Keith J. Grant

CSS

OD PODSZEWKI

Helion

Tytuł oryginału: CSS in Depth

Tłumaczenie: Piotr Pilch

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

ISBN: 978-83-283-4446-4

Helion SA
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)

Dodatkowe materiały do książki można znaleźć pod adresem: *ftp://ftp.helion.pl/przyklady/cssodp.zip*

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

Printed in Poland.

Spis treści

1 Kaskada, specyficzność i dziedziczenie

W rozdziale omówiono następujące zagadnienia:

- Cztery części tworzące kaskadę.
- Różnica między kaskadą a dziedziczeniem.
- Kontrolowanie tego, jakie style są stosowane dla jakich elementów.
- Powszechne nieporozumienia związane z deklaracjami skróconymi.

Kaskadowe arkusze stylów CSS (ang. *Cascading Style Sheets*) nie przypominają wielu rzeczy ze świata projektowania oprogramowania. Mówiąc wprost, nie jest to język programowania, ale wymaga myślenia abstrakcyjnego. Nie jest to klasyczne narzędzie do projektowania, ale wymaga trochę kreatywności. CSS zapewnia pozornie prostą składnię deklaratywną, lecz jeśli miałeś z nią do czynienia w dowolnych dużych projektach, to wiesz, że może osiągnąć utrudniającą pracę złożoność.

Gdy musisz dowiedzieć się, jak coś zrobić w wypadku tradycyjnego języka programowania, zwykle możesz określić, czego szukać (formułując na przykład pytanie „Jak znajdę w tablicy elementy typu *x*?"). W CSS zawężenie problemu do postaci pojedynczego pytania nie zawsze jest proste. Nawet wtedy gdy to się uda, odpowiedź na pytanie często brzmi: „To zależy". Najlepszym sposobem na uzyskanie czegoś jest często uzależnienie tego od konkretnych ograniczeń, a także od tego, jak dokładnie mają być obsługiwane różne skrajne przypadki.

Choć pomocna jest znajomość kilku „sztuczek" lub przydatnych procedur, z których możesz skorzystać, opanowanie CSS wymaga zrozumienia zasad umożliwiających takie praktyki. Książka ta jest pełna przykładów, ale przede wszystkim prezentuje zasady.

W części pierwszej omówiono najpierw najbardziej fundamentalne zasady języka: kaskadę, model pudełkowy oraz szeroką gamę dostępnych typów jednostek. Większości projektantów witryn internetowych znane są kaskada i model pudełkowy. Nie są im obce jednostki pikselowe, a być może słyszeli o tym, że „zamiast nich powinni używać jednostki em". Prawdę powiedziawszy, z zagadnieniami tymi związanych jest mnóstwo informacji, a pobieżne zaznajomienie się z nimi spowoduje, że nie zrobisz większych postępów. Jeśli w ogóle zamierzasz opanować CSS, musisz najpierw poznać podstawy, i to dogłębnie.

Wiem, że jesteś podekscytowany faktem rozpoczęcia poznawania tego, co CSS ma najnowszego i najlepszego do zaoferowania. Jest to ekscytująca sprawa. Najpierw jednak powrócimy do podstaw. Dokonam ich szybkiego przeglądu (prawdopodobnie jesteś już z nimi zaznajomiony), a następnie zagłębię się w każde zagadnienie. Moim celem jest wzmocnienie fundamentu, na którym spoczywa reszta elementów języka CSS.

W rozdziale zaczniemy od litery *C* w skrócie *CSS*, która to litera reprezentuje kaskadę. Dokładnie wyjaśnię, jakie jest jej działanie, po czym przybliżę, w jaki sposób wykorzystać ją praktycznie. Dalej zajmiemy się powiązanym z kaskadą zagadnieniem, czyli dziedziczeniem. Po wyjaśnieniu tego zagadnienia przejdziemy do właściwości skróconych oraz związanych z nimi powszechnych nieporozumień.

Wszystkie te zagadnienia dotyczą stosowania żądanych stylów dla wybranych elementów. Istnieje wiele „pułapek", w które często wpadają projektanci. Właściwe zrozumienie tych pojęć da lepszą kontrolę nad tym, aby kod CSS działał zgodnie z naszymi wymaganiami. Przy odrobinie szczęścia bardziej docenisz też fakt korzystania z CSS, a nawet będzie to sprawiać Ci satysfakcję.

1.1. Kaskada

Zasadniczo CSS dotyczy deklarowania reguł: w różnych okolicznościach oczekujemy wystąpienia określonych zdarzeń. Jeśli dana klasa zostanie dodana do danego elementu, mają zostać zastosowane dane style. Jeśli element *X* jest elementem podrzędnym elementu *Y*, mają zostać użyte dane style. Przeglądarka korzysta następnie z tych reguł, określa miejsca, w których mają zastosowanie konkretne reguły, a także używa ich do renderowania strony.

Gdy przyjrzysz się niewielkim przykładom, proces ten okaże się raczej prosty. Jeśli jednak utworzony arkusz stylów powiększy się albo zwiększy się liczba stron, dla których jest on stosowany, zaskakująco szybko kod może stać się złożony. W CSS dostępnych jest często kilka sposobów osiągnięcia tego samego. Zależnie od tego, jakie rozwiązanie zastosujesz, możesz uzyskać zupełnie odmienne wyniki, gdy zmienia się struktura stron HTML lub gdy style są stosowane względem różnych stron. Kluczowa część procesu projektowania z użyciem CSS sprowadza się do tworzenia reguł w taki sposób, aby były przewidywalne.

Pierwszym krokiem do tego jest zrozumienie, jak dokładnie przeglądarka korzysta z reguł. Każda reguła sama w sobie może być jasna, ale co się dzieje, gdy dwie reguły powodują konflikt informacji dotyczących sposobu użycia stylu dla elementu? Możesz stwierdzić, że jedna z reguł nie działa zgodnie z oczekiwaniami, ponieważ kolejna reguła powoduje z nią konflikt. By móc przewidywać sposób zachowania reguł, trzeba zrozumieć kaskadę.

Aby to zilustrować, zostanie zbudowany prosty nagłówek strony podobny do tego, jaki możesz ujrzeć u góry przedstawionej strony internetowej (rysunek 1.1). Nagłówek zawiera tytuł strony umiejscowiony ponad zestawem turkusowych odnośników nawigacji. Ostatni odnośnik ma kolor pomarańczowy, żeby móc go wyróżnić jako coś w rodzaju specjalnego odnośnika.

Rysunek 1.1. Nagłówek strony i odnośniki nawigacji

W trakcie tworzenia tego nagłówka strony prawdopodobnie stwierdzisz, że jesteś zaznajomiony z większością użytego kodu CSS. Dzięki temu możliwe będzie skoncentrowanie się na aspektach arkuszy stylów CSS, które możesz uważać za oczywiste lub uznawać za tylko częściowo zrozumiałe.

Aby zacząć, utwórz dokument HTML i arkusz stylów o nazwie *styles.css*. Do dokumentu dodaj kod z listingu 1.1.

Listing 1.1. Kod znaczników nagłówka strony

```
<!doctype html>
<head>
  <link href="styles.css" rel="stylesheet" type="text/css" />
</head>
<body>
  <header class="page-header">
    <h1 id="page-title" class="title">Wypalacze kawy Wombat</h1> <-- Tytuł strony.
<nav>
      <ul id="main-nav" class="nav"> <-- Lista odnośników nawigacji.
<li><a href="/">Strona główna</a></li>
        <li><a href="/coffees">Kawy</a></li>
        <li><a href="/brewers">Kawiarki</a></li>
        <li><a href="/specials" class="featured">Promocje</a></li> <-- Wyróżniony odnośnik.
      </ul>
    </nav>
  </header>
</body>
```

UWAGA Repozytorium zawierające wszystkie kody z listingów zamieszczonych w książce jest dostępne do pobrania pod adresem *ftp://ftp.helion.pl/przyklady/cssodp.zip*. Repozytorium to uwzględnia cały osadzony kod CSS wraz z powiązanym kodem HTML znajdującym się w zestawie plików HTML.

Gdy dwie lub większa liczba reguł dotyczy tego samego elementu na stronie, mogą one powodować wystąpienie deklaracji wywołujących konflikt. Taką sytuację prezentuje listing 1.2. Zawiera on trzy zestawy reguł, z których każdy określa inny styl czcionki

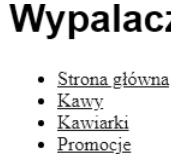
dla tytułu strony. W tytule nie można jednocześnie używać trzech różnych czcionek. Która z nich zostanie wybrana? Aby się o tym przekonać, do pliku CSS dodaj kod z poniższego listingu.

Listing 1.2. Deklaracje wywołujące konflikt

```
h1 {                    <-- Selektor znacznika (lub typu).
font-family: serif;
}
#page-title {           <-- Selektor identyfikatora.
  font-family: sans-serif;
}
.title {                <-- Selektor klasy.
  font-family: monospace;
}
```

Zestawy reguł z deklaracjami wywołującymi konflikt mogą występować jeden po drugim albo mogą być rozrzucone w obrębie arkusza stylów. W każdym wariancie z punktu widzenia użytego kodu HTML wszystkie dotyczą tego samego elementu.

Wszystkie trzy zestawy reguł próbują ustawić dla nagłówka inną rodzinę czcionek. Która z nich zwycięży? Aby uzyskać odpowiedź, przeglądarka postępuje zgodnie z zestawem reguł, dlatego wynik jest przewidywalny. W tym wypadku reguły dyktują, że zwycięzcą jest druga deklaracja z selektorem identyfikatora. Tytuł uzyska czcionkę sans-serif (rysunek 1.2).



Rysunek 1.2. Selektor identyfikatora zwycięża z innymi zestawami reguł, zapewniając dla tytułu czcionkę sans-serif

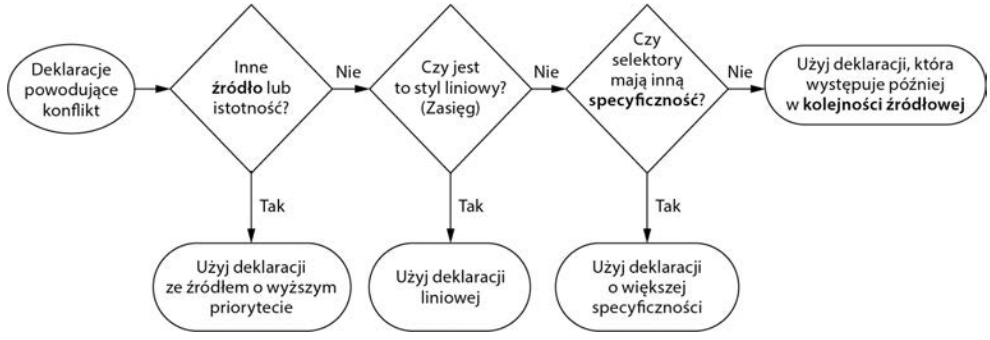
Kaskada to nazwa reprezentująca taki zestaw reguł. Określa ona sposób rozwiązywania konfliktów, a ponadto stanowi fundamentalną część decydującą o działaniu arkuszy stylów CSS. Choć najbardziej doświadczeni projektanci mają ogólną orientację w kwestii kaskady, jej elementy są czasami źle rozumiane.

Rozłóżmy kaskadę. W momencie wystąpienia konfliktu między deklaracjami w celu określenia różnicy między nimi kaskada bierze pod uwagę następujące trzy kwestie:

1. *Pochodzenie arkusza stylów*. Miejsce, z którego pochodzą style. Użyte style są stosowane w połączeniu ze stylami domyślnymi przeglądarki.
2. *Specyficzność selektorów*. Określanie, jakie selektory mają pierwszeństwo względem innych.
3. *Kolejność źródłowa*. Kolejność, w jakiej style są deklarowane w arkuszu stylów.

Reguły kaskady są uwzględniane w powyższej kolejności. Na rysunku 1.3 pokazano, jak są one stosowane na wyższym poziomie.

Reguły te umożliwiają przeglądarkom przewidywalne działanie przy rozwiązywaniu dowolnej niejednoznaczności w kodzie CSS. Omówmy po kolei każdą z reguł.



Rysunek 1.3. Ogólny diagram kaskady prezentujący pierwszeństwo deklaracji

Szybki przegląd terminologii

Zależnie od tego, gdzie poznawałeś CSS, możesz nie być zaznajomiony z wszystkimi nazwami różnych elementów składni CSS. Nie będę rozwijać tej kwestii, ale ponieważ terminów tych będę używał w książce, najlepiej, aby ich znaczenie było oczywiste.

Poniżej podano wiersz kodu CSS. Nosi on nazwę **deklaracji**. Składa się ona z **właściwości** (`color`) i **wartości** (`black`):

```
color: black;
```

Właściwości nie należy mylić z atrybutami, które są częścią składni języka HTML. Na przykład w elemencie `<a href="/">` `href` jest to atrybut znacznika `a`.

Grupa deklaracji wewnątrz nawiasów klamrowych nosi nazwę **bloku deklaracji**. Blok ten jest poprzedzony **selektorem** (w tym wypadku `body`):

```
body {
  color: black;
  font-family: Helvetica;
}
```

Selektor i blok deklaracji są razem określane mianem **zestawu reguł**.

I wreszcie, **specyficzne reguły** (ang. *at-rules*) to konstrukcje językowe rozpoczynające się od symbolu `@`, takie jak reguły `@import` lub zapytania `@media`.

1.1.1. Źródło arkusza stylów

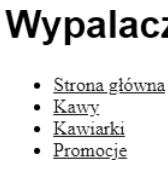
Arkusze stylów, które dodajesz do strony internetowej, nie są jedynymi, jakie stosuje przeglądarka. Istnieją różne typy lub źródła arkuszy stylów. Twoje style są nazywane stylami **autora**. Dostępne są też style agenta użytkownika, które są domyślnymi stylami przeglądarki. Style te mają niższy priorytet, dlatego zostaną przesłonięte przez użyte style.

UWAGA Niektóre przeglądarki pozwalają użytkownikom zdefiniować **arkusz stylów użytkownika**. Jest to traktowane jako trzecie źródło, z priorytetem umiejscowionym między priorytetami stylów agenta użytkownika i stylów autora. Style użytkownika są rzadko stosowane i nie masz nad nimi kontroli, dlatego dla uproszczenia z nich zrezygnowałem.

Style agenta użytkownika różnią się nieznacznie w poszczególnych przeglądarkach, ale zasadniczo realizują to samo: nagłówki (od `<h1>` do `<h6>`) i akapity (`<p>`) uzyskują górny i dolny margines, listy (`<ol>` i `<ul>`) otrzymują lewe wypełnienie, a ponadto ustawiane są kolory odnośników i domyślne rozmiary czcionek.

STYLE AGENTA UŻYTKOWNIKA

Przyjrzyjmy się ponownie przykładowej stronie (rysunek 1.4). Czcionka tytułu to sans-serif z powodu dodanych stylów. Style agenta użytkownika określają kilka innych rzeczy: lista ma lewe wypełnienie, a także właściwość `list-style-type` o wartości `disc` w celu utworzenia punktorów. Odnośniki są niebieskie i podkreślone. Nagłówek i lista mają górny i dolny margines.



Rysunek 1.4. Style agenta użytkownika ustawione domyślnie dla nagłówka strony internetowej

Po uwzględnieniu stylów agenta użytkownika przeglądarka stosuje Twoje style, czyli style autora. Umożliwia to deklaracjom określonym przez Ciebie przesłonięcie tych ustawionych przez arkusz stylów agenta użytkownika. Jeśli w kodzie HTML dodasz odnośnik do kilku arkuszy stylów, wszystkie one będą mieć to samo źródło, czyli autora.

Style agenta użytkownika ustawiają to, czego zwykle wymagasz, dlatego nie wykonają niczego zupełnie nieoczekiwanego. Jeśli nie spodoba Ci się to, co zrobią z określoną właściwością, możesz ustawić dla niej własną wartość w arkuszu stylów. Zróbmy to teraz. Możesz przesłonić wybrane style agenta użytkownika, które nie są Ci potrzebne. W efekcie strona będzie wyglądać jak na rysunku 1.5.

Rysunek 1.5. Style autora przesłaniają style agenta użytkownika, ponieważ mają wyższy priorytet

W listingu 1.3 usunąłem powodujące konflikt deklaracje rodziny czcionek z wcześniejszego przykładu i dodałem nowe deklaracje w celu ustawienia kolorów, a także przesłonięcia marginesów agenta użytkownika oraz wypełnienia listy i punktorów. Zmodyfikuj arkusz stylów tak, aby uwzględnić w nim te zmiany.

Listing 1.3. Przesłanianie stylów agenta użytkownika

```
h1 {
  color: #2f4f4f;
  margin-bottom: 10px;
}
#main-nav {
  margin-top: 10px;
  list-style: none;
  padding-left: 0;
}
#main-nav li {
  display: inline-block;
}
#main-nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
```

Ogranicza marginesy.

Usuwa style listy agenta użytkownika.

Sprawia, że pozycje listy pojawiają się obok siebie, a nie jedna na drugiej.

Zapewnia odnośnikom nawigacji wygląd przycisków.

```
  border-radius: 2px;
  text-decoration: none;
}
```

Zapewnia odnośnikom nawigacji wygląd przycisków.

Jeśli od dawna korzystasz z CSS, prawdopodobnie przesłaniałeś style agenta użytkownika. Podczas wykonywania tego działania używana jest źródłowa część kaskady. Twoje style zawsze będą przesłaniać style agenta użytkownika, ponieważ źródła są różne.

UWAGA Możesz zauważyć, że w powyższym kodzie użyłem selektorów identyfikatora. Są powody, dla których należy tego unikać. Później poświęcę temu trochę miejsca.

WAŻNE DEKLARACJE

W wypadku reguł źródła stylów występuje następujący wyjątek: deklaracje oznaczone jako **ważne**. Deklaracja może zostać oznaczona jako ważna przez dodanie na jej końcu, ale przed średnikiem, adnotacji `!important`:

```
color: red !important;
```

Deklaracje oznaczone adnotacją `!important` są traktowane jako źródło o wyższym priorytecie, dlatego ogólna kolejność preferencji źródeł w porządku malejącym jest następująca:

1. Ważny autor.
2. Autor.
3. Agent użytkownika.

Kaskada niezależnie rozwiązuje konflikty dla każdej właściwości każdego elementu strony. Jeśli na przykład dla akapitu ustawisz pogrubioną czcionkę, w dalszym ciągu będą obowiązywać marginesy górny i dolny z arkusza stylów agenta użytkownika (chyba że jawnie dokonasz ich przesłonięcia). Pojęcie źródła stylów zaznaczy swoją obecność przy omawianiu przejść i animacji, ponieważ operacje te wprowadzają do powyższej listy więcej źródeł. Adnotacja `!important` to interesująca ciekawostka związana z CSS, do której wkrótce powrócimy.

1.1.2. Specyficzność

Jeśli nie jest możliwe rozwiązanie konfliktu między deklaracjami na podstawie ich źródła, następnym krokiem podejmowanym przez przeglądarkę jest zapewnienie rozstrzygnięcia przez sprawdzenie **specyficzności** deklaracji. Kluczowe jest zrozumienie tego zagadnienia. Przez długi czas możesz działać, nie rozumiejąc źródła arkuszy stylów, gdyż 99% stylów używanych w witrynie internetowej pochodzi z tego samego źródła. Jeśli jednak nie zrozumiesz specyficzności, ona się odpłaci. Niestety pojęcie to jest często pomijane.

Przeglądarka określa specyficzność w ramach dwóch części: w stylach stosowanych liniowo w kodzie HTML oraz w stylach używanych za pomocą selektora.

STYLE LINIOWE

Jeśli w celu zastosowania stylów korzystasz z atrybutu `style` języka HTML, deklaracje są uwzględniane tylko względem danego elementu. W rezultacie są to deklaracje „zasięgowe”, które przesłaniają wszystkie deklaracje zastosowane z utworzonego arkusza

stylów lub za pomocą znacznika `<style>`. Style liniowe nie mają selektora, ponieważ są stosowane bezpośrednio dla elementu, do którego się odwołują.

Zależy Ci, aby na stronie wyróżniony odnośnik *Promocje* w menu nawigacji miał kolor pomarańczowy (rysunek 1.6). Omówię kilka sposobów umożliwiających osiągnięcie tego. Zacznę od stylów liniowych z listingu 1.4.

Rysunek 1.6. Zastosowanie stylów liniowych powoduje przesłonięcie stylów użytych za pomocą selektorów

Listing 1.4. Style liniowe przesłaniające deklaracje zastosowane gdzie indziej

```
<li>
  <a href="/specials" class="featured"
    style="background-color: orange;"> <-- Styl liniowy zastosowany za pomocą atrybutu style.
    Promocje
  </a>
</li>
```

Aby ujrzeć stronę w przeglądarce, zmodyfikuj jej kod w celu uwzględnienia kodu z powyższego listingu (wkrótce dokonane zmiany zostaną wycofane).

Aby w arkuszu stylów przesłonić deklaracje liniowe, konieczne będzie dodanie do deklaracji adnotacji `!important`, co spowoduje, że stanie się ona źródłem o wyższym priorytecie. Jeśli style liniowe oznaczono jako ważne, nic nie może ich przesłonić. Preferowane jest wykonanie takiego działania w obrębie utworzonego arkusza stylów. Cofnij dokonaną zmianę, gdyż przyjrzymy się lepszym rozwiązaniom.

SPECYFICZNOŚĆ SELEKTORA

Druga część specyficzności określana jest przez selektory. Na przykład selektor z dwiema nazwami klasy ma większą specyficzność niż selektor z tylko jedną nazwą klasy. Jeśli jedna deklaracja ustawi kolor tła na pomarańczowy, lecz druga deklaracja, z większą specyficznością, ustawi dla tła kolor turkusowy, przeglądarka użyje tego drugiego koloru.

Aby to zilustrować, sprawdźmy, co się stanie, gdy spróbujemy ustawić dla wyróżnionego odnośnika kolor pomarańczowy za pomocą zwykłego selektora klasy. Zaktualizuj ostatnią część arkusza stylów tak, aby uwzględniał kod z listingu 1.5.

Listing 1.5. Selektory z różnymi specyficznościami

```
#main-nav a {  <-- Selektor z większą specyficznością.
  color: white;
  background-color: #13a4a4;  <-- Turkusowy kolor tła.
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}
.featured {
  background-color: orange;
}
```

Deklaracja pomarańczowego tła nie przesłoni koloru turkusowego z powodu specyficzności selektora.

To nie działa! Wszystkie odnośniki nadal mają kolor turkusowy. Dlaczego? Pierwszy użyty tutaj selektor jest bardziej specyficzny niż drugi. Pierwszy selektor składa się z identyfikatora i nazwy znacznika, natomiast drugi selektor tworzy nazwa klasy. W tym jednak wypadku chodzi o coś więcej niż tylko o stwierdzenie, który selektor jest dłuższy.

Różne typy selektorów też mają odmienne specyficzności. Selektor identyfikatora ma specyficzność większą niż na przykład selektor klasy. Okazuje się, że pojedynczy identyfikator ma większą specyficzność niż selektor z dowolną liczbą klas. Podobnie specyficzność selektora klasy jest większa niż selektora znacznika (nazywanego również **selektorem typu**).

Dokładne reguły specyficzności są następujące:

- Jeśli selektor ma więcej identyfikatorów, zwycięża, czyli jest bardziej specyficzny.
- W razie remisu wygrywa selektor z największą liczbą klas.
- W razie remisu wygrywa selektor z największą liczbą nazw znaczników.

Przeanalizuj selektory widoczne w listingu 1.6 (nie dodawaj ich jednak do kodu własnej strony). Selektory zapisano w kolejności zwiększającej się specyficzności.

Listing 1.6. Selektory ze zwiększającą się specyficznością

```
html body header h1 {  ❶ Cztery znaczniki.
color: blue;
}
body header.page-header h1 {  ❷ Trzy znaczniki i jedna klasa.
  color: orange;
}
.page-header .title {  ❸ Dwie klasy.
  color: green;
}
#page-title {  ❹ Jeden identyfikator.
  color: red;
}
```

W tym wypadku najbardziej specyficzny selektor z jednym identyfikatorem oznaczono liczbą ❹, dlatego jego deklaracja koloru czerwonego jest stosowana dla tytułu. Następny pod względem specyficzności selektor z dwiema nazwami klasy ma liczbę ❸. Selektor ten zostałby użyty, gdyby brakowało selektora identyfikatora ❹. Selektor ❸ ma większą specyficzność niż selektor ❷ pomimo jego długości: dwie klasy są bardziej specyficzne niż jedna klasa. I wreszcie, selektor ❶ jest najmniej specyficzny. Zawiera cztery typy elementu (czyli nazwy znacznika), lecz nie ma żadnych identyfikatorów lub klas.

UWAGA Selektory pseudoklas (np. `:hover`) i selektory atrybutów (np. `[type="input"]`) mają taką samą specyficzność jak selektor klas. Selektor uniwersalny (*) i kombinatory (>, +, ~) nie mają żadnego wpływu na specyficzność.

Jeśli wydaje się, że deklaracja dodana przez Ciebie do kodu CSS nie powoduje żadnego efektu, często jest to wynikiem przesłonięcia jej przez bardziej specyficzną regułę. Wielokrotnie projektanci tworzą selektory za pomocą identyfikatorów, nie zdając sobie

sprawy z tego, że powoduje to powstanie większej specyficzności, którą później trudno przesłonić. Jeśli wymagasz przesłonięcia stylu zastosowanego przy użyciu identyfikatora, musisz skorzystać z innego identyfikatora.

Jest to proste zagadnienie, ale jeśli nie zrozumiesz specyficzności, możesz doprowadzić się do frustracji, próbując stwierdzić, dlaczego jedna reguła działa, a inna nie.

NOTACJA ZWIĄZANA ZE SPECYFICZNOŚCIĄ

Typowym sposobem wskazania specyficzności jest zastosowanie postaci liczbowej, w wypadku której często po każdej liczbie wstawiany jest przecinek. Na przykład zapis 1,2,2 wskazuje specyficzność jednego identyfikatora, dwóch klas i dwóch znaczników. Identyfikatory z najwyższym priorytetem są wyszczególniane jako pierwsze. Po nich następują klasy, a później znaczniki.

Selektor `#page-header #page-title` ma dwa identyfikatory i nie ma żadnych klas ani znaczników. Można stwierdzić, że selektor ten ma specyficzność 2,0,0. Selektor `ul li` z dwoma znacznikami, lecz bez żadnych identyfikatorów i klas, ma specyficzność 0,0,2. W tabeli 1.1 pokazano selektory z listingu 1.6.

Tabela 1.1. Różne selektory oraz odpowiadające im specyficzności

Selektor	Identyfikatory	Klasy	Znaczniki	Notacja
`html body header h1`	0	0	4	0,0,4
`body header.page-header h1`	0	1	3	0,1,3
`.page-header .title`	0	2	0	0,2,0
`#page-title`	1	0	0	1,0,0

Określenie, który selektor jest bardziej specyficzny, sprowadza się obecnie do porównania liczb. Specyficzność 1,0,0 ma pierwszeństwo względem specyficzności 0,2,2, a nawet w stosunku do specyficzności 0,10,0 (nie polecam jednak zapisywania selektorów tak długich jak selektor z 10 klasami), gdyż pierwsza liczba (identyfikatory) ma wyższy priorytet.

Sporadycznie używana jest notacja złożona z czterech liczb, gdzie 0 lub 1 to najbardziej znacząca cyfra, określająca to, czy deklaracja jest stosowana za pośrednictwem stylów liniowych. W tym wypadku styl liniowy ma specyficzność 1,0,0,0. Spowodowałoby to przesłonięcie zastosowanych stylów za pomocą selektorów, które mogłyby zostać wskazane jako mające specyficzność 0,1,2,0 (jeden identyfikator i dwie klasy) lub coś podobnego.

KWESTIE ZWIĄZANE ZE SPECYFICZNOŚCIĄ

Próba zastosowania przez Ciebie pomarańczowego tła przy użyciu selektora `.featured` nie udała się. Selektor `#main-nav` ma identyfikator przesłaniający selektor klasy (specyficzności 1,0,1 i 0,1,0). Aby to poprawić, możesz wziąć pod uwagę kilka opcji. Przyjrzyjmy się paru możliwym rozwiązaniom.

Najszybszym z nich jest dodanie adnotacji `!important` do deklaracji, która ma zostać wyróżniona. Zmodyfikuj deklarację tak, aby uwzględniała kod zamieszczony w listingu 1.7.

Listing 1.7. Możliwe pierwsze rozwiązanie

```
#main-nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}
.featured {
  background-color: orange !important;
}
```

Zapewnia ważność deklaracji, która w efekcie ma źródło o wyższym priorytecie.

Rozwiązanie to się sprawdza, ponieważ adnotacja `!important` ustanawia dla deklaracji źródło o wyższym priorytecie. Z pewnością jest to łatwe, ale też naiwne rozwiązanie. Na razie może być wystarczające, lecz później może spowodować problemy. Jeśli rozpoczniesz dodawanie adnotacji `!important` do wielu deklaracji, to, co się stanie, gdy konieczne okaże się wyróżnienie czegoś, co już zostało ustanowione jako ważne? Jeżeli użyjesz tej adnotacji dla kilku deklaracji, będzie mieć miejsce zgodność źródeł, dlatego obowiązywać będą zwykłe reguły specyficzności. Ostatecznie zakończy się to powrotem do punktu wyjścia. Gdy już rozpoczniesz korzystać z adnotacji `!important`, prawdopodobnie będziesz sięgać po nią coraz częściej.

Poznajmy lepszy sposób. Zamiast próbować obejść reguły specyficzności selektorów, spróbujmy sprawić, aby okazały się przydatne. Co sie stanie, gdy zwiększy się specyficzność danego selektora? Zmodyfikuj w kodzie CSS zestawy reguł tak, aby uwzględniały kod z listingu 1.8.

Listing 1.8. Możliwe drugie rozwiązanie

```
#main-nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}
#main-nav .featured {
  background-color: orange;
}
```

Pozostaje specyficzność 1,0,1.

Specyficzność jest zwiększana do 1,1,0.

Adnotacja !important nie jest już potrzebna.

Rozwiązanie to również się sprawdza. Selektor ma po jednym identyfikatorze i klasie, co zapewnia mu specyficzność 1,1,0, która jest większa niż w wypadku selektora `#main-nav a` (ze specyficznością 1,0,1), dlatego dla elementu stosowany jest pomarańczowy kolor tła.

Możliwe jest jednak zastosowanie jeszcze lepszego rozwiązania. Zamiast *zwiększania* specyficzności drugiego selektora sprawdźmy, czy możliwe jest *zmniejszenie* specyficzności pierwszego selektora. Element zawiera również klasę `<ul id="main-nav" class="nav">`. Oznacza to, że możesz zmienić kod CSS tak, aby odwoływał się do elementu za pomocą nazwy jego klasy, a nie jego identyfikatora. W sposób pokazany w listingu 1.9 zmień w selektorach `#main-nav` na `.nav`.

Listing 1.9. Możliwe trzecie rozwiązanie

```
.nav {
  margin-top: 10px;
  list-style: none;
  padding-left: 0;
}

.nav li {
  display: inline-block;
}
.nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}
.nav .featured {
  background-color: orange;
}
```

Zmienia #main-nav na .nav w całym arkuszu stylów.

Zmniejsza specyficzność (0,1,1) pierwszego selektora.

Zwiększa specyficzność (0,2,0) drugiego selektora.

Zmniejszyłeś specyficzność selektorów. Pomarańczowe tło jest na tyle jasne, żeby przesłonić turkusowy kolor.

Na podstawie przedstawionych przykładów widać, że specyficzność staje się raczej czymś w rodzaju „wyścigu zbrojeń". Dotyczy to szczególnie dużych projektów. Przeważnie, jeśli to możliwe, najlepiej utrzymać niską specyficzność. Dzięki temu, gdy pojawi się potrzeba przesłonięcia czegoś, możliwy będzie wybór.

1.1.3. Kolejność źródłowa

Trzecim i ostatnim krokiem analizowania kaskady jest kolejność źródłowa. Jeśli źródło i specyficzność są takie same, pierwszeństwo ma deklaracja pojawiająca się dalej w arkuszu stylów lub występująca w arkuszu, który dołączono później w obrębie strony.

Oznacza to, że możesz modyfikować kolejność źródłową w celu określenia stylu wyróżnionego odnośnika. Jeśli utworzysz dwa powodujące konflikt selektory o równej specyficzności, pierwszeństwo będzie mieć ten z nich, który pojawi się jako ostatni. Przyjrzyjmy się czwartej możliwości, zaprezentowanej w listingu 1.10.

Listing 1.10. Możliwe czwarte rozwiązanie

```
.nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}

a.featured {
  background-color: orange;
}
```

Tworzy jednakowe specyficzności (0,1,1).

W tym rozwiązaniu specyficzności są równe. Kolejność źródłowa określa, jaka deklaracja zostanie zastosowana dla odnośnika, co powoduje uzyskanie pomarańczowego, wyróżnionego przycisku.

Rozwiązuje to problem, ale potencjalnie powoduje też nowy problem: choć wyróżniony przycisk w obrębie deklaracji `nav` wygląda dobrze, co będzie, gdy zdecydujesz się użyć klasy `featured` w kolejnym odnośniku gdzieś na stronie poza deklaracją `nav`? W efekcie uzyskasz dziwną mieszankę stylów: pomarańczowe tło, lecz nie biały kolor tekstu, wypełnienie czy zaokrąglenie odnośników nawigacji (rysunek 1.7).

Rysunek 1.7. Klasa featured umieszczona poza deklaracją nav daje dziwne rezultaty

Listing 1.11 prezentuje kod znaczników powodujący takie działanie. W tym wypadku pojawia się element, do którego odwołuje się drugi selektor, lecz nie pierwszy, co daje niepożądany wynik. Konieczne będzie zdecydowanie, czy styl pomarańczowego przycisku ma funkcjonować poza deklaracją `nav`. Jeśli tak, niezbędne będzie zapewnienie, że są stosowane również dla niego wszystkie żądane style.

Listing 1.11. Kod wyróżnionego odnośnika poza deklaracją nav

```
<header class="page-header">
  <h1 id="page-title" class="title">Wypalacze kawy Wombat</h1>
  <nav>
    <ul id="main-nav" class="nav">
      <li><a href="/">Strona główna</a></li>
      <li><a href="/coffees">Kawy</a></li>
      <li><a href="/brewers">Kawiarki</a></li>
      <li><a href="/specials" class="featured">Promocje</a></li>
    </ul>
  </nav>
</header>
<main>
  <p>
    Koniecznie sprawdź
    <a href="/specials" class="featured">nasze promocje</a>.
  </p>
</main>
```

Dla kodu wyróżnionego odnośnika poza deklaracją nav style zostaną użyte częściowo.

Nie mając żadnych innych informacji o Twoich wymaganiach związanych z przykładową stroną internetową, skłaniałbym się do pozostania przy trzecim rozwiązaniu (listing 1.9). W idealnej sytuacji w wypadku własnej witryny internetowej będziesz w stanie dobrze zastanowić się nad swoimi wymaganiami. Być może wiesz, że prawdopodobnie będziesz potrzebować wyróżnionego odnośnika w innych miejscach. W takiej sytuacji bardziej właściwe będzie czwarte rozwiązanie (listing 1.10), które zostanie uzupełnione o style obsługujące klasę `featured` w innym miejscu strony.

Jak wcześniej wspomniałem, w CSS bardzo często najlepsza odpowiedź brzmi: „To zależy”. Istnieje wiele ścieżek prowadzących do osiągnięcia tego samego wyniku.

Warto rozważyć kilka możliwości i pomyśleć o konsekwencjach każdej z nich. Mając do czynienia z problemem dotyczącym stosowania stylów, często zajmuję się nim dwuetapowo. Po pierwsze, stwierdzam, jakie deklaracje zapewnią właściwy wygląd elementu. Po drugie, zastanawiam się nad możliwymi sposobami określającymi strukturę selektorów i wybieram ten, który najlepiej spełnia wymagania.

STYLE ODNOŚNIKÓW I KOLEJNOŚĆ ŹRÓDŁOWA

Po rozpoczęciu nauki CSS być może stwierdziłeś, że selektory użyte do określenia stylów odnośników powinny pojawiać się w konkretnej kolejności. Wynika to stąd, że kolejność źródłowa wpływa na kaskadę. Listing 1.12 prezentuje style odnośników na stronie we „właściwej" kolejności.

Listing 1.12. Style odnośników

```
a:link {
  color: blue;
  text-decoration: none;
}
a:visited {
  color: purple;
}
a:hover {
  text-decoration: underline;
}
a:active {
  color: red;
}
```

Kaskada jest powodem, dla którego ta kolejność jest istotna: w wypadku tej samej specyficzności późniejsze style przesłaniają te wcześniejsze. Jeśli dwa takie style lub więcej dotyczy jednocześnie jednego elementu, ostatni styl może przesłonić pozostałe. Jeśli użytkownik umieści kursor nad klikniętym odnośnikiem, style związane z operacją umieszczania będą mieć pierwszeństwo. Jeśli zaś użytkownik aktywuje odnośnik (czyli kliknie go) po umieszczeniu nad nim kursora, pierwszeństwo będą mieć style dotyczące aktywowania.

Mnemonikiem ułatwiającym zapamiętanie takiej kolejności są anglojęzyczne słowa tworzące kombinację *LoVe/HAte* (ang. *Link*, *Visited*, *Hover*, *Active* — odnośnik, odwiedzony, umieszczenie nad, aktywowanie). Zauważ, że jeśli zmodyfikujesz jeden z selektorów tak, aby miał inną specyficzność niż pozostałe, spowoduje to zaburzenie, czego efektem może być uzyskanie nieoczekiwanych wyników.

WARTOŚCI KASKADOWE

Aby przetworzyć każdą właściwość dla każdego elementu na stronie, przeglądarka postępuje zgodnie z trzema krokami reprezentowanymi przez pochodzenie, specyficzność i kolejność źródłową. Deklaracja, która „zwycięża" kaskadę, nazywana jest *wartością kaskadową*. Dla każdej właściwości każdego elementu istnieje co najwyżej jedna wartość kaskadowa. Określony akapit (<p>) na stronie może mieć marginesy górny i dolny, ale nie może mieć dwóch różnych górnych marginesów lub dwóch różnych

dolnych marginesów. Jeśli kod CSS określa różne wartości dla jednej właściwości, kaskada wybierze tylko jedną podczas renderowania elementu. Będzie to wartość kaskadowa.

WARTOŚĆ KASKADOWA — wartość określonej właściwości stosowana dla elementu jako wynik kaskady.

Jeśli właściwości nigdy nie określono dla elementu, nie ma on żadnej wartości kaskadowej dla tej właściwości. Na przykład ten sam akapit może nie mieć określonego obramowania lub wypełnienia.

1.1.4. Dwie praktyczne zasady

Jak być może wiesz, w korzystaniu z kaskady obowiązują dwie powszechne i praktyczne zasady. Ponieważ mogą one okazać się pomocne, oto ich przypomnienie:

1. *Nie używaj identyfikatorów w selektorze*. Nawet jeden identyfikator znacznie zwiększa specyficzność. Gdy niezbędne jest przesłonięcie selektora, a często nie dysponujesz kolejnym sensownym identyfikatorem możliwym do użycia, konieczne okazuje się skopiowanie oryginalnego selektora i dodanie kolejnej klasy w celu odróżnienia go od selektora, który próbujesz przesłonić.
2. *Nie używaj adnotacji* `!important`. Przesłonięcie tej adnotacji jest jeszcze trudniejsze niż identyfikatora. Po zastosowaniu po raz pierwszy owej adnotacji konieczne będzie dodawanie jej za każdym razem, gdy będzie miała zostać przesłonięta oryginalna deklaracja. W tej sytuacji w dalszym ciągu niezbędne będzie radzenie sobie ze specyficznością.

Te dwie zasady mogą być dobrą wskazówką, ale nie trzymaj się ich przy każdej sposobności. Są wyjątki, dla których zasady te mogą być przydatne, lecz nigdy nie stosuj ich jako odruchowej reakcji mającej na celu poradzenie sobie z konfliktem specyficzności.

Istotna uwaga dotycząca ważności

Gdy tworzysz moduł JavaScript przeznaczony do dystrybucji (np. w postaci pakietu NPM), szczególnie nakłaniam Cię do tego, aby nie stosować stylów w sposób liniowy za pośrednictwem kodu JavaScript, jeśli można tego uniknąć. W przeciwnym razie projektanci używający Twojego pakietu będą zmuszeni do zaakceptowania bez zmian zastosowanych stylów lub sięgnięcia po adnotację `!important` dla każdej właściwości, którą postanowią zmienić.

Zamiast tego do pakietu dołącz arkusz stylów. Jeśli utworzony komponent wymaga dynamicznego wprowadzania zmian stylów, prawie zawsze preferowane jest użycie kodu JavaScript do dodawania klas do elementów i usuwania z nich klas. Użytkownicy mogą wtedy skorzystać z arkusza stylów, a ponadto mają możliwość edytowania go w dowolny żądany sposób bez konieczności borykania się ze specyficznościami.

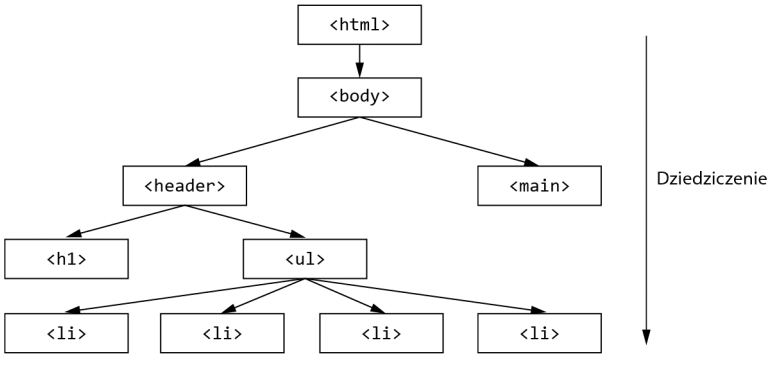
W ciągu kilku ostatnich lat pojawiła się seria praktycznych metod ułatwiających zarządzanie specyficznością selektorów. Przyjrzymy się im szczegółowo w rozdziale 9. Będzie w nim szerzej mowa o radzeniu sobie ze specyficznością. Wspomnę o jednym miejscu,

w którym poprawne jest użycie adnotacji `!important`. Na razie jednak, gdy już wiesz, jak działa kaskada, przejdziemy do następnego zagadnienia.

1.2. Dziedziczenie

Istnieje jeszcze jeden sposób, w jaki element może otrzymać style. Jest to **dziedziczenie**. Kaskada jest często łączona z pojęciem dziedziczenia. Choć te dwa zagadnienia są ze sobą powiązane, należy zrozumieć każde z osobna.

Jeśli element nie ma żadnej wartości kaskadowej w wypadku danej właściwości, może ją dziedziczyć po elemencie nadrzędnym. Powszechne jest stosowanie właściwości `font-family` względem elementu `<body>`. Wszystkie elementy nadrzędne zawarte w tym elemencie odziedziczą zatem czcionkę zapewnianą przez tę właściwość. Nie musisz jej stosować jawnie dla każdego elementu na stronie. Na rysunku 1.8 pokazano, jak dziedziczenie przebiega w dół drzewa modelu DOM.



Rysunek 1.8. Dziedziczone właściwości są przekazywane w dół drzewa modelu DOM od węzłów nadrzędnych do ich węzłów podrzędnych

Nie wszystkie jednak właściwości są dziedziczone. Domyślnie objęte są tym tylko niektóre właściwości. Zasadniczo są to właściwości, w wypadku których dziedziczenia będziesz *oczekiwać*. Są to głównie właściwości powiązane z tekstem: `color`, `font`, `font-family`, `font-size`, `font-weight`, `font-variant`, `font-style`, `line-height`, `letter-spacing`, `text-align`, `text-indent`, `text-transform`, `white-space` i `word-spacing`.

Dziedziczonych jest też kilka innych właściwości, takich jak właściwości list: `list-style`, `list-style-type`, `list-style-position` i `list-style-image`. Właściwości obramowania tabel `border-collapse` i `border-spacing` również są dziedziczone. Zauważ, że działanie tabel w zakresie obramowania kontrolują te dwie właściwości, a nie częściej stosowane właściwości służące do określania obramowań dla elementów niebędących tabelami (nie byłoby wskazane przekazywanie w dół przez element `<div>` swojego obramowania każdemu elementowi podrzędnemu). Powyższa lista nie jest kompletna, ale naprawdę niewiele do tego brakuje.

Na swojej stronie możesz wykorzystać dziedziczenie do własnych potrzeb, stosując czcionkę dla elementu `<body>` i umożliwiając odziedziczenie reprezentującej jej wartości przez elementy podrzędne tego elementu (rysunek 1.9).

Rysunek 1.9. Zastosowanie właściwości font-family względem elementu body i umożliwienie odziedziczenia tej samej wartości przez wszystkie elementy podrzędne

Dodaj kod z listingu 1.13 na początku arkusza stylów w celu uwzględnienia tej zasady dla własnej strony.

Listing 1.13. Zastosowanie właściwości font-family dla elementu nadrzędnego

```
body {
  font-family: sans-serif;
}
```

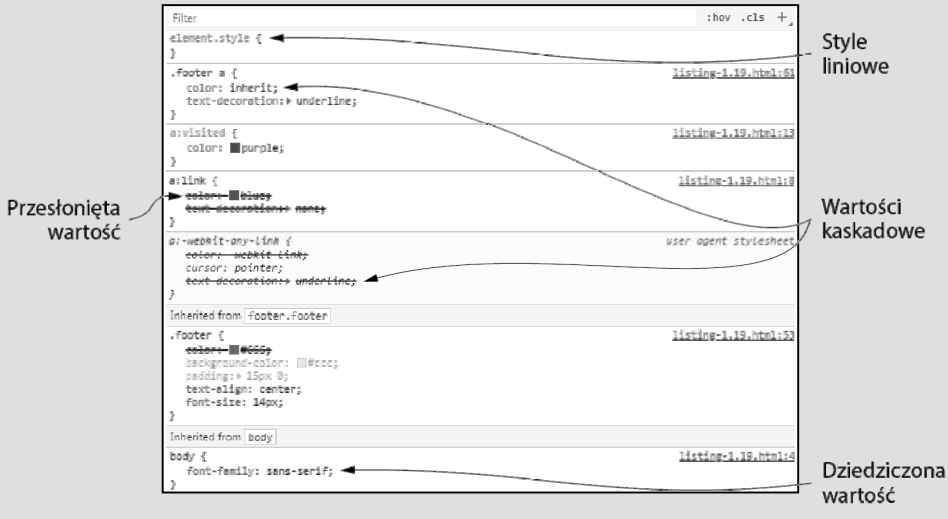
Dziedziczona właściwość będzie stosowana również dla elementów podrzędnych.

Dodanie właściwości do elementu `body` powoduje zastosowanie jej dla całej strony. Możliwe jest jednak odwołanie się do określonego elementu na stronie, który będzie dziedziczyć jedynie względem swoich elementów podrzędnych. Dziedziczenie będzie przechodzić od elementu do elementu do momentu przesłonięcia przez wartość kaskadową.

Używaj narzędzi dla programistów!

Skomplikowane zagnieżdżenie wartości dziedziczących i przesłaniających się wzajemnie może szybko okazać się trudne do opanowania. Jeśli nie jesteś jeszcze zaznajomiony z narzędziami dla programistów przeglądarki, nabierz nawyku korzystania z nich.

Narzędzia te zapewniają wgląd w to, jakie dokładnie reguły są stosowane względem jakich elementów i dlaczego. Kaskada i dziedziczenie to abstrakcyjne pojęcia: narzędzia dla programistów stanowią najlepszy sposób, jaki jest mi znany, aby się w tym zorientować. Uruchom te narzędzia przez kliknięcie elementu prawym przyciskiem myszy i wybranie z menu kontekstowego pozycji *Zbadaj* lub *Zbadaj element*. Poniżej pokazano przykład tego, co ujrzysz.



Inspektor stylów prezentuje każdy selektor odwołujący się do badanego elementu, porządkując elementy według ich specyficzności. Poniżej tych informacji widoczne są wszystkie dziedziczone właściwości. Od razu można zaznajomić się z całą kaskadą i dziedziczeniem dla elementu.

Istnieje wiele drobnych funkcji, które ułatwiają zrozumienie tego, co ma miejsce w wypadku stylów elementu. Style położone bliżej samej góry przesłaniają te znajdujące się niżej. Przesłonięte style są przekreślone. Po prawej stronie widoczna jest nazwa arkusza stylów oraz numer wiersza dla każdego zestawu reguł, dlatego możesz je znaleźć w kodzie źródłowym. Dzięki temu możesz dowiedzieć się dokładnie, który element odziedziczył które style oraz jakie jest ich źródło. Za pomocą widocznego u samej góry pola filtru możesz też zawęzić deklaracje do określonego zestawu.

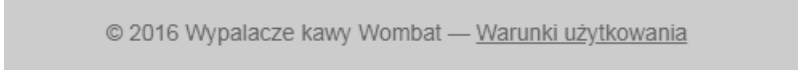
1.3. Wartości specjalne

Dostępne są dwie wartości specjalne, które możesz zastosować względem dowolnej właściwości w celu ułatwienia modyfikowania kaskady: `inherit` i `initial`. Przyjrzyjmy się tym wartościom.

1.3.1. Użycie słowa kluczowego inherit

Czasami wymagane będzie dziedziczenie, a wartość kaskadowa je uniemożliwia. Aby to było możliwe, możesz skorzystać ze słowa kluczowego `inherit`. Za jego pomocą możesz przesłonić inną wartość. W efekcie element odziedziczy tę wartość specjalną po swoim elemencie nadrzędnym.

Załóżmy, że do strony dodajesz jasnoszarą stopkę. W stopce mogą się znajdować odnośniki, ale nie mają zbytnio się wyróżniać, ponieważ stopka nie jest istotną częścią strony. W związku z tym dla odnośników w stopce ustawisz kolor ciemnoszary (rysunek 1.10).



Rysunek 1.10. Odnośnik Warunki użytkowania po odziedziczeniu przez niego szarego koloru tekstu

Na końcu strony umieść kod znaczników z listingu 1.14. Zwykła strona będzie mieć więcej zawartości między stopką i nagłówkiem, ale przykład ten spełni swoją funkcję.

Listing 1.14. Stopka z odnośnikiem

```
<footer class="footer">
  &copy; 2016 Wypalacze kawy Wombat &mdash;
  <a href="/terms-of-use">Warunki użytkowania</a>
</footer>
```

Kolor czcionki będzie zwykle ustawiony dla wszystkich odnośników na stronie (jeśli będzie inaczej, kolor zostanie określony przez style agenta użytkownika). Kolor ten jest też stosowany dla odnośnika *Warunki użytkowania*. Aby odnośnik w stopce miał szary kolor, niezbędne będzie jego przesłonięcie. W tym celu do arkusza stylów dodaj kod z listingu 1.15.

Listing 1.15. Wartość inherit

```
a:link {
  color: blue;
}
…
.footer {
  color: #666;
  background-color: #ccc;
  padding: 15px 0;
  text-align: center;
  font-size: 14px;
}
.footer a {
  color: inherit;
  text-decoration: underline;
}
```

Globalny kolor odnośników na stronie.

Kolor tekstu stopki ustawiono na szary.

Kolor czcionki jest dziedziczony ze stopki.

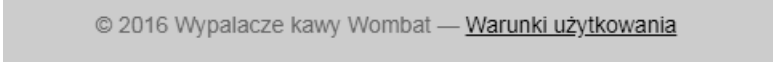
Trzeci zestaw reguł w powyższym kodzie przesłania niebieski kolor odnośnika, zapewniając odnośnikowi w stopce wartość kaskadową `inherit`. A zatem odnośnik dziedziczy kolor po swoim elemencie nadrzędnym `<footer>`.

Tutaj korzyścią jest to, że odnośnik w stopce będzie się zmieniać wraz z resztą jej zawartości, gdy cokolwiek ją zmodyfikuje (może to spowodować edycja drugiego zestawu reguł lub kolejny styl w innym miejscu może przesłonić styl stopki). Jeśli na przykład tekst stopki określonych stron jest bardziej ciemnoszary, odnośnik zostanie odpowiednio zmieniony.

Możesz też użyć słowa kluczowego `inherit`, aby wymusić dziedziczenie właściwości, która normalnie nie jest dziedziczona, takiej jak właściwość obramowania lub wypełnienia. W odniesieniu do tego można wymienić kilka praktycznych zastosowań, ale w rozdziale 3. zaznajomisz się z jednym przydatnym wariantem podczas omawiania określania wymiarów elementu.

1.3.2. Użycie słowa kluczowego initial

Czasami stwierdzisz, że zastosowałeś dla elementu style, które chcesz wycofać. W tym celu możesz skorzystać ze słowa kluczowego `initial`. Każda właściwość CSS ma wartość początkową lub domyślną. Jeśli danej właściwości przypiszesz wartość `initial`, spowoduje ona w efekcie ponowne ustawienie dla właściwości jej wartości domyślnej. Przypomina to trwałe zdefiniowanie tej wartości. Na rysunku 1.11 pokazano, jak renderowana jest stopka, gdy zamiast słowa kluczowego `inherit` zostanie dla niej ustawiona wartość `initial`.



Rysunek 1.11. Wartość początkowa właściwości koloru reprezentuje kolor czarny

OSTRZEŻENIE Słowo kluczowe `initial` nie jest obsługiwane w żadnej wersji przeglądarek Internet Explorer i Opera Mini. Słowo to jest rozpoznawane przez wszystkie inne podstawowe przeglądarki, w tym przez przeglądarkę Edge, czyli następcę przeglądarki Internet Explorer 11 firmy Microsoft.

Listing 1.16 prezentuje kod CSS użyty do utworzenia strony z rysunku 1.11. Ponieważ kolor czarny jest wartością początkową właściwości koloru w większości przeglądarek, kod `color: initial` odpowiada kodowi `color: black`.

Listing 1.16. Wartość initial

```
.footer a {
  color: initial;
  text-decoration: underline;
}
```

Korzyść z tego jest taka, że nie musisz zbytnio o tym myśleć. Aby usunąć obramowanie z elementu, użyj kodu `border: initial`. W celu przywrócenia dla elementu jego szerokości domyślnej zastosuj kod `width: initial`.

Być może żeby wykonać tego rodzaju operację resetowania, masz nawyk stosowania wartości `auto`. Okazuje się, że do osiągnięcia tego samego rezultatu możesz użyć kodu `width: auto`. Wynika to stąd, że `auto` to wartość domyślna właściwości `width`.

Godne uwagi jest jednak to, że `auto` nie jest wartością domyślną wszystkich właściwości. Wartość ta nie jest nawet poprawna w wypadku wielu właściwości. Na przykład kody `border-width: auto` i `padding: auto` są niepoprawne, a tym samym nie powodują żadnego efektu. Możesz poświęcić czas na określenie wartości początkowej tych właściwości, ale często łatwiej użyć wartości `initial`.

UWAGA Deklaracja `display: initial` odpowiada deklaracji `display: inline`. Nie przyjmie ona postaci deklaracji `display: block` niezależnie od tego, dla jakiego typu elementu zostanie zastosowana. Wynika to stąd, że słowo kluczowe `initial` przywraca wartość początkową dla właściwości, a nie dla elementu. `inline` to wartość domyślna właściwości związanych z wyświetlaniem.

1.4. Właściwości skrócone

Właściwości skrócone to właściwości umożliwiające jednoczesne ustawienie wartości dla kilku innych właściwości. Na przykład `font` to właściwość skrócona, która pozwala ustawić kilka właściwości czcionek. Następująca deklaracja określa właściwości `font-style`, `font-weight`, `font-size`, `line-height` i `font-family`:

```
font: italic bold 18px/1.2 "Helvetica", "Arial", sans-serif;
```

Podobnie:

- `background` to właściwość skrócona dla wielu właściwości tła: `background-color`, `background-image`, `background-size`, `background-repeat`, `background-position`, `background-origin`, `background-chip` i `background-attachment`.
- `border` to właściwość skrócona właściwości `border-width`, `border-style` i `border-color`, które również są właściwościami skróconymi.
- `border-width` to właściwość skrócona właściwości określających szerokości lewego, dolnego, prawego i górnego obramowania.

Właściwości skrócone są przydatne, ponieważ pozwalają zachować zwięzłość i przejrzystość kodu. Kilka dziwactw z nimi związanych nie jest jednak od razu oczywistych.

1.4.1. Uważaj na właściwości skrócone przesłaniające „po cichu" inne style

Większość właściwości skróconych umożliwia pominięcie określonych wartości i określenie tylko tego, czym jesteś zainteresowany. Godne uwagi jest jednak to, że przy postępowaniu w ten sposób w dalszym ciągu ustawiane są pominięte wartości. Niejawnie ustawiana jest dla nich ich wartość początkowa. Może to spowodować przesłonięcie „po cichu" stylów zdefiniowanych w innym miejscu. Jeśli na przykład użyłeś właściwości skróconej `font` dla tytułu strony bez określania właściwości wagi czcionki `font-weight`, nadal będzie ustawiona dla niej wartość `normal` (rysunek 1.12).

Rysunek 1.12. Właściwości skrócone spowodują ustawienie dla pominiętych właściwości ich wartości początkowej

Aby sprawdzić, jak to działa, do arkusza stylów dodaj kod z listingu 1.17.

Listing 1.17. Właściwość skrócona określająca wszystkie powiązane wartości

```
h1 {
  font-weight: bold;
}
.title {
  font: 32px Helvetica, Arial, sans-serif;
}
```

Początkowo może się wydawać, że kod `<h1 class="title">` spowoduje uzyskanie pogrubionego nagłówka. Tak jednak nie jest. Style widoczne w listingu 1.18 odpowiadają powyższemu kodowi.

Listing 1.18. Rozszerzony odpowiednik kodu z listingu 1.17 z właściwością skróconą

```
h1 {
  font-weight: bold;
}
.title {
  font-style: normal;
  font-variant: normal;
  font-weight: normal;
  font-stretch: normal;
  line-height: normal;
  font-size: 32px;
  font-family: Helvetica, Arial, sans-serif;
}
```

Wartości początkowe właściwości.

Oznacza to, że zastosowanie powyższych stylów względem elementu `<h1>` powoduje uzyskanie zwykłej wagi czcionki, a nie pogrubienia. Style te mogą też przesłonić inne style

czcionek, które w innym razie byłyby dziedziczone z elementu nadrzędnego. Spośród wszystkich właściwości skróconych właściwość `font` w najbardziej widoczny sposób powoduje problemy, ponieważ ustawia tak obszerny zestaw właściwości. Z tego powodu unikam tej właściwości, z wyjątkiem ustawiania za jej pomocą ogólnych stylów elementu `<body>`. Tego rodzaju problem możesz jednak napotkać w wypadku innych właściwości skróconych, dlatego miej świadomość takiej możliwości.

1.4.2. Kolejność wartości skróconych

Właściwości skrócone próbują „być łagodne", gdy pojawia się kwestia kolejności określonych wartości. Możesz ustawić deklarację `border: 1px solid black` lub deklarację `border: black 1px solid`. Obie zadziałają. Wynika to stąd, że dla przeglądarki jasne jest to, która wartość określa szerokość, która kolor, a która styl obramowania.

Istnieje jednak wiele właściwości, dla których wartości mogą być bardziej niejednoznaczne. Wówczas znacząca jest kolejność wartości. Ważne jest zrozumienie tego zagadnienia w odniesieniu do używanych właściwości skróconych.

GÓRNY, PRAWY, DOLNY I LEWY

Projektanci szczególnie borykają się z kolejnością właściwości skróconych, gdy mają do czynienia z właściwościami takimi jak `margin` i `padding` albo z niektórymi właściwościami obramowania, które określają wartości dla każdego z czterech boków elementu. W wypadku tych właściwości wartości obowiązuje kolejność zgodna z ruchem wskazówek zegara, począwszy od górnego boku.

Pamiętanie o tej kolejności może uchronić przed kłopotami. Okazuje się, że angielskie słowo *TRouBLe* (kłopot) stanowi mnemonik, którego możesz użyć do zapamiętania kolejności: *Top* (górny), *Right* (prawy), *Bottom* (dolny), *Left* (lewy).

Z powyższego mnemonika możesz skorzystać, aby ustawić wypełnienie dla czterech boków elementu. Odnośniki widoczne na rysunku 1.13 mają górne wypełnienie równe 10 px, prawe wypełnienie równe 15 px, dolne wypełnienie równe 0 oraz lewe wypełnienie wynoszące 5 px. Choć wygląda to nierówno, ilustruje zasadę.

Rysunek 1.13. Elementy z różnymi wypełnieniami z każdej strony

Listing 1.19 prezentuje kod CSS powiązany z tymi odnośnikami.

Listing 1.19. Określanie wypełnienia dla każdego boku elementu

```
.nav a {
  color: white;
  background-color: #13a4a4;
  padding: 10px 15px 0 5px; ←——— Górne, prawe, dolne i lewe wypełnienie.
  border-radius: 2px;
  text-decoration: none;
}
```

Właściwości, których wartości są zgodne z tym wzorcem, obsługują również notacje skrócone. Jeśli deklaracja kończy się, zanim zostanie podana wartość dla jednego z czterech

boków, bok ten uzyskuje swoją wartość od przeciwległego boku. Określ trzy wartości, a lewy i prawy bok użyją drugiej wartości. Podaj dwie wartości, a pierwsza z nich zostanie zastosowana przez górny i dolny bok. Jeśli podasz tylko jedną wartość, będzie ona obowiązywać dla wszystkich czterech boków. A zatem równorzędne są wszystkie poniższe deklaracje:

```
padding: 1em 2em;
padding: 1em 2em 1em;
padding: 1em 2em 1em 2em;
```

Następujące deklaracje także są wzajemnie równorzędne:

```
padding: 1em;
padding: 1em 1em;
padding: 1em 1em 1em;
padding: 1em 1em 1em 1em;
```

Dla wielu projektantów w tych deklaracjach największym problemem jest sytuacja, gdy zostały podane trzy wartości. Pamiętaj, że powodują one określenie górnego, prawego i dolnego boku. Ponieważ nie podano wartości lewego boku, przyjmie on taką samą wartość jak prawy bok. Druga wartość będzie dotyczyć zarówno lewego, jak i prawego boku. Oznacza to, że deklaracja `padding: 10px 15px 0` stosuje wypełnienie równe 15 px dla lewego i prawego boku, natomiast górne wypełnienie wynosi 10 px, a dolne wypełnienie jest równe 0.

Najczęściej jednak niezbędne będą dwie wartości. Zwłaszcza w wypadku mniejszych elementów często lepiej zastosować więcej wypełnienia dla lewego i prawego boku niż dla górnego i dolnego. Takie rozwiązanie dobrze się sprawdza dla przycisków lub odnośników nawigacji w obrębie strony (rysunek 1.14).

Rysunek 1.14. Wiele elementów wygląda lepiej po zastosowaniu większego wypełnienia poziomego

Zaktualizuj arkusz stylów w celu uwzględnienia kodu z listingu 1.20. W kodzie użyto właściwości skróconej, aby najpierw zastosować wypełnienie pionowe, a następnie poziome.

Listing 1.20. Określanie dwóch wartości wypełnienia

```
.nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px 15px; <-- Najpierw wypełnienie górne i dolne, a następnie lewe i prawe.
  border-radius: 2px;
  text-decoration: none;
}
```

Ze względu na to, że wiele typowych właściwości zgodnych jest z tym wzorcem, warto utrwalić sobie w pamięci tę kolejność.

POZIOMO I PIONOWO

Mnemonik w postaci angielskiego słowa *TRouBLe* dotyczy wyłącznie właściwości, które są stosowane osobno dla wszystkich czterech boków elementu. Inne właściwości obsługują jedynie dwie wartości. Wśród nich są właściwości `background-position`, `box-shadow` i `text-shadow` (choć, mówiąc wprost, nie są to właściwości skrócone). W porównaniu z właściwościami z czterema wartościami, takimi jak `padding`, kolejność tych dwóch wartości jest odwrotna. Deklaracja `padding: 1em 2em` określa najpierw wartości góry i dołu w pionie, a następnie wartości lewej i prawej strony w poziomie. Z kolei deklaracja `background-position: 25% 75%` określa najpierw wartości lewej i prawej strony w poziomie, a później wartości góry i dołu w pionie.

Choć wydaje się nieintuicyjne to, że obie deklaracje określają wartości odwrotnie, przyczyna tego jest oczywista: dwie wartości reprezentują siatkę kartezjańską. W jej wypadku miary są zwykle podawane w kolejności x, y (oś pozioma, a następnie pionowa). Gdyby na przykład wymagane było użycie cienia podobnego do pokazanego na rysunku 1.15, należałoby najpierw określić wartość x (poziomą).

Rysunek 1.15. Cień elementu z określonym położeniem 10px 2px

Listing 1.21 prezentuje style użyte dla przykładowego elementu.

Listing 1.21. Właściwość box-shadow określająca najpierw wartość x, a następnie wartość y

```
.nav .featured {
  background-color: orange;
  box-shadow: 10px 2px #6f9090;
}
```

Przesunięcie cienia o 10 px w prawo i 2 px w dół.

Pierwsza (większa) wartość dotyczy przesunięcia w poziomie, natomiast druga (mniejsza) wartość odnosi się do przesunięcia w pionie.

Jeśli stosujesz właściwość, która określa dwie miary względem narożnika, myśl „kategoriami siatki kartezjańskiej". Jeśli zaś korzystasz z właściwości określającej miary dla każdego boku elementu, myśl „kategoriami zegara".

Podsumowanie

- Utrzymaj kontrolę nad specyficznością selektorów.
- Nie myl kaskady z dziedziczeniem.
- Określone właściwości są dziedziczone, w tym właściwości powiązane z tekstem, listami i obramowaniem tabel.
- Nie myl ze sobą wartości `initial` i `auto`.
- Unikaj kłopotów z właściwościami skróconymi (mnemonik *TRouBLe*).

Skorowidz

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

W

Z

Ź

CSS od podszewki

W ostatnich latach CSS bardzo się rozwinął. Mimo że istnieje od kilku dekad, wciąż jest ważnym elementem warsztatu profesjonalnych projektantów stron internetowych. Co prawda przyswojenie podstaw CSS nie jest specjalnie trudne, ale osiągnięcie prawdziwej biegłości w korzystaniu z tego języka wymaga stałego uczenia się i ciągłych ćwiczeń. Trzeba dobrze zrozumieć wszystkie części kodu CSS, a także sposób, w jaki są do siebie dopasowane. Wysiłek włożony w doskonałe opanowanie języka szybko się zwróci: CSS pozwala zwiększyć wygodę użytkownika, przyspieszyć proces projektowania, uniknąć potencjalnych błędów, wreszcie — urozmaicić i ożywić aplikację.

Ta książka jest kompleksowym przewodnikiem po języku CSS dla osób na różnych poziomach biegłości w stosowaniu kaskadowych arkuszy stylów. Zawiera kluczowe informacje o jego podstawach, dzięki czemu będzie nieocenioną pomocą dla początkujących. Szczegółowo opisano tu również nowości i ulepszenia, które pojawiły się w języku na przestrzeni ostatnich kilku lat. Zaprezentowano także kwestie istotne dla zaawansowanych projektantów, takie jak organizacja kodu w sposób umożliwiający wielokrotne wykorzystywanie czy budowanie niezmiernie przydatnej w pracy zespołowej biblioteki wzorców. Sporo miejsca poświęcono technikom uzyskiwania efektów decydujących o atrakcyjnym i niepowtarzalnym wyglądzie aplikacji.

W tej książce między innymi:

- kaskady, dziedziczenie i moduł pudełkowy
- układy, w tym flexbox i siatka
- modularna organizacja kodu
- tajniki typografii internetowej
- używanie cieni, gradientów, kontrastów i odstępów
- transformacje, przejścia i animacje

CSS. Każda aplikacja może wyglądać pięknie!

Keith J. Grant od 11 lat pisze i utrzymuje aplikacje internetowe oparte na HTML, CSS i JavaScripcie. Obecnie pracuje w Intercontinental Exchange Inc. (ICE), gdzie stworzył i utrzymuje kod CSS witryn internetowych wielkich korporacji, a także giełdy papierów wartościowych w Nowym Jorku. Pisze kod CSS, który umożliwia firmom prezentowanie ich witryn w unikalny i kreatywny sposób. Chętnie dzieli się swoimi doświadczeniami z podopiecznymi — w każdej firmie, w której pracował, szybko stawał się najlepszym instruktorem języka CSS.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

ISBN 978-83-283-4446-4

9 788328 344464

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 79,00 zł