat kolorystyczny.

Wszelkie miary warto określić w oparciu o jakąś regułę bądź schemat proporcji. Przykładowo możemy zdefiniować globalny margines, a wszystkie marginesy dla poszczególnych elementów definiować w jego oparciu. Podobnie można uczynić z rozmiarem tekstu na stronie bądź z kolorami — zdefiniować kolor podstawowy, a inne kolory w schemacie wygenerować za pomocą odpowiednich funkcji do pracy z kolorami. W ten sposób zmiana wariantu kolorystycznego strony będzie niezwykle szybka.

Pamiętaj również o **przyjęciu spójnej konwencji nazewnictwa zmiennych**, abyś nie musiał później zastanawiać się, jakie jest znaczenie konkretnej zmiennej. Zagadnienie to będzie omówione bardziej szczegółowo w dalszych rozdziałach.

Zmienne warto wydzielić do osobnego pliku konfiguracyjnego (np. `_variables.scss`, `_config.scss`). Na początku takiego pliku na ogół na samej górze definiujemy zmienne globalne, bazowe, będące podstawą naszego projektu. Mogą się tam znaleźć definicje podstawowych kolorów, miar, czcionek i elementów typografii, parametrów siatki czy wartości z-index. Tutaj warto również zadbać o porządek, grupując zmienne, czyli np. najpierw definicja siatki, później czcionek, kolorów itd.

Po zdefiniowaniu bazowych zmiennych możemy określić zmienne poszczególnych komponentów, np. nagłówka, artykułu, przycisków.

Przykładowo plik taki może wyglądać następująco:

```
// =====
// GLOBAL VARIABLES
// =====

// Base
$base-margin: 0.625em;
$base-padding: 0.625em;
$base-radius: 0.25em;

// Grid
$grid-column: 4.2358em;
$grid-gutter: 1.618em;
$grid-columns: 12;
$grid-max-width: 1280px;

// Colors
$color-brand: #cc6699;
$color-primary: #333333;
$color-secondary: lighten($color-brand, 20%);
$color-link: blue;

$body-bg: #ffffff;
$text-color: #111111;
```

```
// Fonts
$font-family-sans-serif: "Helvetica Neue", Helvetica, Arial, sans-serif;
$font-family-serif:      Georgia, "Times New Roman", Times, serif;

$text-size: 1rem;

// Global z-index
$zindex-topbar:      10000;
$zindex-tooltip:     10100;
$zindex-modal:       10200;

// =====
// COMPONENT VARIABLES
// =====

// Header
$header-bg: #333333;
$header-height: 56px;

// Buttons
$button-radius: $base-radius;
$button-padding: $base-padding $base-padding*2;
```

Należy przypomnieć, że istnieją inne typy danych, takie jak na przykład mapy, które idealnie nadają się do grupowania wielu zmiennych. W powyższym przykładzie moglibyśmy z powodzeniem zamienić listę zmiennych definiujących właściwość `z-index` w następujący sposób:

```
$zindex: (
  'topbar': 10000,
  'tooltip': 10100,
  'modal': 10200,
);
```

Warto również pamiętać o istnieniu flagi `!default`, która pozwala określać domyślną wartość zmiennej w przypadku, gdy przewidujemy możliwość jej nadpisania.

Dobrym pomysłem jest również podpatrzenie, jak z definiowaniem ustawień i zmiennych radzą sobie uznane, popularne frameworki, takie jak Bootstrap (https://github.com/twbs/bootstrap-sass/blob/master/assets/stylesheets/bootstrap/_variables.scss) czy Foundation (https://github.com/zurb/foundation-sites/blob/develop/scss/settings/_settings.scss). Pozwoli Ci to poznać dobre praktyki innych deweloperów.

Zagnieżdżanie selektorów

Możliwość zagnieżdżania jest bardzo przydatną cechą języka Sass — pozwala na grupowanie kodu i czyni go łatwiejszym w zarządzaniu, jednak zbyt głębokie zagnieżdżanie może być przyczyną problemów. Technika ta przyczyniła się do złej sławy preprocesorów na samym początku ich istnienia — krytykowano je dlatego, że generowały rozległe, nader specyficzne selektory i nieelegancki, rozdęty kod CSS. Jednak jak to często bywa, przyczyną było nieprawidłowe używanie zagnieżdżania i nieprzestrzeganie kilku prostych zasad.

Po pierwsze: **stosując zagnieżdżenia, staraj się unikać odwzorowywania struktury kodu HTML**. Odzworowanie struktury HTML w selektorach skutkuje sztywnym kodem, który ciężko jest przenosić i stosować do innych elementów interfejsu. Ponadto może prowadzić do utworzenia bardzo długich i mocno specyficznych selektorów, co w rezultacie ma również negatywny wpływ na wydajność. Używaj więc klas i myśl w kategoriach mikrokomponentów, które można stosować w dowolnym miejscu strony, niezależnie od struktury dokumentu HTML. Unikaj więc zagnieżdżeń, które generują selektory typu:

```
body #page .main .box div.body > div a
```

Drugą zasadą powszechną wśród wielu deweloperów jest **stosowanie maksymalnie 3 – 4 poziomów zagnieżdżeń**, choć niektórzy idą dalej, rekomendując maksymalnie 2 poziomy. Choć nie zawsze jest to możliwe, zasada ta skutecznie zmusza nas do pisania kodu w oparciu o komponenty interfejsu użytkownika i minimalizuje ryzyko tworzenia zbyt specyficznych selektorów. Jeżeli w Twoim kodzie zagnieżdżenia schodzą na głębsze poziomy bądź jesteś zmuszony do używania deklaracji `!important`, powinieneś pomyśleć o zoptymalizowaniu i przepisaniu kodu.

Jeśli odpowiednio przemyślisz strukturę elementu interfejsu użytkownika i zastosujesz odpowiednie nazewnictwo klas (stosując takie konwencje jak choćby *OOCSS*, *BEM* czy *Atomic Design*), jest wysoce prawdopodobne, że w ogóle nie będziesz musiał stosować zagnieżdżenia lub je zminimalizujesz. Oczywiście nie chodzi o to, aby zagnieżdżenia unikać całkowicie. Należy **używać go tam, gdzie trzeba, a nie dlatego, że po prostu można**.

Zagnieżdżanie jest rekomendowane na przykład wtedy, gdy używamy pseudoklas i pseudo-elementów. Pozwala to utrzymać porządek i strukturę komponentu w jednym miejscu.

Pamiętaj również, że warto od czasu do czasu (szczególnie na początku pracy z Sass) **sprawdzać wygenerowany kod CSS**. Jeśli widzisz w nim zbyt długie bądź niepożądane selektory, warto wrócić do kodu Sass i nieco go zoptymalizować.

Domieszki, rozszerzenia i selektory zastępcze

Techniki te opisałem dość szczegółowo w części I książki i na tym etapie powinieneś wiedzieć, kiedy każda z tych technik ma zastosowanie.

Domieszki to bardzo przydatne narzędzie w Sass, które umożliwia pisanie kodu wielokrotnego użytku, bez niepotrzebnych powtórzeń. Tak jak w pozostałych przypadkach należy jednak pamiętać o umiarze i prostocie. Tak jak nie ma sensu przerabianie każdego niemal fragmentu kodu na domieszkę, tak również należy wystrzegać się tworzenia bardzo skomplikowanych i rozbudowanych domieszek, które mają „robić wszystko”. Jeśli napisana przez Ciebie domieszka osiąga zbyt duże rozmiary (przykładowo nie mieści się na jednym ekranie edytora), to być może należy pomyśleć o rozbiciu jej na mniejsze.

Często można jeszcze spotkać domieszki pisane tylko po to, aby generowały kod CSS obsługujący przedrostki producentów. Taka praktyka była kiedyś powszechna, jednak odkąd istnieją specjalne narzędzia zajmujące się problemem kompatybilności przeglądarek i obsługi prefiksów, takie rozwiązanie nie ma sensu. Szerzej opisuję ten problem w części IV — „Przegląd innych domieszek biblioteki Bourbon”.

Jeśli chodzi o stosowanie rozszerzeń, temat ten został obszernie omówiony w pierwszej części książki. Dla usystematyzowania wiedzy powtórzmy najistotniejsze wskazówki:

- W zdecydowanej większości przypadków użycie domieszek jest sensowniejsze. W przeciwieństwie do rozszerzeń — trudniej o nieprzewidziane rezultaty, a sam kod Sass jest również łatwiejszy do zrozumienia.
- Używaj domieszek, jeśli istnieje potrzeba przekazania argumentów, a gdy chcesz wykorzystać wielokrotnie powtarzające się fragmenty kodu, dobrym rozwiązaniem będą selektory zastępcze (ang. *placeholders*).
- Dyrektywy `@extend` używaj tylko wtedy, gdy istnieje silna relacja między elementami, a jeśli takiej relacji nie ma, używaj domieszek. Grupowanie selektorów, jako rezultat rozszerzania, nie może następować przypadkowo, powinny być one logicznie powiązane ze sobą.
- Nie wykorzystuj dyrektywy `@extend` do rozszerzania selektorów (klas) występujących w arkuszach więcej niż raz ani selektorów zagnieżdżonych — pozwoli to uniknąć nieprzewidzianych sytuacji. Jeszcze bezpieczniej będzie wykorzystać w takiej sytuacji selektory zastępcze, tworzone raz w całym arkuszu.
- Jeśli nie jesteś w stanie łatwo przewidzieć rezultatu działania dyrektywy `@extend` w danym miejscu bądź wynikowy kod CSS zawiera fragmenty, których się nie spodziewałeś, popraw swój kod, wykorzystując selektory zastępcze lub domieszki.
- Sprawdzaj podczas używania dyrektywy `@extend`, czy wynikowy kod CSS jest zgodny z Twoimi przewidywaniami.
- Selektory zastępcze możesz z powodzeniem wykorzystać w tandemie z domieszkami. Oto dobry przykład:

```
%btn {
  border: 0;
  cursor: pointer;
  display: block;
  font-size: 14px;
  margin: 10px;
  padding: 10px;
  text-align: center;
}

@mixin button($bg) {
  @extend %btn;
  background-color: $bg;
  &:hover {
    background-color: lighten($bg,10%);
  }
}

.btn-default {
  @extend %btn;
  border: 1px solid grey;
}
```

```
.btn-primary {  
  @include button(blue);  
}
```

Funkcje i pętle

Jak już wiesz, funkcja sama z siebie nie generuje żadnego kodu CSS, zwraca natomiast wartość jakiegoś działania, która może być użyta w kodzie. Jeśli wykonujesz wielokrotnie jakieś obliczenia, warto wykorzystać do tego celu gotowe funkcje lub stworzyć własną.

Nie wahaj się więc używać funkcji, szczególnie w przypadkach takich jak wyliczenia wielkości i miar (np. elementy układu strony), konwersje jednostek (np. z wartości px na em) czy operacje na kolorach (np. zmiana odcienia, rozjaśnienie koloru). Wykorzystajmy cechę Sass, która pozwala wykonywać obliczenia zamiast na sztywno wpisywać wynik działania w kodzie.

Zastosowanie funkcji będzie również konieczne, jeśli planujesz wykorzystać bardziej złożone typy danych, jak listy lub mapy.

Pętle to dyrektywy sterujące umożliwiające automatyzację generowania kodu. W praktyce najczęściej wykorzystuje się pętle `@for` oraz `@each`. Pamiętaj, że **zasadniczym celem stosowania pętli jest zautomatyzowanie pewnych zadań, redukcja powtórzeń kodu oraz uczynienie naszego kodu bardziej zwięzłym**. Jeśli masz do napisania reguły, w których zauważasz jakiś schemat (np. selektory bądź właściwości CSS różniące się o ustaloną wartość numeryczną), to warto zastosować pętle. Pamiętaj o technice interpolacji, która będzie w takim przypadku pomocna. Dobrym przykładem może być generowanie stylów do określania szerokości kolumn w siatce.

Bardziej zaawansowanym deweloperom pętle przydadzą się również do pracy z mapami, na przykład do generowania schematów kolorystycznych.

Przypomnijmy jeszcze ogólne wytyczne do stosowania poszczególnych rodzajów pętli:

- Jeśli znasz dokładną liczbę powtórzeń (iteracji) pętli lub potrzebujesz wygenerować określoną ilość reguł, powinieneś wykorzystać pętlę `@for`. Jest ona także wskazana tam, gdzie konieczne jest użycie zmiennej licznikowej, np. generując selektory typu `:nth-child()`. Zmienna licznikowa jest częścią pętli, jest więc zawsze dostępna.
- Jeśli pracujesz z listami lub mapami, wygodniejsze będzie użycie pętli `@each` z uwagi na jej specyficzny mechanizm działania.
- Pętla `@while` ma podobne zastosowanie jak `@for`, z tą różnicą, że daje nam większą kontrolę nad zmienną licznikową. W praktyce jest jednak najrzadziej wykorzystywanym typem pętli.

Importowanie

Importowanie plików cząstkowych przy użyciu dyrektywy `@import` jest konieczne niemal w każdym projekcie. Jak wiesz, umożliwia podział całości kodu na mniejsze, osobne pliki i ich późniejsze łączenie, co w przypadku dużych, rozbudowanych projektów znacznie ułatwia organizację arkuszy stylów.

W dalszym rozdziale dowiesz się, jakie są różne strategie podziału i organizacji kodu. Teraz natomiast warto wspomnieć o kilku podstawowych zasadach dotyczących samego importowania.

Przed wszystkim staraj się, aby **zawsze istniał w Twoim projekcie jeden główny plik Sass, który będzie służył wyłącznie do importowania innych plików** i „składania” poszczególnych modułów w całość. Plik taki najczęściej przyjmuje postać *main.scss*, *global.scss* czy też *<nazwa-aplikacji>.scss* i znajduje się w głównym folderze z arkuszami Sass. Warto, aby plik ten był również uporządkowany i przejrzysty. W zależności od struktury naszego projektu plik taki mógłby wyglądać następująco:

```
#vendor libraries
@import "vendor/normalize";
@import "vendor/mq";
@import "vendor/bourbon/bourbon";

#settings
@import "config/variables";

#helpers
@import "helpers/mixins";
@import "helpers/helpers";
@import "helpers/animations";

#base styles
@import "base/base";
@import "base/typography"

#components
@import "components/header";
@import "components/menu";
@import "components/search";
```

Pamiętaj, że kolejność importowania jest istotna, dlatego pliki zawierające na przykład definicje zmiennych bądź elementy wykorzystywane w obrębie całego projektu (domieszki, biblioteki zewnętrzne) muszą być dołączane jako pierwsze.

Gdy z jakiegoś powodu musisz ograniczyć działanie selektora bądź uczynić go bardziej specyficznym, możesz zastosować **importowanie zagnieżdżone**:

```
#main-page {
  @import "base/base";
  @import "base/typography"
}
```

Przy okazji importowania warto jeszcze wspomnieć o jednej rzeczy, mianowicie o „niechcianym kodzie”, inaczej określanym jako „plik wstydu”. To proste rozwiązanie zaproponowali panowie Chris Coyier, Harry Roberts i Dave Rupert (<http://csswizardry.com/2013/04/shame-css/>). O co chodzi?

Mianowicie istnieją sytuacje, gdy niestety jesteśmy zmuszeni do umieszczenia kodu, z którego nie jesteśmy specjalnie dumni, jest on konieczny do obsługi jakiejś starszej wersji strony, nadpisania reguł, zawiera tymczasowe poprawki i łaty czy też jest to zapomniany, ale niezbędny kod CSS, którego nikt nie ma czasu porządnie przepisać. Cały ten niechciany kod warto wtedy umieścić w jednym pliku o nazwie `_shame.scss` (bądź `_shame.css`). Plik ten importujemy na samym końcu pliku głównego (`_global.scss`). Warto też dodać komentarze wyjaśniające powód umieszczenia tych stylów.

Instrukcja warunkowa

Jak wiesz, Sass umożliwia wykonanie instrukcji warunkowej przy użyciu dyrektyw `@if` i `@else`. Najczęściej stosuje się je w przypadku tworzenia domieszek, funkcji, własnych bibliotek ogólnego przeznaczenia czy po prostu komponentów wielokrotnego użytku, kiedy w jednym fragmencie kodu potrzebujemy uzyskać inny rezultat w zależności od określonego warunku.

Spójrzmy na poniższy przykład:

```
@function bg-text-color($bg-color) {
  @if lightness($bg-color) < 50% {
    @return white;
  } @else {
    @return black;
  }
}

.btn-dark {
  background-color: #333333;
  color: bg-text-color(#333333);
}

.box-light {
  background-color: #eeeeee;
  color: bg-text-color(#eeeeee);
}
```

Tworzymy ogólną funkcję, która ma za zadanie zwrócić prawidłowy kolor tekstu w zależności od koloru tła (prawidłowy, czyli widoczny niezależnie od koloru tła). Dzięki temu prostemu rozwiązaniu wystarczy użycie jednej funkcji i zawsze zostanie automatycznie wygenerowany prawidłowy kolor.

Debugowanie kodu przy użyciu map źródłowych (source maps)

Source maps to dodatkowe pliki generowane przez kompilator, które mogą ułatwić nam znajdowanie błędów w kodzie. Jaki to ma związek z optymalizacją i wydajnym kodem? Otóż może być to przydatne szczególnie w kwestii analizy generowanych selektorów. Jeśli zdarzy nam się, że

w wynikowym pliku CSS otrzymamy zbyt specyficzny, długi, rozbudowany bądź nieoczekiwany selektor, narzędzia deweloperskie w przeglądarce wraz z mapami źródłowymi umożliwią nam szybkie znalezienie przyczyny.

Mapy źródłowe pozwalają nam zobaczyć, która linijka kodu Sass odpowiada za wygenerowanie konkretnej reguły w CSS. Dzięki temu bez konieczności przeszukiwania kodu źródłowego panel deweloperski w przeglądarce od razu wskaże nam właściwe miejsce.

Aby można było włączyć generowanie *source maps*, konieczne jest ustawienie odpowiedniego trybu kompilacji. Gdy w wierszu poleceń uruchamiamy standardową komendę do „obserwowania” i kompilacji plików Sass, dodajemy dodatkowo parametr `--sourcemap`:

```
sass --watch sass_project/scss:sass_project/css --sourcemap
```

Jeśli teraz zajrzysz do wygenerowanego pliku CSS, zobaczysz, że na końcu znajduje się dodatkowa linijka komentarza o postaci:

```
/*# sourceMappingURL=styles.css.map */
```

Ten fragment kodu to nic innego jak wskazanie na plik mapy źródłowej, który to z kolei ma za zadanie mapować skompilowany kod CSS z deklaracji w pliku Sass.

Drugą czynnością, którą musimy wykonać, aby skorzystać z tego narzędzia, jest sprawdzenie obsługi map źródłowych w przeglądarce — obsługuje je większość nowoczesnych przeglądarek takich jak Chrome, Firefox czy Safari.

Chrome

W przeglądarce Chrome od wersji 39 obsługa *source maps* jest włączona domyślnie, więc nie musisz niczego konfigurować, aby z nich skorzystać. Jeśli z jakiegoś powodu ich obsługa jest wyłączona, możesz to zmienić, wchodząc w *Narzędzia dla programistów*, a następnie wybierając ustawienia i odnajdując opcję *Włącz obsługę map źródłowych CSS* (ang. *Enable CSS source maps*).

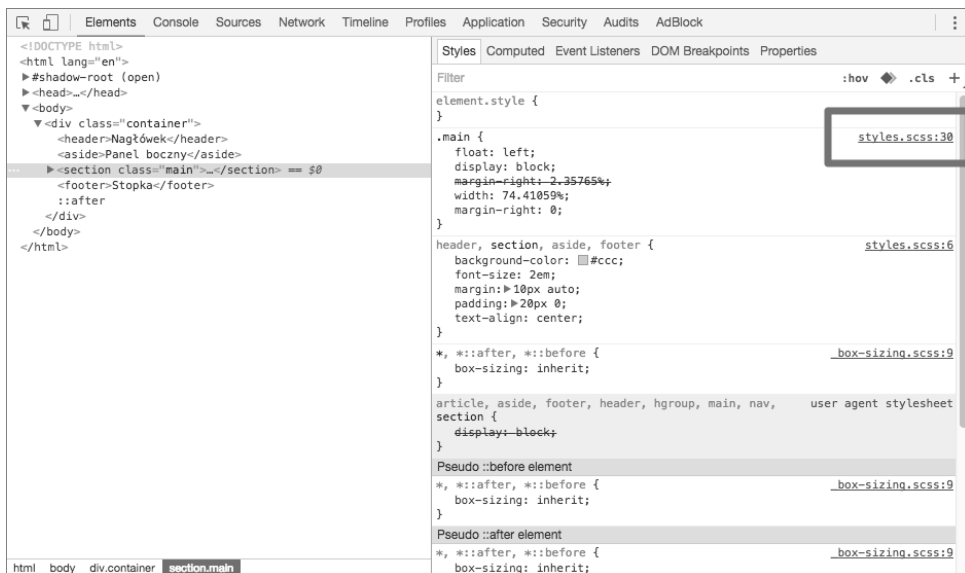
Firefox

W przeglądarce Firefox od wersji 29 obsługa map źródłowych CSS jest również włączona. Możesz to sprawdzić, włączając *Inspektora*, a następnie wejść w *Ustawienia narzędzi* i odszukać opcję *Oryginalne źródła* (ang. *Show original source maps*).

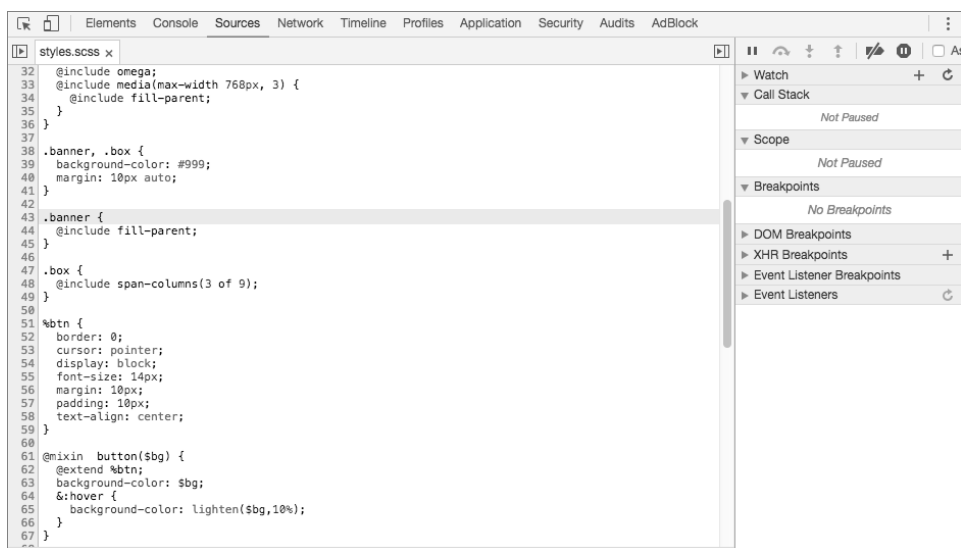
Safari

W Safari również obsługa map jest włączona domyślnie. Żadna dodatkowa konfiguracja nie jest potrzebna.

Kiedy obsługa map jest włączona, posługiwanie się nimi jest nadzwyczaj proste. Wystarczy w *Inspektorze WWW* otworzyć panel stylów, zbadać odpowiedni element w dokumencie HTML bądź znaleźć interesującą nas regułę CSS. Po jej prawej stronie zobaczymy wskazanie na linijkę kodu Sass, która jest odpowiedzialna za wygenerowanie tej reguły CSS. Poniższy rysunek ukazuje to na przykładzie przeglądarki Chrome:



Kiedy klikniemy z kolei link przy danej regule CSS, otworzy się panel z plikiem źródłowym Sass, wskazując konkretną linijkę kodu.



W powyższym przykładzie znalezienie odpowiedniej liniжки w kodzie Sass bez użycia map źródłowych oczywiście byłoby bardzo łatwe, jednak w przypadku mocno rozbudowanego projektu i podziału kodu na wiele plików, stosowania zagnieżdżeń czy rozszerzeń ich pomoc może okazać się niezastąpiona.

Skorowidz

A

algebra Boole'a, 117
animacja, 24
Animate.sass, 24
Animate.scss, 248
architektura
 projektu, 61, 157, 169, 170, 205
 7-1, 185
 Atomic Design, 187
 BEM, 176
 ECSS, 178
 InuitITCSS, 190
 ITCSS, 189
 komponentowa, 172, 173, 176, 178, 179,
 180, 205
 OOCSS, 174
 SMACSS, 181
 tworzenie, 170, 172, 173, 187, 189, 193, 194
arkusz stylów, *Patrz też:* CSS
 analiza, 203
 tworzenie, 15, 32, 145, 170
Atom, 31
Autoprefixer, 201, 202, 256

B

BEM, 174, 176, 178
biblioteka, 171
 Bootstrap, *Patrz:* Bootstrap
 Bourbon, *Patrz:* Bourbon
 Breakpoint, 211, *Patrz:* Breakpoint
 CSSComb, 196
 Libsass, 25, 29
 Neat, *Patrz:* Neat
 Sass-mq, *Patrz:* Sass-mq
 tworzenie, 166

Bitters, 247
błąd, 75, 146, 166
Bootstrap, 16, 161
Bourbon, 24, 150, 201, 246, 249
 dodatki, 251
 funkcja, 257
 instalowanie, 250
Bourbon Neat, 24
Brackets, 31
Breakpoint, 247, 270
Breakpoints-Sass, 24
Buttons, 248

C

Catlin Hampton, 16
ciąg znaków, 37, 65, 116, 117, 137
 długość, 137
 losowy, 148
 przycinanie, 216
Coda, 31
Codekit, 25
Codepen, 27
Color Me Sass, 248
Compass, 24, 150, 201, 246
Compass.app, 25
Coyier Chris, 166
cross-browser compatibility, *Patrz:*
 kompatybilność między przeglądarkami
CSS, 15, *Patrz też:* arkusz stylów
 rozszerzenie, 15
 szybkie resetowanie stylów, 24, 209
 zorientowane obiektowo, *Patrz:* OOCSS
CSS Deck, 27
CSS reset, *Patrz:* CSS szybkie resetowanie stylów

CSS3

- prefiks, 24
- właściwość transition, 255

CSS-Crush, 16

czcionka, 37, 219, 222, 259, 278, 279

D

dane

- pobieranie, 141, 142, 143
- typ, 37, 115, 146, 152
 - boolowski, *Patrz*: wartość logiczna
 - klucz:wartość, 37, 115, 120, 141, 143, 145
 - kolor, *Patrz*: kolor
 - lista, *Patrz*: lista
 - mapa, *Patrz*: mapa
 - numeryczny, *Patrz*: liczba, wartość
 - numeryczna
 - tekstowy, *Patrz*: ciąg znaków
 - wartość nieokreślona, *Patrz*: null, wartość
 - nieokreślona
- domieszka, 20, 21, 24, 32, 76, 78, 86, 88, 92, 100, 104, 105, 106, 109, 110, 127, 136, 144, 146, 151, 162, 163, 197, 204, 210, 211
 - argument, 81, 89, 163
 - nazwany, 83
 - nieokreślona liczba, 83
 - wartość domyślna, 82
- breakpoint, 270
- clearfix, *Patrz*: technika clearfix
- ellipsis, 252
- fill-parent, 264
- media, 267
- mq, 275
- nazwa, 77, 89
- omega, 264
- outer-container, 263
- shift, 266
- span-columns, 263, 264
- triangle, 252
- tworzenie, 77, 166

dyrektywa

- @at-root, 230
- @content, 84, 85, 211, 228
- @else, 166
- @extend, 21, 90, 91, 92, 96, 97, 98, 101, 104, 105, 163, 196
- @function, 150

@if, 148, 166

@import, 20, 59, 60, 165, 169
implementacja, 59

@include, 21, 77, 78, 80, 82, 92, 197

@media, 52

@mixin, 21, 76

@return, 150

sterująca, 126

dziedziczenie, 21, 22, 90, 91, 97, 100, 109

E

Eclipse, 31

ECSS, 178, 180

edytor tekstu, 31, 194, 198

ekosystem PostCSS, 202

element

- interaktywny, 68
- plywający, 97, 211
- pozycja, 215, 253
- responsywny, 266
- rozmiar, 216, 253, 260
- szerokość, 32, 263
- wygląd, 90
- wyśrodkowanie, 213

Eppstein Chris, 16

Eyeglass, 246

F

flaga

- !default, 42, 63, 64, 161
- !global, 42, 43, 228
- !optional, 99
- style, 55

format

- SASS, 17
- SCSS, 17
- wyjściowy, 54, 57
 - kompaktowy, 54, 56
 - rozszerzony, 54, 55
 - skompresowany, 54, 56
 - zagnieżdżony, 54, 55

framework Compass, *Patrz*: Compass

Frost Brad, 187

funkcja, 136, 164, 204

abs, 138

adjust-color, 73

funkcja

- adjust-hue, 71
- ceil, 138
- comparable, 147
- complement, 71
- darken, 70
- desaturate, 71
- em, 258
- floor, 138
- grayscale, 71
- if, 148
- index, 139
- introspekcyjna, 145
- join, 139
- length, 139
- lighten, 70
- list-separator, 139
- łańcuch tekstowy, 137
- mapa, 144
- map-get, 141
- map-has-key, 143
- map-keys, 145
- map-merge, 144
- map-values, 145
- matematyczna, 137
- max, 138
- min, 138
- mix, 72
- modular-scale, 259
- new-breakpoint, 268
- nth, 139
- opacify, 73
- percentage, 138
- przetwarzanie
 - list, 139
 - map, 140
 - selektorów, 148
- quote, 137
- rem, 257
- rgb, 72
- rgba, 72
- round, 137
- saturate, 70
- selector-append, 148
- selector-parse, 148
- selector-unify, 148
- str-length, 137
- to-lower-case, 137

- to-upper-case, 137
- transparentize, 72
- tworzenie, 150, 151, 166
- type-of, 146
- unique-id, 148
- unit, 147
- unitless, 147
- unquote, 137
- variable-exists, 145
- wbudowana, 114, 136, 145
 - map-get, 135
- zagnieżdżanie, 74

G

- Geany, 31
- Giraudel Hugo, 185
- grid, *Patrz:* siatka

H

- HSL, *Patrz:* model przestrzeni barw HSL

I

- IDE, 31
- ikona, 222
- inline media query, *Patrz:* zapytanie medialne
śródliniowe
- instrukcja
 - @extend, 90
 - @if, 126
 - warunkowa, 126, 166, 205, 230
- interpolacja, 122, 123, 216, 222

J

- JSFiddle, 27

K

- klasa nazwa, 50
- Koala, 25, 26
- kolor, 18, 32, 36, 37, 68, 117
 - harmonia, 69
 - koło, 69
 - komplementarny, 71
 - mieszanie, 72

nasycenie, 70
 przestrzeń barw, *Patrz:* model przestrzeni barw
 przezroczystość, 69, 72, 73
 przyciemnianie, 68, 70
 rozjaśnianie, 70
 schemat
 analogowy, 69
 komplementarny, 69
 monochromatyczny, 69
 triada, 69
 zamiana w odcień skali szarości, 71
 koło kolorów, *Patrz:* kolor koło
 komentarz, 54, 57, 89, 109, 197
 blokowy, 56, 58, 198
 cichy, 54, 56, 57, 58, 198
 głośny, 54, 57
 jednowierszowy, 54, 56
 liniowy, 56
 wielowierszowy, 54, 56
 Komodo Edit, 31
 kompatybilność między przeglądarkami, 24, 162, 201, 256
 kompilacja, 25
 kompilator
 node-sass, 25
 Ruby Sass, 25, 27, 33
 SassC, 25
 komunikat
 generowanie, 146
 o błędzie, 75

L

Less, 16
 liczba, 65, 115
 negatywna, 65
 wartość bezwzględna, 138
 z jednostką, 65, 115, 147
 dzielenie, 66
 konwersja, 257, 258
 przeliczenie, 212
 zaokrąglenie, 137
 link, 215
 lista, 24, 119, 120, 132, 135, 139, 220
 długość, 139
 element, 139
 łączenie, 139

separator, 139
 wartości, 115
 zagnieżdżona, 120
 Live Reload, 25

Ł

łańcuch
 selektorów, 47
 tekstowy, *Patrz:* ciąg znaków

M

mapa, 37, 115, 120, 133, 134, 135, 140, 161, 164, 222
 klucz, 141, 142, 143, 145
 scalanie, 144
 wielopoziomowa, 223
 zagnieżdżona, 141
 źródłowa, 166, 167
 media query, *Patrz:* zapytanie medialne
 mixin, *Patrz:* domieszka
 model przestrzeni barw, 68
 HSL, 69, 70, 71
 RGB, 69, 72
 RGBA, 69, 72
 Modular Scale, 277
 motyw
 kolorystyczny, 64
 stylistyczny, *Patrz:* skórka tematyczna
 Myth, 16

N

namespacing, *Patrz:* przestrzeń nazw
 Neat, 247, 260
 NetBeans, 31
 Normalize-sass, 248
 notacja
 BEM, 50
 z myślnikiem, 199
 Notepad++, 31
 null, 37, 115, 118

O

Object Oriented CSS, *Patrz:* OOCSS
 OOCSS, 174

operacja arytmetyczna, 65, 66, 109, 114, 147
 dzielenie, 66
 na ciągach znaków, 117
 odejmowanie, 65

operator
 and, 118, 128
 arytmetyczny, 118
 matematyczny, 37
 not, 118, 128
 or, 118, 128
 porównania, 118
 potrójny, 230
 warunkowy, 230

P

pętla, 129, 135, 164, 216, 222
 @each, 132, 133, 135, 139, 143, 164, 204
 @for, 129, 135, 164, 204
 @while, 131, 135, 164
 nieskończona, 132

plik

 .css, 17
 .sass, 17
 .scss, 17
 _shame.scss, 166

 częstkowy, 20, 60, 62, 89, 109, 170

 importowanie, 60

 global.scss, 165

 importowanie, 20, 59, 165, 169, 170, 204

 kolejność, 60, 165

 zagnieżdżone, 60, 165

 kompresja, 202

 konfiguracyjny, 160

 main.scss, 165

 nazwa, 60, 62

 resetujący style, 209

 styles.css.map, 34

 styles.scss, 33

 wstydu, 166

 wynikowy, 54, 57

Plumber, 277, 278

PostCSS, 16

preprocesor, 16

Prepros, 25

przeglądarka

 kompatybilność, *Patrz:* kompatybilność

 między przeglądarkami

 obsługa map źródełowych, 167

przestrzeń

 barw, 68

 nazw, 200

przycisk, 24, 68, 213, 220

pseudoelement, 47, 48

pseudoklasa, 47, 48

R

Refills, 247

reguła, 44

 @import

 zagnieżdżanie, 61

 deklaracja, 44

 DRY, 16, 21, 76

 zagnieżdżanie, *Patrz:* zagnieżdżanie reguł

Rework, 16

RGB, *Patrz:* model przestrzeni barw RGB

RGBA, *Patrz:* model przestrzeni barw RGBA

Roberts Harry, 166, 189

rodzic, 48, 49, 230

rozszerzanie selektorów, *Patrz:* selektor

 rozszerzanie

Rupert Dave, 166

S

Saffron, 248

Sass, 15, 16

 architektura projektu, *Patrz:* architektura
 projektu

 kod, 17

 błąd, *Patrz:* błąd

 debugowanie, 166, 167

 komentowanie, *Patrz:* komentarz

 kompilacja, 17, 25, 27, 29, 62, 75, 167, 202

 optymalizacja, 159, 166

 produkcyjny, 202

 tworzenie, 31, 33

 znajdowanie błędów, 75, 166, 203

 konwencje nazw, 199

 powłoka interaktywna, 124

 rozpoczęcie pracy, 25, 157

Sassline, 248, 277

Sassmeister, 27

Sass-mq, 211, 248, 275

SassScript, 114

Sassy Inputs, 248

schemat kolorystyczny, 68
 scope, *Patrz*: zmienna zasięg
 Scout, 25, 26
 selektor
 atrybutu, 47
 dziecka, 47
 przetwarzanie, 148
 pseudoelementu, 47
 pseudoklasy, 47
 rodzica, 48, 49, 230
 rozszierzenie, 91, 92, 97, 99, 100, 109, 196
 zagnieżdżony, 22, 32, 44, 48, 197
 zastępczy, 101, 102, 103, 104, 106, 110, 163, 204
 złożony, 47, 92
 semantyka, 39
 siatka, 24, 246, 260
 konfiguracja, 261, 262
 tworzenie, 216
 zagnieżdżanie, 264
 sieć społecznościowa, 220
 Singularity.gs, 247, 269
 skalowanie, 16
 skórka tematyczna, 60
 słowo kluczowe
 \$red, 73
 \$saturation, 73
 @each, 132, 133, 134
 @for, 129
 @include, 77
 @mixin, 77
 in, 132, 134
 SMACSS, 181
 Snook Jonathan, 181
 source map, *Patrz*: mapa źródłowa
 sprites, 246
 string, *Patrz*: ciąg znaków
 strona, 218
 dostępność, 271
 układ, 260, 261, 263
 struktura, 115
 Stylelint, 203
 Sublime Text, 31
 Sullivan Nicole, 174
 Sunglass, 248
 Susy, 24, 247, 269
 szablon tematyczny, 64, 224, 225

T

tabelka, 24
 technika
 clearfix, 211, 251
 ellipsis, 216, 252
 responsywna, 260, 266, 270
 tekst, 218, 277
 czcionka, 259
 przycinanie, 216
 rozmiar, 219, 259
 zestaw czcionek, 219
 ternary operator, *Patrz*: operator warunkowy
 TextWrangler, 31
 theme, *Patrz*: skórka tematyczna
 Typesettings, 248, 277
 typografia, 218, 246, 277

V

vendor prefix, *Patrz*: CSS3:prefiks
 Vim, 31

W

warstwa, 221
 wartość
 liczbowa, 37, 147
 logiczna, 37, 115, 117
 nieokreślona, 115, 118
 null, *Patrz*: null
 numeryczna, 115
 procentowa, 138
 tekstowa, 37, 115, 116
 web accessibility, *Patrz*: strona dostępność
 Webstorm, 31
 Weizenbaum Natalie, 16
 właściwość, 21, 46, 90
 CSS, *Patrz*: właściwość
 CSS3 transition, 255
 deklarowanie, 194, 195
 zagnieżdżanie, *Patrz*: zagnieżdżanie
 właściwości
 z-index, 221
 wyrażenia negacja, 65

Z

- zagnieżdżanie, 19, 44, 45, 161, 162
 - funkcji, 74
 - reguł, 19, 48
 - reguła @import, 61
 - selektorów, 46, 230
 - siatki, *Patrz:* siatka zagnieżdżanie
 - właściwości, 51
 - zapytań medialnych, 52
- zapytanie medialne, 24, 84, 197, 260, 270
 - @media, 52, 210
 - rozdzielczość, 273
 - śródliniowe, 52, 209
 - tworzenie, 270, 275
 - typ mediów, 272
 - właściwość, 272
 - zagnieżdżanie, *Patrz:* zagnieżdżanie zapytań medialnych
- zasada DRY, 159
- zmienna, 18, 32, 36, 37, 65, 108, 114, 122, 145, 159, 204
 - definiowanie, 36, 42
 - funkcjonalna, 40
 - globalna, 40, 42, 85
 - grupowanie, 161
 - licznikowa, 129, 131, 135
 - lokalna, 41, 61, 196
 - nazwa, 36, 38, 39, 108, 160, 199, 200
 - typ, 146
 - wartość
 - domyślna, 42, 63
 - semantyczna, 39
 - wartość domyślna, 161
 - wartość semantyczna, 39
 - zasięg, 40, 42, 85
 - zawartość, 37
- znak
 - !, 38
 - #, 38, 122
 - \$, 36, 81, 117
 - %, 38, 101, 110
 - &, 48, 49, 50, 117
 - *, 38, 117
 - */, 54, 58
 - /, 38, 66
 - /*, 54, 58
 - //, 54, 56, 58
 - @, 38
 - +, 38, 65
 - alfanumeryczny, 116
 - ampersand, *Patrz:* znak &
 - apostrof, 116
 - cudzysłowu, 116, 137
 - krzyżyka, 38, 122
 - myślnika, 38, 40, 77, 199
 - podkreślenia, 38, 40, 60, 62, 77, 199
 - specjalny, 38, 116

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

Sass to metajęzyk, który umożliwia szybsze i wydajniejsze tworzenie bardziej zaawansowanych arkuszy stylów, niż jest to możliwe przy użyciu samego języka CSS. Co więcej, ten świetny metajęzyk zapewnia pełną zgodność ze standardowym CSS-em. Sass eliminuje wady i ograniczenia CSS, a ponadto oferuje możliwości znane z innych języków programowania. Twórcom stron WWW przydaje się to szczególnie w dużych i rozbudowanych projektach komercyjnych. A wszystko to można osiągnąć niewielkim kosztem — jeśli znasz CSS, opanowanie Sass nie wymaga zbyt dużo czasu ani wysiłku.

Jeśli tworzysz strony WWW i chcesz wzbogacić swój warsztat o znajomość jednego z ciekawszych oraz najdynamiczniej rozwijających się narzędzi designerskich i deweloperskich, sięgnij po książkę *Sass. Nowoczesne arkusze stylów*. Zrozumiesz dzięki niej zarówno podstawowe, jak i bardziej zaawansowane elementy składni języka, poznasz dobre praktyki tworzenia projektów Sass oraz nauczysz się je optymalizować. Zaznajomisz się również z praktycznymi przykładami zastosowania tego narzędzia oraz bibliotekami i dodatkami rozszerzającymi jego możliwości. Daj się wprowadzić w świat Sass!

- Podstawowe informacje na temat preprocesora Sass, jego działania i składni
- Zaawansowane elementy języka wraz z przykładami
- Liczne przykłady praktycznych technik przydatnych podczas tworzenia stron WWW
- Biblioteki i narzędzia ułatwiające pracę projektanta
- Techniki optymalizacji i zasady tworzenia prawidłowego kodu
- Metody pracy przy rozbudowanych projektach z wykorzystaniem wzorców i architektury
- Tworzenie skórek przy użyciu Sass i Bootstrap
- Instalacja i konfiguracja środowiska pracy i dodatków

Opanuj Sass i twórz przebojowe strony WWW w nowoczesny sposób!

Helion

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

Sprawdź najnowsze promocje:
➤ <http://helion.pl/promocje>
Książki najchętniej czytane:
➤ <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
➤ <http://helion.pl/novosci>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-2619-4



9 788328 326194

Informatyka w najlepszym wydaniu

cena: 59,00 zł