Technologia i rozwiązania

Less

Podstawy programowania

Poznaj język Less i zaawansowane możliwości CSS-a!

Bass Jobsen

Helion

[PACKT] open source* PUBLISHING community experience distilled

Tytuł oryginału: Less Web Development Essentials, Second Edition

Tłumaczenie: Piotr Rajca

ISBN: 978-83-283-1754-3

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/lesspp.zip

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

Printed in Poland.

Spis treści

ROZDZIAŁ 2

Stosowanie zmiennych i wstawek

Niniejszy rozdział jest poświęcony nieco dokładniejszej prezentacji Lessa, a konkretnie zagadnieniom stosowania zmiennych i wstawek. Zmienne w kodzie Lessa są przeważnie definiowane w jednym miejscu, choć można ich używać i zmieniać ich wartości. W Lessie zmienną można napisać przez umieszczenie za nią definicji. Zmienne są używane do definiowania często stosowanych wartości, dzięki czemu można je edytować tylko w jednym miejscu. Bazując na zasadzie *DRY* (nie powtarzaj się), używanie zmiennych do zapisywania często stosowanych wartości pomaga w tworzeniu witryn, których utrzymanie jest łatwiejsze. Z kolei wstawki służą do określania właściwości klas. Dzięki nim w jednym wierszu kodu można połączyć wiele deklaracji i używać ich w wielu miejscach kodu. W tym rozdziale przedstawię sposób tworzenia wstawek, korzystania z nich oraz ich wielokrotnego używania w projektach, jak również stosowania do tworzenia lepszych arkuszy stylów CSS bez powielania kodu.

W tym rozdziale przedstawię następujące zagadnienia:

- Stosowanie komentarzy w kodzie Lessa.
- Stosowanie zmiennych.
- Interpolacja zmiennych.
- Sposoby zapisywania wartości.
- Stosowanie wstawek.

Stosowanie komentarzy w kodzie Lessa

Komentarze sprawiają, że kod staje się bardziej przejrzysty i zrozumiały dla innych. To bardzo ważne, by móc precyzyjnie zrozumieć przeznaczenie i działanie analizowanego kodu. Właśnie z tego powodu na samym początku tego rozdziału przedstawię kilka informacji i przykładów komentarzy.

> **Wskazówka**
>
> Nie należy ograniczać liczby i wielkości komentarzy, mając na uwadze wielkość plików, czas ich pobierania oraz wydajność działania stron. W ramach procesu kompilacji i minimalizacji wynikowego kodu CSS wszystkie komentarze i inne fragmenty kodu służące do określania jego struktury praktycznie rzecz biorąc zostaną całkowicie usunięte. Zatem w celu poprawienia przejrzystości kodu i ułatwienia jego zrozumienia można dodawać komentarze bez żadnych ograniczeń.

Komentarze w kodzie Lessa można dodawać dokładnie tak samo, jak robi się to w kodzie CSS. Tekst komentarza jest zapisywany między sekwencjami znaków /* i */. Oprócz tego Less pozwala na tworzenie komentarzy, które rozpoczynają się od sekwencji znaków // i obejmują całą dalszą zawartość wiersza.

Jedynie prawdziwe komentarze CSS (/* */) będą kopiowane do wygenerowanego arkusza stylów CSS. Nawet jednak one zostaną usunięte z kodu CSS w wyniku jego minimalizacji. Poniższy przykład pozwala przekonać się, w jaki sposób działają komentarze w kodzie Lessa:

```
/* Komentarz Bassa
.mixins() { ~"ta wstawka jest umieszczona w komentarzu";}
*/
```

Zagnieżdżone komentarze

Choć Less, podobnie jak PHP i JavaScript, nie pozwala na zagnieżdżanie komentarzy, to jednak nic nie stoi na przeszkodzie, by umieszczać komentarze jednowierszowe, zaczynające się od sekwencji znaków //, wewnątrz normalnych. Takie zagnieżdżone komentarze przedstawia poniższy przykład:

```
/*
// komentarz zagnieżdżony
*/
```

Komentarze specjalne

Narzędzia do minimalizacji kodu definiują czasami specjalną składnię komentarzy, by pozwolić na umieszczanie w zminimalizowanym kodzie ważnych komentarzy, takich jak informacje o licencji. Tej składni można używać, by umieszczać na początku swojego arkusza stylów postano-

wienia licencyjne. W przypadku stosowania wtyczki *clean-css* i jej minimalizatora używanego przez kompilator Lessa działający z poziomu wiersza poleceń takie ważne komentarze należy umieszczać między sekwencjami znaków `/*!` i `!*/`, jak w poniższym przykładzie:

```
/*!
bardzo ważny komentarz!
   !*/
```

Uwaga

Trzeba pamiętać, że przed zastosowaniem opcji `-clean-css` konieczne będzie zainstalowanie wtyczki Lessa *clean-css*. Można to zrobić, wykonując następujące polecenie:

```
npm install less-plugin-clean-css
```

Stosowanie zmiennych

Zmienne w Lessie ułatwiają utworzenie odpowiedniej organizacji plików i upraszczają ich utrzymanie. Pozwalają na określanie często używanych wartości w jednym miejscu, a następnie stosowanie ich w całym pozostałym kodzie Lessa. Zmiennych można używać, na przykład, do ustawiania finalnych wartości właściwości arkusza stylów. Wyobraź sobie, że już nigdy więcej nie będziesz musiał wyszukiwać w arkuszach stylów wszystkich deklaracji konkretnego koloru. A w jaki sposób działają zmienne? Otóż zmienne muszą mieć nazwy, które zaczynają się od znaku `@`.

Przykładami nazw zmiennych mogą być `@color`, `@size` czy też `@tree`. W nazwach zmiennych można umieszczać znaki alfanumeryczne, znaki podkreślenia oraz minusy. Oznacza to, że `@this-is-va` ↪`riable-name-with-35-chars` jest prawidłową nazwą zmiennej.

Niestety stosowanie znaku `@` w kodzie Lessa jest trochę niejednoznaczne. Jak wiesz z poprzedniego rozdziału, znak `@` jest także używany w nazwach parametrów wstawek. A to jeszcze nie wszystko! Ponieważ prawidłowy kod CSS jest jednocześnie prawidłowym kodem Lessa, znak `@` może się także pojawiać na początku zapytań medialnych. Jednak na podstawie kontekstu można precyzyjnie określić, czy znak `@` został użyty na początku deklaracji zmiennej. Jeśli na podstawie kontekstu nie da się jednoznacznie określić znaczenia znaku `@`, to zostanie ono opisane w treści niniejszej książki.

Zmiennej można przypisać wartość — taki zapis jest nazywany deklaracją. Wartościami zmiennych mogą być liczby, wielkości wyrażone w pikselach, łańcuchy znaków, listy, a nawet kompletne zestawy reguł. Zestaw reguł zapisany w zmiennej jest określany jako „oddzielony" (ang. *detached ruleset*).

W celu przypisania wartości do zmiennej należy użyć znaku dwukropka (`:`). Deklaracja kończy się natomiast znakiem średnika (`;`). Oto kilka przykładów deklaracji:

```
@width: 10px;
@color: blue;
@list: a b c d;
@csv-list: a, b, c, d;
@escaped-value: ~"dark@{color}";
```

Od momentu udostępnienia Lessa w wersji 1.7 w zmiennych można także zapisywać grupy właściwości, zagnieżdżone zestawy reguł, deklaracje używanych mediów, a nawet inne fragmenty kodu Lessa. Taki kod należy umieszczać wewnątrz pary nawiasów klamrowych, dokładnie tak samo jak w przypadku wstawek. Jest on określany jako oddzielony zestaw reguł.

Taki oddzielony zestaw reguł przedstawia poniższy przykład:

```
@detached-ruleset: { color: white; font-size: small; };
```

Zestaw reguł zadeklarowany w powyższym przykładzie może być używany w następujący sposób:

```
p {
  @detached-ruleset();
}
```

Po zadeklarowaniu zmiennej można używać jej w dowolnym miejscu kodu, by odwołać się do przypisanej jej wartości. Ta możliwość sprawia, że stosowanie zmiennych w kodzie Lessa zapewnia niezwykle duże możliwości.

Organizowanie plików

Jak zauważyłeś, wystarczy raz zadeklarować zmienną, by móc jej używać w dowolnym miejscu kodu. A zatem aby zmieniać wartości zmiennych, także wystarczy robić to w jednym miejscu. Na przykład kod może definiować zmienne w odrębnym pliku o nazwie *less/variables.less*. To doskonały sposób organizowania plików. Jeśli w przyszłości pojawi się konieczność wprowadzenia jakiejś zmiany, będzie wiadomo, gdzie to należy zrobić.

Wracając do przykładów resetowania właściwości CSS oraz określania postaci obramowań i sposobu wymiarowania elementów przedstawionych w poprzednim rozdziale, można przyjąć, że nasz główny plik Lessa będzie obecnie miał następującą postać:

```
@import "less/normalize.less";
@import "less/boxsizing.less";
@import "less/mixins.less";
@import "less/variables.less";
```

W powyższym kodzie dyrektywa `@import` importuje kod ze wskazanego pliku do głównego pliku Lessa. Nazwy plików są zapisywane między znakami cudzysłowu, a za nimi jest umieszczony średnik. Oprócz plików Lessa można importować arkusze stylów CSS, które nie będą przetwarzane przez dyrektywy Lessa. Takie rozwiązanie przedstawię i dokładniej opiszę w rozdziale 5., „Integracja Lessa z własnymi projektami”.

Spróbujmy teraz otworzyć w przeglądarce stronę *http://localhost/rozdzial_02/index.html*. Zostanie wyświetlona strona o bardzo prostym układzie, zawierająca nagłówek, blok treści, menu boczne oraz stopkę składającą się z trzech kolumn. Postać tej strony przedstawia rysunek 2.1. Wszystkie elementy menu mają niebieskie akcenty. Teraz otwórzmy w ulubionym edytorze pliki *less/variables.less*.

Rysunek 2.1. Układ strony określony w kodzie Lessa

Ciekawość zapewne kazała Ci już wcześniej otworzyć ten plik. Nie należy bać się jego złożoności. Zarówno ten kod, jak i cały układ mają za zadanie pokazać ogromne możliwości stosowania zmiennych i wygodę deklarowania ich w jednym miejscu. Takie rozwiązania znacznie lepiej jest zademonstrować na bardziej złożonych i realistycznych przykładach niż na kodzie liczącym kilka wierszy długości. Niemniej jednak wszystkie pozostałe przykłady przedstawione w tym rozdziale doskonale wyjaśnią zasady stosowania zmiennych. Zanim się zorientujesz, będziesz doskonale rozumiał kod umieszczony w tym pliku.

W ramach pierwszego tekstu wartość zmiennej `@dark-color` zapisanej w pliku *less/variables.less* zmieńmy z `darkblue` na `darkgreen`. Po zapisaniu pliku sprawdźmy, jakie ta zmiana dała efekty — jeśli jeszcze nie używasz funkcji Lessa `#!watch`, konieczne będzie odświeżenie strony w przeglądarce.

Teraz układ będzie miał akcenty koloru zielonego. Jeśli kogoś nie przekonała jeszcze przydatność zmiennych, to zapewne ta demonstracja rozwiała wszelkie wątpliwości. Oczywiście w praktyce nie da się zmienić wyglądu całej witryny, modyfikując pojedynczy wiersz kodu, ale ten przykład wyraźnie pokazuje, że Less potrafi znacząco ułatwić pracę projektantów.

Załóżmy, że właśnie zakończyliśmy pracę nad ciemnozielonym układem witryny i pokazujemy ją szefowi. „Świetna robota! — mówi szef. — Wiem, że prosiłem o zielony, ale jeśli nie masz

nic przeciwko, wolałbym jednak witrynę w kolorze czerwonym". Dzięki zastosowaniu zmiennych Lessa możemy się uśmiechnąć i zmienić w pliku *less/variables.less* wartość zmiennej `@dark-color` z `darkgreen` na `darkred`.

Jak widać, kod HTML strony jest przejrzysty i prosty — bez stylów umieszczanych bezpośrednio w znacznikach HTML, a nawet bez nazw klas. Pojawił się jednak inny problem, na który należy zwrócić uwagę: trzeba będzie określać nazwy zmiennych, deklarować je i zapisywać w sprytny i odpowiedni sposób. Należy przy tym zachować konsekwencję, gdyż zagadnienie to ma naprawdę duże znaczenie. Podczas organizowania zmiennych trzeba zawsze robić to w ten sam sposób, stosować te same konwencje nazewnicze i we wszystkich miejscach kodu, gdzie kontekst nie jest dostatecznie jasny, należy stosować komentarze. Pamiętaj, że ktoś inny powinien móc w dowolnym momencie przejąć prace nad kodem i zrozumieć go bez żadnej pomocy. By to zagwarantować, musisz nieco dokładniej poznać zmienne.

Określanie nazw zmiennych

Zmiennym zawsze należy nadawać znaczące i opisowe nazwy. Nazwy takie jak `@a1` i `@a2` zostaną skompilowane, ale nie są zbyt dobre, bo kiedy liczba zmiennych się powiększy lub trzeba będzie coś zmienić głęboko w kodzie, trudno będzie sobie przypomnieć, do czego służyła zmienna `@a2`. Trzeba zatem będzie odszukać jej kontekst, by dowiedzieć się, do czego była używana w plikach Lessa, albo jeszcze gorzej: trzeba będzie przeanalizować elementy HTML, by sprawdzić, jakie reguły CSS są w nich używane, aby na ich podstawie określić kontekst kodu Lessa. W takiej przykrej sytuacji będziesz musiał zacząć pracę od początku.

Przykładami dobrych nazw zmiennych są `@nav-tabs-active-link-hover-border-color` oraz `@dark-color`. Obie są znaczące i opisowe, gdyż wyjaśniają przeznaczenie zmiennej, a nie jej wartość. Proces pozwalający tworzyć takie nazwy zmiennych jest określamy mianem **semantycznego doboru nazw**. A zatem nazwa `@dark-color` jest lepsza od `@red`. Co więcej, w niektórych przypadkach warto wybierać nazwy, które będą jeszcze bardziej szczegółowe, na przykład `@brand-color`. Zmienna o takiej nazwie mogłaby zawierać kolor stanowiący główny motyw kolorystyczny projektu witryny, podobnie jak zmienna `@dark-color` zastosowana w ostatnim przykładzie. Jeśli taki kolor zostanie zmieniony, na przykład z ciemnoczerwonego na jasnozielony, to zmienna `@brand-color: lightgreen` wciąż będzie mieć sensowną nazwę. Niemniej jednak w przypadku deklaracji `@dark-color: lightgreen` czy też `@red: lightgreen` nie można powiedzieć, że nazwy zmiennych zostały właściwie dobrane.

W powyższych przykładach poszczególne słowa tworzące nazwy zmiennych są od siebie oddzielane znakami minusa, ponadto nazwy są w całości zapisywane małymi literami. Nie istnieją żadne ścisłe reguły narzucające konieczność stosowania takiego sposobu zapisu nazw; wielu programistów stosuje popularny, alternatywny sposób zapisu nazw określany jako notacja **CamelCase**. W tej notacji dwie przedstawione wcześniej nazwy zmiennych miałyby postać `@navTabsActiveLinkHoverBorderColor` i `@darkColor`. Obie przedstawione metody zapisu nazw poprawiają przejrzystość kodu.

Wskazówka

Podczas pisania kodu CSS i HTML przywykłeś zapewne do stosowania nazw składających się z dwóch słów oddzielonych znakiem minusa i zapisywanych małymi literami nazw klas, identyfikatorów, czcionek oraz innych elementów kodu. Ta konwencja została także zastosowana w niniejszej książce, a zatem wszystkie nazwy będą zapisywane małymi literami, a ich poszczególne słowa oddzielane od siebie znakiem minusa.

To, czy ktoś preferuje notację *CamelCase* czy zapis z minusami, nie ma szczególnego znaczenia. Dużo ważniejsze jest to, by wybraną konwencję nazewniczą stosować konsekwentnie i we wszystkich plikach z kodem Lessa.

Wskazówka

Stosowanie nazw ze słowami oddzielanymi znakami minusa może przysporzyć pewnych problemów w przypadku wykonywania obliczeń arytmetycznych. W takich przypadkach konieczne będzie dodawanie znaków odstępu. Na przykład deklaracja odwołująca się do wartości zmiennej `@value` pomniejszonej o 1, `@value-1`, zostanie potraktowana jako jedna nazwa zmiennej, a nie wyrażenie `@value -1`.

Stosowanie zmiennych

Wraz ze stopniowym powiększaniem się projektu w którymś momencie przestanie być możliwe zapisywanie wartości wszystkich właściwości CSS w zmiennych. Wówczas konieczne będzie podjęcie decyzji, które wartości powinny zostać zapisane w zmiennych, a które będą podane na stałe. Nie istnieją żadne ścisłe reguły określające, jak należy dokonywać takiego wyboru, ale w następnych akapitach znajdziesz porady dotyczące tego, jak to robić.

Przede wszystkim należy znaleźć takie wartości właściwości, które są używane w kodzie więcej niż jeden raz. Wielokrotne występowanie wartości jest oczywistym sygnałem sugerującym możliwość zastosowania zmiennej. W poprzednim przykładzie taką wartością właściwości jest zmienna `@dark-color`.

Oprócz tego zmienne można stosować we właściwościach używanych do dostosowywania wyglądu projektu. Przykładem takiej zmiennej może być `@basic-width`.

Warto też zastanowić się nad tworzeniem zmiennych do przechowywania komponentów, które będą wielokrotnie stosowane w kodzie. Wracając do przedstawionego przykładu, można by uznać, że nagłówek strony będzie używany także w innych projektach. Aby zapewnić taką możliwość, należałoby utworzyć nowy plik, *less/header.less*, i zaimportować go do pliku głównego przy użyciu dyrektywy:

```
@import "less/header.less";
```

Organizowanie zmiennych

Aby zapewnić możliwość wielokrotnego stosowania komponentów, można umieszczać takie komponenty lub funkcje w odrębnych plikach Lessa i odpowiednio do nich dostosować używane zmienne. W ramach prezentacji takiego rozwiązania podzielimy nasz przykładowy plik Lessa na trzy nowe pliki: *less/header.less*, *less/content.less* oraz *less/footer.less*.

Oto zawartość pliku *less/header.less*:

```
header {
background-color: @header-dark-color;
min-height: @header-height;
padding: 10px;

.center-content;
.border-radius(15px);
.box-shadow(0 0 10px, 70%);

h1 {
color: @header-light-color;
  }
}
```

Warto zwrócić uwagę, że zmienna `@dark-color` została zmieniona na `@header-dark-color`. Wyświetlmy teraz w przeglądarce stronę *http://localhost/rozdzial_02/project.html*, a w edytorze otwórzmy plik *less/project.less*, aby się przekonać, jakie zmiany zostały w nim wprowadzone i jakie są ich efekty.

Teraz dołączmy do głównego pliku *less/project.less* plik *less/header.less*, używając dyrektywy `@import "header.less";`, a w pliku *less/variablesproject.less* utwórzmy sekcję zawierającą poniższy fragment kodu:

```
/* nagłówek (header) */
@header-dark-color: @dark-color;
@header-light-color: @light-color;
@header-height: 75px;
```

Instrukcja `@header-dark-color: @dark-color;` zapisuje wartość zmiennej `@dark-color` w zmiennej `@header-dark-color`. Teraz, w dokładnie taki sam sposób, należy określić zawartość plików *less/content.less* i *less/footer.less* i dołączyć je do głównego pliku Lessa. Jak można zauważyć, po wprowadzeniu tych zmian wygląd strony *http://localhost/rozdzial_02/project.html* się nie zmienił.

A teraz spróbujmy otworzyć w edytorze plik *less/variablesproject.less* i zmienić sekcję dotyczącą stopki w następujący sposób:

```
/* stopka (footer) */
@footer-dark-color:  darkgreen;
@footer-light-color:  lightgreen;
@footer-height: 100px;
@footer-gutter: 10px;
```

Jak widzisz, elementy wyświetlone w stopce strony są teraz zielone.

Ostatnia deklaracja wygrywa!

W poprzednim rozdziale przeczytałeś o kaskadzie CSS, której ostatnia reguła głosi, że o ile wartości pozostałych reguł będą identyczne, to wygrywa wartość zadeklarowana jako ostatnia. Less stosuje dokładnie tę samą strategię — w całym poprzednim kodzie będzie stosowana wartość podana w ostatniej deklaracji zmiennej. Pod tym względem zmienne Lessa można by porównać ze stałymi stosowanymi w innych językach programowania. W poniższym przykładzie, zgodnie z zasadą, że ostatnia deklaracja wygrywa, wartością właściwości `property` będzie `2`:

```
@value: 1;
.class{
property: @value;
}
@value: 2;
```

Ten kod Lessa zostanie skompilowany do następującego kodu CSS:

```
.class{
property: 2;
}
```

Okazuje się, że Less w pierwszej kolejności wczytuje cały kod. Tam, gdzie ma zostać użyta wartość zmiennej, Less zastosuje ostatnią zadeklarowaną bądź ostatnią odczytaną wartość. Reguła określająca, że wygrywa ostatnia deklaracja zmiennej, odnosi się wyłącznie do deklaracji zdefiniowanych w tym samym zasięgu.

W większości języków programowania zasięgi są definiowane jako fragmenty kodu, które kompilator może przetwarzać niezależnie od pozostałych. Funkcje i klasy mogą mieć swoje własne zasięgi. W przypadku Lessa swoje własne zasięgi mają wstawki, które opisałem pod koniec tego rozdziału.

Poniższy przykład pokazuje, że zgodnie z deklaracją umieszczoną w zasięgu wstawki właściwości `property` zostanie przypisana wartość `3`:

```
@value: 1;
.mixin(){
@value: 3;
property: @value;
}
.class{
```

```
.mixin;
}
@value: 2;
```

Ten kod Less zostanie skompilowany do poniższego kodu CSS:

```
.class{
property: 3;
}
```

Powyższy kod pokazuje, że nie można zmieniać wartości zmiennej w trakcie kompilacji. To właśnie ten fakt sprawia, że zmienne teoretycznie można uznać za stałe. Można je porównać z umieszczoną w kodzie definicją liczby `PI`, która zawsze jest taka sama. Taka wartość zostałaby zadeklarowana tylko raz, jako `PI = 3.14`, i nie zmieniałaby się w całym kodzie programu. Dlatego w kodzie Lessa, jeśli zmienne mają być używane jako stałe, należy je deklarować tylko jeden raz.

Powtarzane deklaracje zmiennych oraz zasada, że wygrywa ostatnia z deklaracji, będą używane w wielu projektach Lessa jako mechanizm dostosowywania.

W ramach prezentacji efektów ponownego zadeklarowania zmiennej utwórzmy nowy plik, *less/customized.html*, o następującej zawartości:

```
@import "styles.less";
@dark-color: black;
@basic-width: 940px;
```

Potem odwołajmy się do niego w dokumencie *customized.html*, używając znacznika `link` o następującej postaci:

```
<link rel="stylesheet/less" type="text/css" href="less/customized.less" />
```

Kiedy teraz wyświetlimy stronę *customized.html* w przeglądarce, przekonamy się, że udało się nam stworzyć zmodyfikowaną wersję układu za pomocą jedynie trzech wierszy kodu!

Deklaracje zmiennych nie są statyczne!

Choć zmienne działają jak gdyby były stałymi, nie oznacza to wcale, że ich deklaracje są niezmienne lub statyczne. Przede wszystkim jednej zmiennej można przypisać wartość innej. Takie rozwiązanie przedstawia poniższy fragment kodu:

```
@var2 : 1;
@var1 : @var2;
@var2 : 3;
```

W efekcie zmienna `@var1` przyjmie wartość `3`, a nie `1`. Tworzenie odwołania w formie reguły nie jest tu wcale konieczne, gdyż i tak stosowana jest zasada, że ostatnia deklaracja wygrywa. Zatem zmienna `@var1` przyjmie wartość zmiennej `@var2` z jej ostatniej deklaracji.

W przykładowym kodzie można także zauważyć deklarację `@light-color: lighten(@dark-co ↪lor,40%);`. Funkcja `lighter()` to tak zwana wbudowana funkcja Lessa (wbudowane funkcje Lessa omówię dokładniej w rozdziale 3., „Reguły zagnieżdżone, działania oraz funkcje wbudowane"). Wywołanie funkcji `ligther()` sprawia, że zmiennej `@light-color` zostaje przypisana wartość koloru wyznaczona na podstawie wartości zmiennej `@dark-color`. Warto także zwrócić uwagę na ostatnią deklarację zmiennej `@dark-color`, gdyż podczas wyznaczania koloru zostanie użyta wartość podana właśnie w niej.

Dynamiczne deklaracje zmiennych zapewniają dużą elastyczność, trzeba jednak pamiętać o tym, że wartość należy zadeklarować tylko raz, a po tej deklaracji nie można już jej zmieniać.

Leniwe wczytywanie

Zanim zakończę omawianie zmiennych i przejdę do wstawek, wspomnę jeszcze o leniwym wczytywaniu (ang. *lazy loading*). W kontekście języków programowania termin ten oznacza opóźnienie inicjalizacji obiektu aż do momentu, gdy stanie się on potrzebny. Leniwe wczytywanie jest przeciwieństwem aktywnego wczytywania. Less stosuje leniwe wczytywanie, co oznacza, że zmiennych nie trzeba deklarować, dopóki nie będę potrzebne.

Zrozumienie teoretycznych aspektów tego zagadnienia jest ważne, ale teraz skupmy się na tym, jak leniwe wczytywanie działa w praktyce. W tym celu przeanalizujmy następujący przykład:

```
.class {
property: @var;
}
@var: 2;
```

Ten kod Lessa zostanie skompilowany do następującego kodu CSS:

```
.class {
property: 2;
}
```

Interpolacja zmiennych

W kodzie Lessa zmiennych można używać w nazwach selektorów, nazwach właściwości, adresach URL, a nawet w dyrektywach importu. Kompilator zastąpi odwołanie do zmiennej, zastępując je wartością zmiennej zapisanej w formie łańcucha znaków.

Aby uniknąć ewentualnych niejednoznaczności, nazwy zmiennych można zapisywać w nawiasach klamrowych. Na przykład poniższy fragment kodu Lessa:

```
@var: less;
.@{var} {
  property: ~"@{var}-5";
}
```

zostanie skompilowany do następującego kodu CSS:

```
.less {
  property: less-5;
}
```

Począwszy od wersji 1.6 Lessa, dokładnie w taki sam sposób można stosować zmienne w nazwach właściwości. Na przykład poniższy fragment kodu Lessa:

```
@property: width;
.fixed {
  @{property}: 100%;
  max-@{property}: 500px;
}
```

zostanie skompilowany do następującego kodu CSS:

```
.fixed {
  width: 100%;
  max-width: 500px;
}
```

W niektórych przypadkach wartości będą musiały być zapisane w cudzysłowach (reguły stosowania wartości w takich sytuacjach opisałem w następnym punkcie rozdziału). Mechanizmu interpolacji zmiennych można także użyć do stworzenia *zmiennych zmiennych*, których przykład przedstawia poniższy fragment kodu:

```
@variable: red;
@color: "variable";
p {
  color: @@color;
}
```

Powyższy kod zostanie skompilowany do kodu CSS o następującej postaci:

```
p {
  color: red;
}
```

Taki sposób odwoływania się do zmiennych może mieć tylko jeden poziom zagnieżdżenia, co oznacza, że nie można użyć kodu o postaci `@@@variable`.

Zapisywanie wartości

Less jest rozszerzeniem języka CSS. Oznacza to, że w przypadku próby skompilowania kodu CSS nieprawidłowego lub zapisanego w niestandardowy sposób, którego kompilator nie jest w stanie rozpoznać, Less wyświetli komunikat o błędzie. Trzeba przy tym pamiętać, że Less sprawdza jedynie składnię, nie dbając o to, czy przypisywane wartości mają sens. W poniższym przykładzie właściwości `width` jest przypisywana wartość reprezentująca kolor:

```
p {
width: darkblue;
}
```

Składnia CSS przewiduje natomiast, że właściwość `width` może przyjmować wartości `auto`, `initial`, `inherit` oraz wartość liczbową z określeniem jednostki typu `px`, `cm`, `em` itd. bądź też wartość procentową.

Niektóre przeglądarki definiują właściwości, używając przy tym nieprawidłowego kodu CSS. Najbardziej znanym przykładem takiego działania jest kod o przykładowej postaci `property: ms:somefunction()`. Niektóre z takich reguł można zastępować regułami charakterystycznymi dla konkretnych przeglądarek. Należy jednak zwrócić uwagę, że nieprawidłowe wartości właściwości nie zostaną skompilowane przez Lessa. Aby w skompilowanym kodzie CSS uzyskać regułę o postaci `property: ms:somefunction()`, należałoby użyć następującego kodu Lessa:

```
selector {
  property: ~"ms:somefunction()";
}
```

Taki kod zostanie skompilowany bez żadnych problemów i wygeneruje następujący kod CSS:

```
selector {
  property: ms:somefunction();
}
```

Kod `~"ms:somefunction()"` korzysta ze specjalnego sposobu zapisu, który opiszę dokładniej w dalszej części rozdziału.

Nowa funkcja, `calc()`, wprowadzona w CSS3, jest rdzennym sposobem CSS umożliwiającym wykonywanie prostych obliczeń arytmetycznych i stanowiącym zamiennik wartości o dowolnej długości.

W poniższym przykładzie, zarówno w przypadku kompilacji, jak i importowania, Less nie zwróci prawidłowej wartości:

```
@aside-width: 80px;
.content {
  width: calc(100% - @aside-width)
}
```

Powyższy kod Lessa zostanie skompilowany na następujący kod CSS:

```
.content {
  width: calc(20%);
}
```

W powyższym kodzie Lessa `@aside-width: 80px` jest deklaracją zmiennej o nazwie `aside-width`. Wartość tej zmiennej to 80 pikseli (`80px`). Więcej informacji o zmiennych można znaleźć w kolejnych akapitach. Najważniejsze jest jednak to, że wynik przetwarzania powyższego kodu Lessa jest nieprawidłowy (albo co najmniej niezgodny z oczekiwaniami), gdyż funkcja `calc()` powinna zostać wywołana w czasie wyświetlania strony. W takim przypadku funkcja ta dysponuje możliwością mieszania jednostek, takich jak wartości procentowe i liczby pikseli. Jednak w powyższym przykładzie właściwości `width` w selektorze `.content` przypisywane jest `100%` dostępnej szerokości (czyli cała dostępna szerokość) pomniejszone o `80px` (wartość wyrażającą liczbę pikseli).

Problem ten można rozwiązać, stosując przedstawiony wcześniej, specjalny sposób zapisu. W przypadku Lessa, aby łańcuch znaków nie został przetworzony, należy go zapisać w cudzysłowie (`""`) poprzedzonym tyldą (~). A zatem w powyższym przykładzie wywołanie funkcji `calc()` należałoby zapisać jako `~"calc(100% - @{aside-width})"`.

Koniecznie należy zwrócić uwagę, że nazwa zmiennej `aside-width` została zapisana w nawiasach klamrowych — jest to przykład opisywanej w poprzednim punkcie rozdziału interpolacji zmiennych. Jednak w przypadku opisywanego tu sposobu zapisu zmiennych, określanego w języku angielskim terminem *escaping*, wszystko, co zostało zapisane w nawiasach, zostanie użyte tak, jak zostało zapisane, niemal bez zmian. Jedynym wyjątkiem od tej zasady jest interpolacja zmiennych.

Łańcuchy są sekwencjami znaków. W kodzie Lessa i CSS za łańcuchy uznaje się wartości zapisywane między znakami cudzysłowu. Bez opisywanego tu sposobu zapisu Less kompiluje wszystkie łańcuchy umieszczone w kodzie do postaci łańcuchów CSS.

Na przykład w kodzie CSS właściwość o postaci `width: "calc(100 - 80px)"` nie ma większego sensu, podobnie zresztą jak właściwość `width: calc(100% - @aside-width)`, gdyż wyrażenie `@aside-width` nie ma żadnego znaczenia.

A zatem, korzystając z opisywanego tu sposobu zapisu i z interpolacji zmiennych, można użyć następującego kodu Lessa:

```
@aside-width: 80px;
.content{
  width: ~"calc(100% - @{aside-width});"
}
```

Zostanie on skompilowany do następującego kodu CSS:

```
.content {
  width: calc(100% - 80px);
}
```

Wskazówka

W konkretnym przypadku stosowania funkcji `calc()` kompilator Lessa udostępnia opcję `strict-math` (począwszy od wersji 1.4). Jest ona używana w postaci `-strict-math=on`, gdy kompilator uruchamiany jest z poziomu wiersza poleceń, bądź jako `strictMath: true` w przypadku stosowania kompilacji po stronie klienta. Kiedy opcja `strictMath` jest włączona, wywołanie o postaci `calc(100% - @aside-width)` zostanie skompilowane do postaci kodu `calc(100% - 80px);`. Trzeba przy tym pamiętać, że podczas prac nad kolejnymi wersjami kompilatora Lessa — 1.6, 1.7 oraz 2.0 — wprowadzono wiele zmian w sposobie działania tej opcji.

Stosowanie wstawek

Wstawki odgrywają bardzo ważną rolę podczas stosowania Lessa. Przedstawiłem je już w poprzednim rozdziale, na przykładzie wstawki dodającej do wybranego elementu obramowanie i zaokrąglone wierzchołki. Konwencje używane do określania nazw wstawek pochodzą z programowania obiektowego. Wstawki wyglądają jak funkcje znane z funkcyjnych języków programowania, lecz w rzeczywistości działają jak makra języka C. Umożliwiają one dodanie do wybranej klasy wszystkich właściwości innej klasy poprzez dodanie jej nazwy jako jednej z właściwości klasy, której postać należy określić. Poniższy kod przedstawia przykład wstawki i jej zastosowania:

```
.mixin(){
  color: red;
  width: 300px;
  padding: 0 5px 10px 5px;
}
p{
  .mixin();
}
```

A oto postać wynikowego kodu CSS uzyskanego po kompilacji tego przykładu:

```
p{
  color: red;
  width: 300px;
  padding: 0 5px 10px 5px;
}
```

Ten wynikowy kod CSS zastosowany w witrynie sprawi, że wszystkie znaczniki `<p>` będą miały postać określoną przez właściwości zdefiniowane przez wstawkę `mixin()`. Zaletą jest to, że tę samą wstawkę będzie można zastosować w wielu różnych klasach. Jak można się było przekonać na przykładzie przedstawionym w rozdziale 1., takie właściwości wystarczy zadeklarować tylko jeden raz.

Spróbujmy teraz otworzyć plik *less/mixin.less* dostępny w przykładach dołączonych do książki, w katalogu *rodzial_02*. W przykładach stosowanych w tej książce wszystkie wstawki są umieszczane

w jednym pliku. Wewnątrz tego pliku poszczególne wstawki można uporządkować w zależności od ich funkcji. Takie rozwiązanie, polegające na zapisywaniu wstawek w jednym pliku, ma tę zaletę, że może nas uchronić przed uszkodzeniem kodu podczas usuwania lub modyfikowania innych plików Lessa. W przykładach do tego rozdziału znajdują się dwa pliki Lessa, *less/header.less* oraz *less/footer.less*, których wstawka `border-radius` jest używana. Nie chcielibyśmy, aby w razie wprowadzania zmian w zawartości pliku *header.less* pojawiły się jakieś problemy w kodzie drugiego pliku. Oczywiście nie chcielibyśmy także doprowadzić do sytuacji, w której kod wstawki powtarzałby się w kilku różnych plikach.

Wstawka `box-sizing` zadeklarowana w pliku *less/boxsizing.less* zostanie opisana jako przypadek szczególny. Ma ona bowiem wpływ na wszystkie elementy i pozwala globalnie zmienić ustawienie właściwości `box-sizing`.

Plik *less/mixins.less* zawiera cztery wstawki, które omówię w kolejnych punktach rozdziału. Dwie spośród tych wstawek, a mianowicie `box-shadow` i `clearfix`, mają bardzie złożoną strukturę, korzystającą, na przykład, z zagnieżdżania, ale akurat te dwie wstawki opiszę szczegółowo w następnym rozdziale.

Proste wstawki

W poprzednim rozdziale przedstawiłem kilka przykładów prostych wstawek, takich jak `bordered` lub `roundedcornersmixin`. Prosta wstawka wygląda tak samo jak zwyczajna definicja klasy CSS. Wstawki są wywoływane w klasach i dodają do tych klas swoje właściwości.

W pliku *less/mixins.less* dostępnym w przykładach dołączonych do książki zdefiniowana została wstawka `.center-content`, która przypisuje właściwości `margin` wartość `0 auto`. Ta wstawka jest używana do wyśrodkowania nagłówka układu, pojemnika na treść strony oraz stopki.

> **Wskazówka**
>
> Warto zwrócić uwagę, że przedstawiona tu wstawka `center-content` nie jest jedynym rozwiązaniem zapewniającym efekt wyśrodkowania. W przypadku używanego tu przykładowego układu strony taki sam efekt można uzyskać, umieszczając nagłówek, treść oraz stopkę w jakimś nadrzędnym, wyśrodkowanym pojemniku. Można się także zastanawiać nad nazwą tej wstawki. Gdybyśmy się bowiem zdecydowali, żeby nie wyśrodkowywać treści strony, to taka nazwa wstawki przestałaby mieć sens.

Spróbujmy teraz usunąć ze wstawki właściwość `margin: 0 auto`, odpowiadającą za wyśrodkowanie treści. Efekty tej modyfikacji będzie można zobaczyć po odświeżeniu strony wyświetlonej w przeglądarce.

Wstawki z parametrami

Wspomniałem wcześniej, że wstawki działają jak funkcje stosowane w funkcyjnych językach programowania. Oznacza to, że wstawki, podobnie jak funkcje, mogą mieć parametry. Parametr to wartość zastosowana w połączeniu ze wstawką, a nazwa parametru jest używana w kodzie wstawki do odwoływania się do jego wartości. Poniższy kod przedstawia przykład wstawki z parametrem i jej zastosowania:

```
.mixin(@parameter){
  property: @parameter;
}
.class1 {.mixin(10);}
.class2 {.mixin(20);}
```

Ten kod zostanie skompilowany do kodu CSS o następującej postaci:

```
.class1 {
  property: 10;
}
.class2 {
  property: 20;
}
```

Ten przykład wyraźnie pokazuje, że dzięki parametryzacji wstawki są narzędziem o ogromnych możliwościach. Można ich bowiem wielokrotnie używać, by określać właściwości zależne od wartości parametrów.

Wartości domyślne

Parametry mogą mieć opcjonalne wartości domyślne, definiowane następująco: `.mixin(@pa ➥rameter:defaultvalue);`. Sposób działania tych domyślnych wartości parametrów można przeanalizować na przykładzie wstawki `border-radius` zdefiniowanej w pliku *less/mixins.less*:

```
.border-radius(@radius: 10px)
{
    -webkit-border-radius: @radius;
    -moz-border-radius: @radius;
    border-radius: @radius;
}
```

Warto zwrócić uwagę, że domyślną wartością parametru `@radius` jest `10px`. Jeśli powyższa wstawka zostanie wywołana bez określania wartości parametru, to zostanie zastosowana wartość domyślna. Przeanalizujmy ten oto przykład kodu Lessa:

```
div {
  border-radius(); // bez argumentów
  &.small {
```

```
    border-radius(5px);
  }
}
```

Powyższy kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
div {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;
}
div.small {
  -webkit-border-radius: 5px;
  -moz-border-radius: 5px;
  border-radius: 5px;
}
```

W powyższym kodzie warto zwrócić uwagę na klasę `.small`, która została zagnieżdżona wewnątrz selektora `div`. Kod używa także znaku `&`, by odwołać się do selektora nadrzędnego. Więcej informacji na temat zagnieżdżania selektorów i zastosowania znaku `&` można znaleźć w rozdziale 3., „Reguły zagnieżdżone, działania oraz funkcje wbudowane".

Konwencje nazewnicze i sposoby wywoływania wstawek

W niniejszej książce wstawki mają znaczące, opisowe nazwy, które, podobnie jak nazwy zmiennych, są zapisywane z wykorzystaniem znaku minusa. Stosowanie znaczących i opisowych nazw upraszcza utrzymanie kodu i ułatwia innym osobom jego zrozumienie. Nazwy parametrów i zmiennych zaczynają się od znaku `@`. Zazwyczaj na podstawie kontekstu bez trudu będzie można określić, czy chodzi o zmienną czy o parametr wstawki.

Aby lepiej zrozumieć parametry i zmienne, przeanalizujmy następujący przykład:

```
@defaulvalue-parameter1 :10;
.mixin(@parameter1: @defaulvalue-parameter1)
{
  property: @parameter1;
}
.class {
  .mixin
}
```

Powyższy kod zostanie skompilowany do następującej postaci:

```
.class{
  property: 10;
}
```

W przykładzie tym należy zwrócić uwagę na to, że `@defaultvalue-parameter1` jest zmienną. Kolejny przykład demonstruje zagadnienie zasięgu we wstawkach:

```
@defaulvalue-parameter1 :10;
.mixin(@parameter1: @defaulvalue-parameter1){
  property: @parameter1;
}
.class {
  .mixin
}
@parameter1 : 20;
```

Powyższy kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
.class{
  property: 10;
}
```

W tym przykładzie ostatnia deklaracja `@parameter1` jest umieszczona poza wstawką, przez co wartość będzie wynosić `10`.

Stosowanie wstawek z większą liczbą parametrów

Wstawki mogą mieć więcej niż jeden parametr, przy czym wówczas nazwy poszczególnych parametrów muszą być od siebie oddzielone znakiem przecinka. Także w wielu innych, funkcyjnych językach programowania przecinek jest często stosowany jako znak separatora parametrów. Przecinki są jednak w tym kontekście nieco niejednoznaczne, gdyż nie tylko służą do oddzielania parametrów, lecz również są stosowane do separacji elementów list zapisanych w formacie **cvs** (ang. *comma-separated values* — wartości rozdzielone przecinkami).

Wstawka `.mixin(a,b,c,d)` będzie wywoływana z czterema parametrami. Dokładnie tak samo będzie wywoływana wstawka `.mixin(a;b;c;d)`. A teraz przeanalizujmy przykład wywołania wstawki, które będzie miało następującą postać: `.mixin(a,b,c;d)`. W tym przypadku wstawka ma tylko dwa parametry, z których pierwszy jest listą składającą się z trzech elementów. Jeśli w liście parametrów wstawki zostanie odnaleziony choćby jeden znak średnika, to tylko znaki średnika będą uznawane za separatory parametrów. Poniższy przykład przedstawia efekt, jaki wywoła zastosowanie w liście parametrów dodatkowego średnika:

```
.mixin(@list){
  property: @list;
}
.class{ .mixin(a,b,c,d);} // uwaga na dodatkowy znak średnika!
```

Ten kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
.class{
  property: a, b, c, d;
}
```

Bez tego znaku średnika na końcu wstawka zostałaby wywołana z czterema parametrami. W takim przypadku kompilator Lessa wyświetliłby błąd *RuntimeError: No matching definition found for .mixin(a, b, c, d)*[1], oczekiwałby bowiem znalezienia wstawki zadeklarowanej w następujący sposób: `.mixin(@a, @b, @c, @d)`.

Kolejny przykład przedstawia „przeciążanie" wstawek — sytuację, w której kompilator skompiluje tę wstawkę, która pasuje do wywołania:

```
.mixin(@color; @width) {
  border: 1px solid @color;
  width: 50px;
}
.mixin(@color;) {
  color: green;
}
p {
  &.onlycolor {
    .mixin(green);
  }
  &.including-border {
    .mixin(green;500px);
  }
}
```

Powyższy kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
p.onlycolor {
  color: green;
}
p.including-border {
  border: 1px solid green;
  width: 50px;
}
```

Powyższy przykład wyraźnie pokazuje, że Less pozwala na tworzenie wstawek o tej samej nazwie. W przypadku odnalezienia wstawek o identycznych nazwach kompilator Lessa użyje wstawki o odpowiedniej liczbie parametrów bądź też, jeśli żadna wstawka pasująca do wywołania nie zostanie odnaleziona, zgłosi błąd. Takie dobieranie wstawek na podstawie liczby parametrów można by porównać z przeciążaniem metod — rozwiązaniem stosowanym w wielu językach programowania.

Jeśli wywołanie pasuje do kilku wstawek, jak w poniższym przykładzie, to kompilator użyje wszystkich pasujących wstawek:

```
.mixin(@a){
  property-a: @a;
}
```

[1] Nie znaleziono definicji pasującej do .mixin(a, b, c, d) — *przyp. tłum.*

```
.mixin(@b){
  property-b: @b;
}
class{
  .mixin(value);
}
```

Ten kod Lessa zostanie skompilowany do następującego kodu CSS:

```
class {
  property-a: value;
  property-b: value;
}
```

Złożona wstawka generująca liniowy gradient tła

Dysponujesz już wystarczającą wiedzą teoretyczną, by móc tworzyć bardziej złożone wstawki. W kolejnym przykładzie do trzech elementów umieszczonych w stopce strony dodamy tła o postaci gradientu liniowego składającego się z trzech kolorów, generowane przez odpowiednio sparametryzowaną wstawkę.

Finalną postać elementów stopki przedstawia rysunek 2.2.

Rysunek 2.2. Gradienty liniowe w tle elementów dodane przy użyciu wstawki Lessa

Te gradienty zostały wybrane celowo — ze względu na swoją złożoność i dobrze opisane zmiany, jakim podlegały. W efekcie napiszemy złożoną wstawkę, która bez wątpienia nie jest doskonała, lecz z pewnością znacząco poprawi postać wyświetlanych elementów. Można mieć jednak pewność, że tę wstawkę trzeba będzie od czasu do czasu poprawiać, ze względu na zaprzestawanie obsługi starszych przeglądarek, pojawianie się nowych, zmiany w specyfikacji oraz nowe informacje o gradientach. Dodatkowe przykłady zastosowania gradientów można znaleźć na stronie *https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_gradients*.

Tych koniecznych zmian nie sposób uniknąć, można jednak zminimalizować czas niezbędny do zapewnienia aktualizacji używanych wstawek. Less gwarantuje, że wszystkie gradienty tła będą tworzone na podstawie tej samej wstawki, zdefiniowanej w jednym miejscu.

Najprościej rzecz ujmując, gradienty CSS są definiowane jako obrazy tła. Z tego powodu tworzy się je, korzystając z właściwości `background-image`.

W tej książce do określania gradientów będzie używana właściwość `background-image`. Można jednak znaleźć (w internecie i zapewne w innych książkach) także przykłady, w których do tego samego celu jest używana właściwość `background`. Nie stanowi to wielkiej różnicy. CSS definiuje różne właściwości określające postać tła, takie jak `background-image`, `background-color`, `background-size` oraz `background-position`. `Background` to skrótowa właściwość pozwalająca określić wartości wszystkich tych właściwości w jednej regule. Jeśli pierwszą wartością określoną we właściwości `background` będzie obraz lub, jak w tym przypadku gradient, to wszystkie pozostałe właściwości tła przyjmą swoje domyślne wartości.

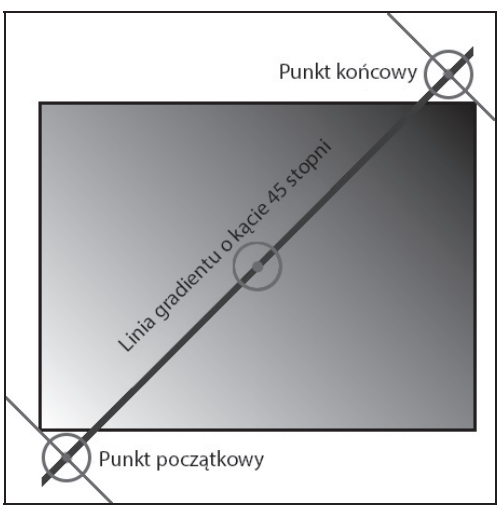
Tworzenie wstawki można zacząć od przygotowania następującej listy wymagań:

- Wstawka musi dawać możliwość określania kierunku gradientu, wyrażonego w stopniach.
- Gradient będzie się składał z trzech kolorów.
- Następnie można określić listę przeglądarek oraz ich wersji, które wstawka ma obsługiwać.

Oto pierwsze wiersze kodu wstawki:

```
.backgroundgradient(@deg: 0deg; @start-color: green; @between-color: yellow;
↪@end-color: red; @between:50%)
{
  background-image: linear-gradient(@deg, @start-color, @between-color
  ↪@between, @end-color);
}
```

Rysunek 2.3 ilustruje przykładową postać gradientu liniowego. Został on stworzony na podstawie rysunku opublikowanego 11 września 2013 roku na stronie *https://drafts.csswg.org/css-images-3/*.



Rysunek 2.3. Jeden ze sposobów prezentacji gradientu o kącie 45 stopni

Wstawka tworząca gradient tła ma pięć parametrów, które zostały opisane na poniższej liście:

- Pierwszy parametr określa kierunek gradientu wyrażony w stopniach. Podana liczba stopni określa kąt pomiędzy osią pionową a kierunkiem gradientu. Opisywanie kierunku rozpoczyna się u dołu elementu. Kąt wynoszący 0 stopni odpowiada gradientowi, który rozpoczyna się u dołu i zmienia ku górze. Następnie kąt zmienia się zgodnie z ruchem wskazówek zegara, tak że kąt o wartości 90 odpowiada gradientowi zmieniającemu się od lewej do prawej, i tak dalej.
- Kolejne trzy parametry opisują trzy kolory gradientu, przy czym dla każdego z nich została określona wartość domyślna.
- Piąty, ostatni parametr określa miejsce, w którym środkowy kolor przyjmie swoją prawdziwą wartość. Wartością tego parametru jest wartość procentowa szerokości elementu, w którym jest wyświetlany gradient. Dla pierwszego i trzeciego koloru wartości te wynoszą odpowiednio `0` i `100`.

Nowoczesne przeglądarki, takie jak Internet Explorer 11, Firefox 16+, Opera 12.10+, Safari 7+ oraz Chrome 26+, obsługują właściwości `background-image`. Jednak z myślą o starszych przeglądarkach należy dodać reguły charakterystyczne dla nich. Pierwszym problemem, jaki się z tym wiąże, jest to, że różne reguły charakterystyczne dla konkretnych przeglądarek stosują różne sposoby określania kąta gradientów. Aby wyeliminować ten problem, można zastosować korekcję 90 stopni, używając w tym celu następującego kodu:

```
.backgroundgradient(@deg: 0deg; @start-color: green; @between-color: yellow;
➥@end-color: red; @between:50%){
@old-angle: @deg – 90deg;
-ms-background-image: linear-gradient(@old-angle , @start-color, @between-color
➥@between, @end-color);
background-image: linear-gradient(@deg, @start-color, @between-color @between,
➥@end-color);
}
```

Właściwość z prefiksem `-ms` jest stosowana przez przeglądarkę IE10, gdyż wcześniejsze wersje nie obsługują obrazów tła. Alternatywnym rozwiązaniem może być dodanie filtra pozwalającego na wyświetlanie gradientów określanych przy użyciu dwóch kolorów. Nie ma możliwości zastosowania tego filtra wraz z obrazem zastępczym, dlatego też z myślą o przeglądarkach bazujących na silniku WebKit, takich jak Chrome i Safari, konieczne będzie utworzenie gradientu przy użyciu funkcji `-webkit-linear-gradient`. Niemniej jednak starsze wersje tych przeglądarek wymagają tworzenia gradientów za pomocą funkcji `-webkit-gradient`. Trzeba przy tym pamiętać, że funkcja `-webkit-gradient` ma niestandardową składnię. Oto ostateczna postać wstawki generującej gradient:

```
.backgroundgradient(@degrees: 0deg; @start-color: green; @between-color:yellow;
➥@end-color: red; @between:50%){
  background-image: -moz-linear-gradient(@degrees, @start-color 0%,
  ➥@between-color @between, @end-color 100%);
```

```
  background: -webkit-gradient(linear, left top, left bottom,
  ↪color-stop(0%, @start-color), color-stop(@between,@between-color),
  ↪color-stop(100%,@end-color));
  background-image : -webkit-linear-gradient(@degrees, @start-color 0%,
  ↪@between-color @between, @end-color 100%);
  background-image: -o-linear-gradient(@degrees, @start-color 0%,
  ↪@between-color @between, @end-color 100%);
  background-image: -ms-linear-gradient(@degrees, @start-color 0%,
  ↪@between-color @between, @end-color 100%);
  background-image: linear-gradient((@degrees - 90deg), @start-color 0%,
  ↪@between-color @between, @end-color 100%);
  filter: progid:DXImageTransform.Microsoft.gradient( startColorstr=
  ↪'@startcolor', endColorstr='@endcolor',GradientType=0 );
}
```

Powyższy przykład pokazuje, że nawet w przypadku korzystania z Lessa tworzony kod może być złożony. Jeśli złożoność takiego kodu nie wynika z konieczności obsługi wielu przeglądarek, to można zauważyć zalety, jakie zapewnia stosowanie Lessa — pozwala on bowiem umieścić cały ten kod w jednym miejscu.

Kod przedstawiony w tym punkcie rozdziału można znaleźć w przykładach dołączonych do książki, w plikach *directivebackgrounds.html* oraz *less/directivebackgrounds.less*, umieszczonych w katalogu *rodzial_02*. Jeśli zobaczywszy ten przykład, zastanawiasz się, po co w ogóle stosować gradienty tła, to zajrzyj na stronę *http://lea.verou.me/css3patterns/*, by się przekonać, co przy ich użyciu można stworzyć.

W niniejszej książce zakładamy, że najlepszą z dostępnych praktyk jest stosowanie wtyczki *autoprefixer*, pozwalającej dodawać do kodu prefiksy przeglądarek. Jak już wspomniałem, wtyczka ta nie dodaje niestandardowych właściwości ani rozwiązań typu *polyfill*. Jeśli pojawia się konieczność zastosowania niestandardowego kodu CSS w celu obsługi starszych przeglądarek, użycie wstawek wydaje się być właściwym rozwiązaniem. Jak pokazał powyższy przykład, dzięki parametrom wstawki umożliwiają tworzenie gradientów o różnej postaci przy użyciu tego samego kodu.

Zmienne specjalne @arguments i @rest

Less definiuje dwie zmienne specjalne. Pierwsza z nich, `@arguments`, zawiera listę wszystkich argumentów. Zmienna ta jest dostępna jedynie we wstawkach. W kodzie Lessa elementy list są oddzielane od siebie odstępami, dzięki czemu zmiennej `@arguments` można używać we właściwościach, które wymagają listy wartości. Wartości skrótowych właściwości `margin` i `padding` można określać przy użyciu list, co zostało wykorzystane w poniższym przykładzie:

```
.setmargin(@top:10px; @right:10px; @bottom: 10px; @left 10px;){
  margin: @arguments;
}
p{
  .setmargin();
}
```

Ten kod Lessa zostanie skompilowany do następującego kodu CSS:

```
p {
  margin: 10px 10px 10px 10px;
}
```

Drugą ze zmiennych specjalnych udostępnianych przez Less jest `@rest`. Nazwa `@rest...` pozwala odwołać się do wszystkich argumentów podanych w wywołaniu wstawki i niemających swoich odpowiedników na liście parametrów.

Zastosowanie zmiennej `@rest...` pozwala zatem podawać w wywołaniu wstawki nieskończoną liczbę argumentów. Należy przy tym zwrócić uwagę, że trzy kropki umieszczone na końcu zmiennej są elementem składni.

Zmienna `@rest...` jest jedynie nazwą. W rzeczywistości to składnia `...` zapewnia możliwość podawania dowolnej liczby argumentów i sprawia, że niezależnie od tego, ile ich będzie, wstawka zostanie dopasowana. Zapis `@rest...` zapewnia te same możliwości, lecz dodatkowo sprawia, że kompilator zapisze niedopasowane argumenty w zmiennej `@rest`. Zamiast `@rest...` można by zastosować dowolną inną nazwę, na przykład `@odd...`, przy czym w takim przypadku niedopasowane argumenty zostałyby zapisane w zmiennej `@odd`.

Poniższy przykład pokazuje, że w zmiennej `@rest...` zostaną umieszczone wszystkie kolejne argumenty podane za zmienną `@a`, dzięki czemu zostaną one przypisane właściwości `property2`:

```
.mixin(@a,@rest...) {
  property1: @a;
  property 2: @rest;
}
element {
  .mixin(1;2;3;4);
}
```

Ten kod zostanie skompilowany do kodu CSS o poniższej postaci:

```
element {
  property1: 1;
  property2: 2 3 4;
}
```

Można także rozważyć stosowanie zmiennej `@rest...` jako listy wartości rozdzielonych przecinkami. Dzięki temu można przepisać przedstawioną wcześniej wstawkę `.backgroundgradient` w nowej postaci, dostępnej w pliku *less/mixinwithdirectivebackgrounds.less*:

```
.backgroundgradient(@deg: 0; @colors...) {
  background-repeat: repeat-x;
  background-image: linear-gradient(@deg, @colors);
}
```

Teraz w wywołaniu wstawki można podać nieskończoną listę kolorów, jak w poniższym przykładzie:

```
div#content {
  .backgroundgradient(90deg;blue,white,black,pink,purple,yellow,green,orange);
}
```

Wygląd gradientu tworzonego przy użyciu powyższego kodu Lessa został przedstawiony na rysunku 2.4.

Rysunek 2.4. Efekty generowane przez wstawkę .backgroundgradient akceptującą nieograniczoną listę kolorów

Warto zwrócić uwagę, że ta nowa wersja wstawki `.backgroundgradient()` nie używa żadnych prefiksów przeglądarek. Aby ta wstawka mogła być używana w starszych przeglądarkach, trzeba ją skompilować z użyciem wtyczki *autoprefixer*.

Przekazywanie zestawów reguł jako argumentów

Oddzielone zestawy reguł, o których wspomniałem w podrozdziale „Stosowanie zmiennych", mogą być także używane jako argumenty wstawek. Przekazywanie oddzielonych zestawów reguł jako argumentów wstawek pozwala definiować wstawki, które będą umieszczały różne zestawy reguł w tej samej klasie nadrzędnej bądź zapytaniu medialnym. Poniższy kod Lessa przedstawia działanie takiego rozwiązania:

```
.large(@rules) {
  @media (min-width:960px){
    @rules();
  }
}
p {
  font-size: 1em;
  .large( {font-size: 2em;});
}
img.large {
  display:none;
  .large( {display:block; max-width:100%;});
}
```

Jak się okazuje, powyższy kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
p {
  font-size: 1em;
}
@media (min-width: 960px) {
  p {
    font-size: 2em;
  }
}
```

```
img.large {
  display: none;
}
@media (min-width: 960px) {
  img.large {
    display: block;
    max-width: 100%;
  }
}
```

Jak widać w wynikach tego przykładu, Less nie scala zapytań medialnych. Aby grupować zapytania medialne w skompilowanym kodzie CSS, konieczne będzie zainstalowanie wtyczki Lessa *group-css-media-queries* (`npm install less-plugin-group-css-media-queries`) lub wtyczki Pleeease (`npm install less-plugin-pleeease`).

Zwracanie wartości ze wstawek

Osoby przyzwyczajone do programowania funkcyjnego albo choćby znające funkcje matematyczne mogą oczekiwać, że wstawki będą zwracały jakieś wartości wynikowe. Oznacza to mniej więcej tyle, że chcemy przekazać do wstawki jakąś wartość `x`, a ona ma zwrócić wartość `y`. Wstawki Lessa nie zapewniają możliwości zwracania jakichkolwiek wartości, ale podobny efekt można zasymulować, korzystając z zasięgu wstawek. Okazuje się, że zmienna zdefiniowana we wstawce zostanie skopiowana do zasięgu, w którym wstawka została wywołana, o ile tylko w zasięgu tym nie została wcześniej zdefiniowana zmienna o tej samej nazwie. Poniższy przykład wyjaśnia to rozwiązanie:

```
.returnmixin(){
  @par1: 5;
  @par2: 10;
}
.mixin(){
  @par2: 5; // zmienna chroniona przed nadpisaniem
  property1: @par1; // zmienna skopiowana z zasięgu wstawki returnmixin
  property2: @par2;
  .returnmixin();
}
element{
  .mixin();
}
```

Skompilowanie powyższego kodu Lessa wygeneruje następujący kod CSS:

```
element {
  property1: 5;
  property2: 5;
}
```

Po przeanalizowaniu powyższego przykładu widać, że właściwość `@property1: @par1` można by porównać z wywołaniem funkcji, takim jak `property1 = returnmixin();`.

> **Wskazówka**
>
> Stosowanie zasięgu do symulowania zwracania wartości może także być używane do przekazywania wstawek. Otóż wstawka zdefiniowana wewnątrz drugiej wstawki będzie dostępna w zasięgu, w którym ta druga wstawka została wywołana. Niemiej jednak, w odróżnieniu od tego, jak jest w przypadku zmiennych, takie wstawki nie są chronione! Ten proces nazywamy odblokowywaniem, ale jego dokładniejsze wyjaśnienie wykracza poza zakres niniejszej książki.

Modyfikowanie zachowania wstawek

W celu poprawienia elastyczności wstawek przydałaby się możliwość modyfikowania sposobu ich działania na podstawie parametrów wejściowych. Less udostępnia kilka różnych mechanizmów zapewniających te możliwości.

Przełączniki

Wyobraźmy sobie wstawkę `color();`, która w zależności od kontekstu przypisuje właściwości `color` wartość `white` lub `black`. Przypiszmy zatem zmiennej `@context` wartość `light` (`@context: light;`) i zadeklarujmy dwie wstawki o tej samej nazwie, jak w poniższym przykładzie:

```
.color(light)
{
  color: white;
}
.color(dark)
{
  color: black;
}
```

W takim przypadku zastosowanie w kodzie wywołania o postaci `.color(@context)` sprawi, że w zależności od zadeklarowanej wartości zmiennej `@context` właściwości `color` zostanie przypisana wartość `white` albo `black`. Jak na razie, takie rozwiązanie może się wydawać niezbyt użyteczne, ale okaże się przydatne, gdy wielkość projektu zacznie rosnąć. Można się o tym przekonać, analizując kod projektu Bootflat, dostępny na stronie *http://www.flathemes.com/*. Ten projekt udostępnia różne motywy dla frameworku Bootstrap. Sam Bootstrap jest frameworkiem korzystającym z Lessa. Bootflat definiuje dwa style, z których jeden bazuje na poprawionych stylach Bootstrapa 3.0, a w drugim, Square UI, zostały usunięte zaokrąglone wierzchołki. W projekcie Bootflat można zastosować wybrany styl, używając w tym celu jednego przełącznika.

Dopasowywanie argumentów

Less pozwala tworzyć wiele wstawek o tej samej nazwie. Jeśli w kodzie faktycznie pojawi się kilka wstawek o tej samej nazwie, to zostaną użyte wszystkie, które pasują do listy argumentów podanych w wywołaniu. Poniższy przykład przedstawia dwie wstawki `color`:

```
.color(@color)
{
  color: @color;
}
.color(@color1,@color2)
{
  color: gray;
}
```

W takim przypadku wywołanie `.color(white)` zostanie skompilowane do postaci `color: white`, natomiast wywołanie `.color(white,black)`— do postaci `color: gray`. Należy przy tym zwrócić uwagę, że wywołanie `.color(white)` nie pasuje do wstawki `.color(@color1,@color2)`, która wymaga dwóch argumentów, a zatem kompilator jej nie użyje.

Wstawki chronione

Less pozwala także na tworzenie większej liczby wstawek o tej samej nazwie mających tę samą liczbę parametrów. Również w takim przypadku, jak pokazuje poniższy przykład, zostaną użyte wszystkie wstawki pasujące do wywołania:

```
.color(@color){
  color: @color;
  display: block;
}
.color(@color) {
  color: blue;
}
.class{
  .color(white)
}
```

Ten kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
.class{
  color: white;
  display: block;
  color: blue;
}
```

> **Wskazówka**
>
> Zastosowanie dwóch właściwości `color` w powyższym przykładzie nie ma większego sensu. Less nie usuwa powtarzających się deklaracji, chyba że zostaną one użyte dokładnie w taki sam sposób.

Dla uniknięcia problemów ze wstawkami o tej samej nazwie można skorzystać z wartowników wstawek (ang. *mixin guards*). Wartownik jest definiowany przy użyciu słowa kluczowego `when`, po którym w nawiasie należy określić warunek. W przypadku, gdy warunek przyjmie wartość `true`, wstawka zostanie użyta. Działanie tego rozwiązania łatwiej będzie można zrozumieć, analizując poniższy przykład:

```
.mixin(@a) when (@<){
  color: white;
}
.mixin(@a) when (@>=1){
  color: black;
}
.class {
  .mixin(0);
}
.class2 {
  .mixin(1);
}
```

Ten kod Lessa zostanie skompilowany do następującego kodu CSS:

```
.class {
  color: white;
}
.class2 {
  color: black;
}
```

Wartowników można używać podobnie jak instrukcji `if` w popularnych językach programowania. Można w nich stosować następujące operatory porównania: `>`, `>=`, `=`, `=<` oraz `<`. Warunki można ze sobą łączyć, oddzielając je od siebie przecinkami. Tak utworzone wyrażenie zwróci wartość `true`, jeśli dowolny z warunków zwróci wartość `true`.

Dostępne jest także słowo kluczowe `and`, które sprawia, że wyrażenie składające się z dwóch warunków zwróci `true`, jeśli każdy z warunków zwróci `true`, na przykład `@a>1 and @a<5`. I w końcu dostępne jest też słowo kluczowe `not`, które pozwala zmienić wartość warunku na przeciwną, na przykład `when (not a = red)`.

> **Wskazówka**
>
> Osoby, które już wcześniej korzystały z zapytań medialnych CSS, na pewno zauważą, że wartownicy w Lessie działają dokładnie tak samo jak zapytania medialne w kodzie CSS.

Warto także zauważyć, że w warunkach wartowników można też stosować wbudowane funkcje Lessa (opisałem je dokładniej w następnym rozdziale). Mogą one operować na wszystkich zdefiniowanych zmiennych, nawet na tych, które nie zostały podane na liście parametrów wstawki.

Poniższy kod prezentuje przykład zastosowania jednej z wbudowanych funkcji w wartowniku wstawki:

```
@style: light;
.mixin(@color) when iscolor(@color) and (@style = light) {
  color: pink;
}
.class() {
  .mixin(red);
}
```

Ten kod Lessa zostanie skompilowany do kodu CSS o następującej postaci:

```
.class {
  color: pink;
}
```

W razie użycia kodu `@style: dark` lub `.mixin(1);` wstawka `mixin` nie zostanie zastosowana.

Wartowniki CSS

Począwszy do wersji 1.5 Lessa, wartowników można także używać bezpośrednio w selektorach. Poniższy kod przedstawia przykład reguły z wartownikiem:

```
h1 when (@mobile = true) {
  font-size: large;
}
```

Takie rozwiązanie sprawia, że kompilując ten sam kod Lessa, będzie można uzyskać kilka różnych wersji kodu CSS.

Gdyby powyższy kod został zapisany w pliku Lessa o nazwie *source.less*, to można by następnie utworzyć dwa kolejne pliki Lessa i kompilować je, by uzyskać dwa różne arkusze stylów. Poniższy kod przedstawia zawartość obu tych przykładowych plików:

```
mobile.less:
@import "source";
@mobile: true;

default.less:
@import "source";
@mobile: false;
```

Zamiast tworzyć odrębne pliki można także zastosować opcję kompilatora `--modify-var` i użyć jej, by wygenerować różne arkusze stylów CSS. Opcja `--modify-var` pozwala zmodyfikować wartość zmiennej zdefiniowanej w kodzie. Jak już wspomniałem, Less korzysta z leniwego wczytywania, dzięki czemu istnieje możliwość zmiany wartości zmiennej poprzez umieszczenie jej definicji na samym końcu kodu.

W ostatnim przykładzie, korzystając z kompilatora działającego z poziomu wiersza poleceń, mobilną wersję arkusza stylów można wygenerować za pomocą następującego polecenia:

```
lessc --modify-var="mobile=true" source.less
```

W przypadku stosowania kompilatora działającego po stronie klienta należy użyć opcji `modify ↪Vars`. Poniższy fragment kodu HTML pokazuje, w jaki sposób można użyć tej opcji, by przypisać zmiennej `mobile` wartość `true`:

```
<link data-modify-vars='{ mobile: "true"}' rel="stylesheet/less"
  type="text/css" href="less/styles.less">
```

Stosowanie wartowników i dopasowywania argumentów do tworzenia pętli

Kiedy Less nie jest w stanie odnaleźć pasującej wstawki, nie zatrzymuje się, lecz zaczyna przetwarzać dalszą część kodu. Ten sposób działania kompilatora można wykorzystać, w połączeniu z wartownikami i dopasowywaniem argumentów, do tworzenia pętli. Aby zrozumieć to rozwiązanie, wyobraź sobie dziesięć klas, z których każda zawiera obraz tła oznaczony odpowiednim numerem. Klasa `.class1` definiuje właściwość `background-image` o wartości `background-1.png`, klasa `.class2` definiuje właściwość `background-image` o wartości `background-2.png` i tak dalej. Takie klasy można wygenerować, używając następującego kodu Lessa:

```
.setbackground(@number) when (@number>0) {
  .setbackground( @number - 1 );
  .class@{number} { background-image: ~"url(backgroundimage-@{number}.png)"; }
}
.setbackground(10);
```

Po skompilowaniu wygeneruje on następujący kod CSS:

```
.class1 {
  background-image: url(backgroundimage-1.png);
}
.class2 {
  background-image: url(backgroundimage-2.png);
}
...
.class10 {
  background-image: url(backgroundimage-10.png);
}
```

Na pierwszy rzut oka ta wstawka może się wydawać złożona, gdyby jednak spróbować samodzielnie ją przetworzyć, to okazałoby się, że zawiera wiele elementów, które już poznaliśmy.

Przedstawiona w powyższym przykładzie wstawka `setbackground` wywołuje sama siebie. Programiści nazywają taką technikę rekurencją. A jak działa ta wstawka?

Wywołanie `.setbackground(10);` pasuje do wstawki `.setbackground(@number)`, gdyż warunek `@number>0` jest spełniony, a zatem można użyć wstawki. Także pierwsza próba przetworzenia wywołania `.setbackground(@number - 1);` zakończy się pomyślnym odnalezieniem pasującej

wstawki. Oznacza to, że kompilator ponownie wykona wstawkę. Ten proces będzie się powtarzał aż do momentu, gdy wyrażenie `@number - 1` przyjmie wartość 0, ponieważ w tym przypadku żadna pasująca wstawka nie zostanie odnaleziona. Teraz kompilator cofnie się z powrotem do miejsca, w którym się zatrzymał, by wykonać tę wstawkę.

Tym ostatnim miejscem było wywołanie wstawki, w którym zmienna `@number` była równa 1, a więc kompilator przetworzy deklarację `.class@{number} { background-image: ~"url(backgroundimage-↪@{number}.png)"; }`, podstawiając w miejsce zmiennej `@number` wartość 1. Jeszcze wcześniejsze miejsce, w którym kompilator się zatrzymał, odpowiadało wywołaniu wstawki, w którym zmienna `@number` miała wartość 2. A zatem kompilator przetworzy deklarację `.class@{number} { background-image: ~"url(backgroundimage-@{number}.png)"; }`, podstawiając w miejsce zmiennej `@number` wartość 2. W ten sposób kompilator wróci aż do miejsca, w którym zmienna `@number` miała wartość 10, a wtedy cały kod będzie już skompilowany. Kompilator będzie więc mógł zakończyć działanie.

Oprócz wartowników i dopasowywania argumentów powyższy kod zawiera także przykład wstawiania zmiennej do nazwy właściwości, w deklaracji klasy `@class@{number}`, oraz przetwarzania łańcuchów znaków, we fragmencie `~"url(backgroundimage-@{number}.png)";`. Ta wstawka pokazała również konieczność stosowania dodatkowych znaków odstępu podczas wykonywania działań arytmetycznych. Dzięki nim kod `@number - 1` nie zostanie potraktowany jak nazwa zmiennej `@number-1`.

Stosowanie wstawek do przetwarzania w pętli zbioru wartości

W niektórych sytuacjach istnieje możliwość zdefiniowania wartości w formie zestawu wstawek i używania tej struktury do tworzenia powtarzającego się kodu. Przeanalizujmy następujący kod Lessa:

```
.widths() {
  .set("small",100px);
  .set("medium",200px);
  .set("large",400px);
}
div {
  .widths();
  .set(@name,@width) {
    @classname: ~"@{name}";
    &.@{classname} {
      width: @width;
    }
  }
}
```

W wyniku jego kompilacji zostanie wygenerowany następujący kod CSS:

```
div.small {
  width: 100px;
}
div.medium {
```

```
  width: 200px;
}
div.large {
  width: 400px;
}
```

Zrozumienie powyższego kodu Lessa wcale nie jest łatwym zadaniem. Wstawka `.widths()` zawiera serię wywołań wstawki `.set()` i podobnie jak wiele innych wstawek nie będzie bezpośrednio generowała wyników podczas kompilacji. Wywołanie wstawki `.widths()` umieszczone wewnątrz selektora `div` udostępnia w bieżącym zasięgu wstawkę `.set()`. Następnie wstawka `.set()` wywołuje w tym samym zasięgu wstawkę `.set(@name,@width)`, a w każdym z tych wywołań używane są przekazane do niego dane. Symbol `&` zastosowany we wstawce `.set()` został opisany w następnym rozdziale.

Alternatywnym sposobem operowania na tablicy wartości jest zastosowanie biblioteki wstawek `for`, dostępnej na stronie *https://github.com/seven-phases-max/less.curious/blob/master/articles/generic-for.md*.

Korzystając ze struktury wstawki `for`, powyższy kod można zapisać w następującej postaci:

```
@data: "small" 100px, "medium" 200px, "large" 400px;
div {
  .for(@data); .-each(@width) {
    @classname: e(extract(@width,1));
    &.@{classname} {
      width: extract(@width,2);
    }
  }
}
```

Powyższy kod Lessa korzysta z dwóch funkcji wbudowanych, `e()` oraz `extract()`, które opisałem dokładniej w następnym rozdziale.

Słowo kluczowe !important

Ostatnim zagadnieniem przedstawionym w tym rozdziale będzie słowo kluczowe `!important`. Zastosowanie tego słowa w deklaracji daje jej najwyższy priorytet w sytuacjach, gdy do jednego elementu pasuje kilka selektorów. Słowo kluczowe `!important` ma wyższy priorytet nawet od stylów podanych bezpośrednio w kodzie HTML elementów, co pokazuje poniższy przykład:

```
<style>
  p{color:green !important;}
</style>
<p style="color:red;">zielony</p>
```

W przypadku użycia powyższego kod przeglądarka wyświetli tekst w kolorze zielonym. Jak pokazuje ten przykład, słowa kluczowego `!important` można używać, aby zmieniać style, któ-

rych nie można edytować, które, na przykład, zostały podane bezpośrednio w kodzie HTML. Niemniej jednak tego słowa kluczowego należy używać ostrożnie, gdyż jedynym sposobem przesłonięcia reguły, która z niego korzysta, jest napisanie innej reguły, w której także zostanie ono użyte. Każde nieprawidłowe lub niepotrzebne zastosowanie słowa `!important` w kodzie Lessa utrudni jego zrozumienie i utrzymanie.

W kodzie Lessa słowa kluczowego `!important` można używać nie tylko w odniesieniu do właściwości, lecz także do całych wstawek. Jeśli `!important` zostanie użyte w wywołaniu wstawki, to kompilator umieści je również we wszystkich właściwościach zadeklarowanych w danej wstawce. Ten sposób użycia słowa kluczowego `!important` przedstawia następujący przykład:

```
.mixin(){
  property1: 1;property2: 2;
}
.class{
  .mixin() !important;
}
```

Powyższy kod Lessa zostanie skompilowany w następujący sposób:

```
.class{
  property1: 1 !important;
  property2: 2 !important;
}
```

Słowa kluczowego `!important` można także używać w zmiennych:

```
@color: red !important;
div {
  color: @color;
}
```

Skompilowanie tego kodu spowoduje wygenerowanie następującego kodu CSS:

```
div {
  color: red !important;
}
```

Podsumowanie

W tym rozdziale przedstawiłem zmienne i wstawki. Dowidziałeś się w niego także, dlaczego definiowanie zmiennych i wstawek w jednym miejscu pozwala zmniejszyć wielkość kodu i uprościć jego utrzymanie.

Z następnego rozdziału dowiesz się jeszcze więcej o wstawkach, sposobach ich zagnieżdżania i rozszerzania. Oprócz tego przedstawię wbudowane funkcje Lessa, których można używać do manipulowania wartościami we wstawkach i w innych miejscach kodu.

Skorowidz

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

Z

Less
Podstawy programowania

Less (Leaner CSS) jest preprocesorem CSS, dzięki któremu projektant witryny może tworzyć łatwy w utrzymaniu kod, nadający się do wielokrotnego użycia bez powielania tych samych jego fragmentów. Less doskonale realizuje zasadę DRY (ang. Do not Repeat Yourself — „nie powtarzaj się"). Rozszerza składnię CSS o zmienne, wstawki, funkcje i udostępnia wiele sposobów na poprawę wydajności pracy projektanta. Ułatwia tworzenie atrakcyjnych witryn, a utrzymywanie już istniejących aplikacji jest dzięki niemu mniej czasochłonne.

Dzięki tej książce zrozumiesz filozofię działania Less, a co za tym idzie, zaczniesz pisać kod przejrzysty, czytelny, zwięzły i możliwy do wielokrotnego wykorzystania. Szybko zauważysz poprawę wydajności swojej pracy. Znajdziesz tu praktyczne porady dotyczące integracji Less z istniejącymi czy też nowymi projektami. Oszczędzisz dzięki temu sporo czasu. Zapewne docenisz to, że w książce przedstawiono także sposoby pisania i wykorzystania wtyczek Less, które ułatwiają sprostanie nawet bardzo specyficznym wymaganiom.

Jeśli korzystasz z CSS — przekonaj się, jak świetnym narzędziem jest Less!

Dzięki tej książce poznasz:

- zaawansowane możliwości CSS
- zastosowanie zmiennych, wstawek, reguł zagnieżdżonych i wbudowanych funkcji Less
- zasady korzystania z frameworka Bootstrap 3 za pomocą preprocesora Less
- integrację Less z WordPressem

Autorem książki jest doświadczony projektant **Bass Jobsen**, który tworzy strony WWW od 1995 roku i używał przy tym chyba wszystkich możliwych języków programowania. Zawsze wypracowuje jak najbardziej funkcjonalny interfejs użytkownika. Właściwie codziennie korzysta z Less, także przy tworzeniu motywów WordPressa, a przede wszystkim w pracy nad wspaniałym projektem StreetArt.nl. Jest znany z gotowości do dzielenia się swoją wiedzą z mniej doświadczonymi kolegami.

[PACKT] open source PUBLISHING community experience distilled

Helion

39066 numer katalogowy

księgarnia internetowa
http://helion.pl

zamówienia telefoniczne
0 801 339900
0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

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

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-283-1754-3

cena: 49,00 zł