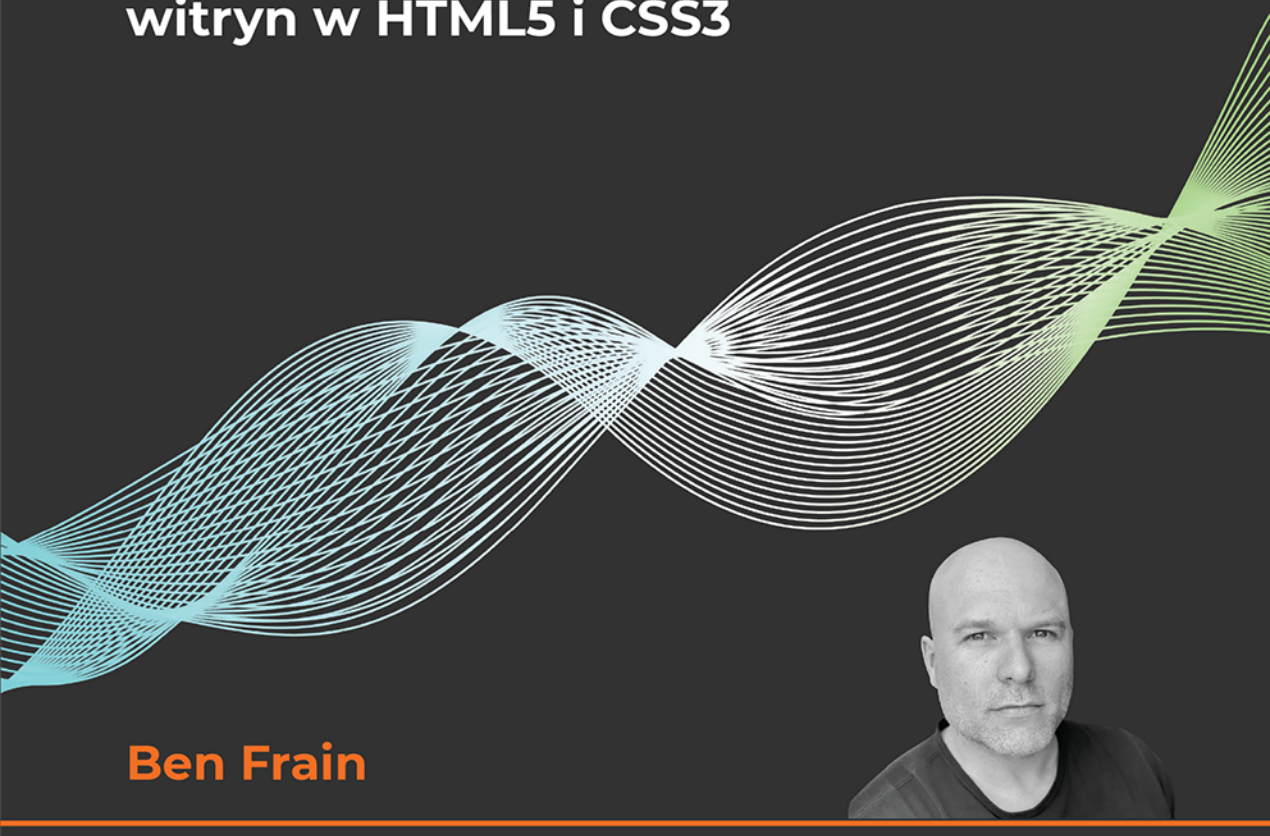
Wydanie III

Responsive Web Design

Projektowanie elastycznych witryn w HTML5 i CSS3



Ben Frain

Helion

Packt

Tytuł oryginału: Responsive Web Design with HTML5 and CSS - Third Edition Develop future-proof responsive websites using the latest HTML5 and CSS techniques

Tłumaczenie: Maksymilian Gutowski, z wykorzystaniem fragmentów poprzednich wydań w przekładzie Macieja Reszotnika i Łukasza Piwki.

ISBN: 978-83-283-7419-5

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)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/reswe3.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/reswe3
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

5

ROZDZIAŁ

Układy oparte na siatce CSS

Największym przełomem w tworzeniu układów CSS jest bez wątpienia Grid, czyli siatka.

Dysponujemy teraz systemem do tworzenia układów, który umożliwia uzyskanie wszystkiego tego, co było dla nas do tej pory dostępne przy użyciu mniejszej ilości kodu, a jednocześnie w sposób bardziej przewidywalny. Ponadto dzięki niemu możemy osiągnąć efekty, które wcześniej nie były możliwe!

Powiedziałbym wręcz, że siatka CSS nie jest rezultatem ewolucji, tylko rewolucji. W tym module pojawiły się zupełnie nowe koncepcje, które nie miały żadnych odpowiedników w poprzednich wersjach CSS. Bądź wobec tego świadom, że przyzwyczajenie się do posługiwania się nimi może zająć trochę czasu. Zaufaj mi jednak — jest to warte swojej ceny. Zaczynamy!

- W tym rozdziale:
- dowiesz się, czym jest CSS Grid i jakie problemy rozwiązuje;
- poznasz najważniejsze pojęcia stosowane w pracy z układami opartymi na siatce;
- poznasz terminologię dotyczącą siatki;
- dowiesz się, jak utworzyć siatkę;
- dowiesz się, jak rozmieszczać w siatce elementy;
- dowiesz się, jak tworzyć sprawne, responsywne wzorce z wykorzystaniem minimalnej ilości kodu;
- nauczysz się składni zbiorczej reguł siatki.

Pod koniec rozdziału wykonasz także krótkie ćwiczenie — wykorzystasz w nim niektóre z przedstawionych do tej pory technik, aby przeprowadzić refaktoryzację części układu witryny *https://rwd.education*, z którą zetknąłeś się w poprzednich rozdziałach.

CSS Grid i problemy, które rozwiązuje

Siatka CSS jest systemem tworzenia dwuwymiarowych układów. Flexbox, który omówiliśmy w poprzednim rozdziale, obsługuje rozmieszczanie elementów w jednym wymiarze (lub kierunku). Kontener flexboksa rozkłada elementy albo w rzędzie, albo w kolumnie, ale nie umożliwia rozmieszczania ich poziomo i pionowo jednocześnie — od tego jest siatka.

Powinienem od razu zwrócić Ci uwagę, że nie musisz wybierać między flexboksem a siatką. Te dwa systemy nie wykluczają się wzajemnie. Zazwyczaj używam obu, niekiedy nawet w ramach pojedynczego komponentu wizualnego.

Żeby było zupełnie jasne, zastosowanie siatki nie wiąże się z odrzuceniem innych metod wyświetlania elementów. Siatka może bez problemu zmieścić w sobie kontenery giętkie, a elementy interfejsu utworzone w siatce można również zawrzeć we flexboksie, w standardowym bloku lub bloku śródliniowym.

Najlepsze rozwiązanie jest uzależnione od okoliczności: czasami jest nim siatka, czasami flexbox, a czasami jeszcze inny rodzaj układu.

Prawda jest taka, że układy siatkowe tworzyliśmy w CSS już od wielu lat, ale zwyczajnie nie mieliśmy konkretnego mechanizmu, który by do tego służył. Korzystaliśmy z bloków, pływających elementów, tabel i innych pomysłowych technik do ominięcia problemu, jakim był brak właściwego systemu do tworzenia układów opartych na siatkach w CSS. Teraz jest na szczęście inaczej.

Przy użyciu CSS Grid możemy tworzyć siatki o niemal nieskończonej liczbie różnych form i rozmieszczeniu ich elementów składowych tam, gdzie chcemy, niezależnie od kolejności elementów w kodzie źródłowym. Co więcej, siatka uwzględnia także sytuacje, w których dodawane są nowe elementy, i dostosowuje się do bieżących potrzeb. Sam ten opis może się jednak wydawać nadmiernie pochlebny, więc nie marnujmy czasu i przejdźmy do rzeczy.

Podstawowa składnia siatki

Aby skorzystać z CSS Grid, musimy przekazać przeglądarce informacje o tym:

- z ilu rzędów i kolumn siatka będzie się składać;
- jakie te rzędy i kolumny mają mieć rozmiary;
- gdzie na siatce mają się znaleźć elementy;
- co powinno się stać, kiedy rozmiar siatki ulegnie zmianie lub doda się do niej więcej elementów.

Wymaga to zatem po prostu zapoznania się z właściwą terminologią.

Pojęcia i terminy dotyczące siatki

Na początek należy zapoznać się z pojęciami „jawnej" i „niejawnej" definicji rozmieszczenia elementów. Siatka definiowana w arkuszu stylów przy użyciu kolumn i rzędów jest siatką definiowaną jawnie; to rozmieszczenie elementów, które jednoznacznie określiłeś. Siatka definiowana niejawna jest z kolei rozmieszczeniem elementów, z jakimi mamy do czynienia, kiedy w siatce pojawiają nieprzewidziane dodatkowe elementy. Rozmieszczenie tych nowych elementów jest określane na podstawie układu siatki jawnej.

Kolejną właściwością, która jest dla wielu ludzi niejasna (a z całą pewnością była dla mnie), jest fakt, że linie siatki znajdują się po obu stronach elementów siatki. Obszar pośrodku, pomiędzy liniami, nazywa się „torem" siatki. Kiedy dwa tory z dwóch kierunków się ze sobą krzyżują, powstaje „obszar siatki".

Najważniejsze jest to, że elementy w siatce możesz rozmieszczać w odniesieniu do linii siatki (wskazując tym samym obszar siatki) lub samych obszarów, jeżeli zostały one nazwane.

> Liczba torów w siatce jest teoretycznie nieograniczona. Przeglądarki mają jednak możliwość przycinania siatek. Jeżeli wskażesz wartość przekraczającą limit przeglądarki, siatka zostanie do tego limitu przycięta. W praktyce przekroczenie limitu przeglądarki wydaje się mało prawdopodobne, ale wspominam o tym dla porządku.

Tworzenie siatki

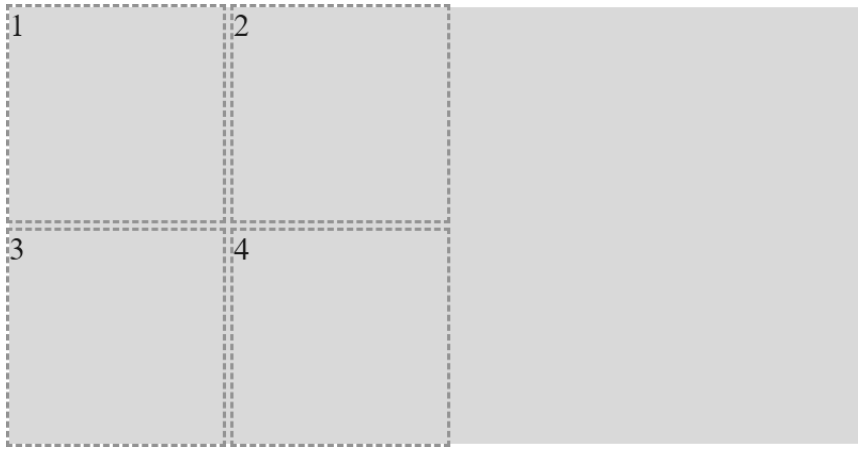
Oto wprowadzenie do sekcji specyfikacji W3C dotyczącej siatki definiowanej jawnie: *https://www.w3.org/TR/css-grid-1/#explicit-grids*. Warto je przeczytać kilkakrotnie, ponieważ jest pełne informacji istotnych dla zrozumienia, w jaki sposób siatka działa:

> „Trzy właściwości: `grid-template-rows`, `grid-template-columns` i `grid-template-areas` wspólnie definiują jawnie rozkład kontenera siatki. Ukończona siatka może się okazać większa, niż można by przypuszczać, ze względu na pojawienie się elementów poza siatką jawną; w takim przypadku tworzy się tory niejawne, których wielkość jest określana na podstawie własności `grid-auto-rows` i `grid-auto-columns`.
>
> Wielkość siatki zdefiniowanej jawnie jest określona przez wyższą wartość spośród liczby rzędów lub kolumn zdefiniowanej przez `grid-template-areas` oraz liczby rzędów lub kolumn o wymiarach zdefiniowanych przez własności `grid-template-rows` lub `grid-template-columns`.
>
> Wszystkie rzędy lub kolumny, które zdefiniowano w `grid-template-areas`, lecz których rozmiarów nie określono własnościami `grid-template-rows` lub `grid-template-columns`, uzyskują rozmiar oparty na własnościach `grid-auto-rows` lub `grid-auto-columns`. Jeśli te własności nie definiują jawnych torów, każda oś siatki zdefiniowanej jawnie składa się z jednej linii.

> Indeksy numeryczne we własnościach `grid-placement` są oznaczeniami elementów liczonych od krawędzi siatki zdefiniowanej jawnie. Wartości dodatnie określają położenie elementów względem punktu początkowego (zaczynając od 1), podczas gdy ujemne wskazują położenie względem punktu końcowego (zaczynając od –1).
>
> Własności `grid` i `grid-template` umożliwiają stosowanie deklaracji zbiorczych, za pomocą których można zdefiniować wszystkie trzy własności siatki zdefiniowanej jawnie (`grid-template-rows`, `grid-template-columns` i `grid-template-areas`). Deklaracja zbiorcza wyzerowuje także własności kontrolujące siatkę zdefiniowaną niejawnie, podczas gdy `grid-template` pozostawia je niezmienionymi".

Wiem, że jeśli w ogóle nie pracowałeś z CSS Grid, może to wyglądać dość onieśmielająco, a powyższy wyimek ze specyfikacji może Ci się wydawać zupełnie niezrozumiały. Mam jednak nadzieję, że po przeczytaniu tego rozdziału i samodzielnym poeksperymentowaniu z siatką wszystko stanie się jaśniejsze.

Nie martw się, jeśli powyższe było w dużej mierze niezrozumiałe. Zacznijmy pracę z siatką od bardzo prostego przykładu. Oto najprostszy układ siatkowy, składający się z czterech ponumerowanych pól. W przeglądarce będzie on wyglądał tak jak na rysunku 5.1.



Rysunek 5.1. Nasza pierwsza siatka; prościej się nie da

Oto kod HTML:

```
<div class="my-first-grid">
    <div class="grid-item-1">1</div>
    <div class="grid-item-2">2</div>
    <div class="grid-item-3">3</div>
    <div class="grid-item-4">4</div>
</div>
```

Zwróć uwagę na to, że przy korzystaniu z CSS Grid używamy kontenera, który odpowiada siatce jako całości, a elementy siatki są jego bezpośrednimi potomkami. Znaczniki potom-

ków należy pisać w kolejności, która wydaje się najsensowniejsza z perspektywy treści; siatka pozwala na umieszczenie ich w układzie tam, gdzie tego potrzebujesz. Oto arkusz stylów:

```
.my-first-grid {
    display: grid;
    grid-gap: 10px;
    grid-template-rows: 200px 200px;
    grid-template-columns: 200px 200px;
    background-color: #e4e4e4;
}

.grid-item-1 {
    grid-row: 1;
    grid-column: 1;
}
.grid-item-2 {
    grid-row: 1;
    grid-column: 2;
}
.grid-item-3 {
    grid-row: 2;
    grid-column: 1;
}
.grid-item-4 {
    grid-row: 2;
    grid-column: 2;
}

[class^='grid-item'] {
    outline: 3px dashed #f90;
    font-size: 30px;
    color: #333;
}
```

Należy tutaj zwrócić uwagę na własności dotyczące siatki. Dodałem kilka obramowań i tło, aby łatwiej było zobaczyć rozkład siatki oraz wielkość i kształt jej elementów.

Wskazujemy kontener jako siatkę przy użyciu własności `display: grid`, następnie używamy `grid-template-rows: 200px 200px`, aby zdefiniować dwa rzędy, oba wysokie na 200 pikseli, wreszcie `grid-template-columns: 200px 200px`, aby zdefiniować w siatce dwie kolumny, obie szerokie na 200 pikseli.

W odniesieniu do potomków kontenera używamy własności `grid-row` z podaną wartością liczbową, która wskazuje, w którym rzędzie element ma być rozmieszczony, oraz `grid-column`, która wskazuje odpowiednią kolumnę.

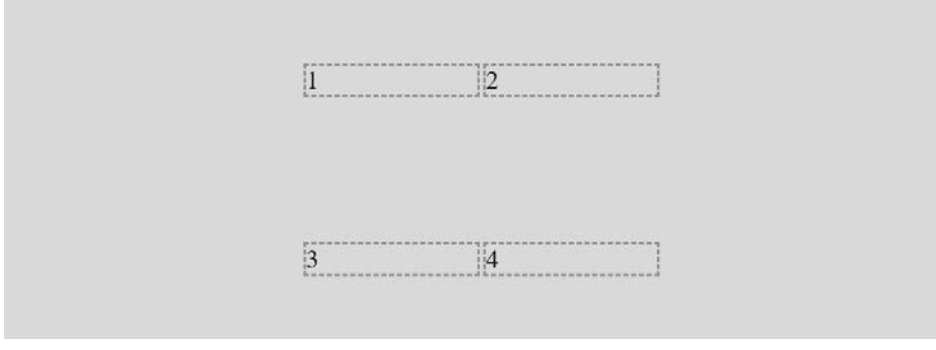
Potomkowie siatki domyślnie mają standardowy układ. Mimo że elementy siatki wchodzą w jej skład, w tym przykładzie, ponieważ wszystkie pozostają elementami `div`, wciąż są wyświetlane zgodnie z deklaracją `display: block`. To istotna kwestia, kiedy już zajmujemy się rozmieszczaniem elementów siatki.

Możesz poeksperymentować z pierwszym przykładem, używając kodu z folderu *05-01*.

Wykorzystajmy omówione w poprzednim rozdziale własności wyrównywania, aby spróbować wyśrodkować elementy siatki.

```
.my-first-grid {
    display: grid;
    grid-gap: 10px;
    grid-template-rows: 200px 200px;
    grid-template-columns: 200px 200px;
    background-color: #e4e4e4;
    align-items: center;
    justify-content: center;
}
```

Kiedy po raz pierwszy miałem do czynienia z CSS Grid i wypróbowałem takie mniej więcej rozwiązanie, spodziewałem się, że liczby zostaną idealnie wyśrodkowane w swoich torach siatki. Tak jednak nie było (rysunek 5.2).



Rysunek 5.2. Jeśli nie określisz szerokości, siatka wykorzysta całą dostępną przestrzeń

Jeśli zastanowimy się przez moment, co właściwie zrobiliśmy, wszystko zacznie mieć sens. Utworzyliśmy siatkę z dwiema kolumnami i dwoma rzędami, przy czym każdy z tych komponentów ma wysokość i szerokość 200 pikseli, a następnie poleciliśmy wyśrodkowanie elementów w poziomie i pionie. Ponieważ użyliśmy wartości `grid` zamiast `inline-grid`, siatka wypełnia całą szerokość strony, pomimo że elementy naszej siatki nie potrzebują tyle przestrzeni.

Zmodyfikujmy ten kod tak, aby szerokość siatki była dopasowana do jej zawartości. Ponadto wyśrodkujmy elementy względem nadrzędnych elementów siatki. W tym celu musimy sprawić, aby same stały się elementami flexboksa lub siatki. Ponieważ w tym rozdziale zajmujemy się CSS Grid, wprowadźmy odpowiednie zmiany:

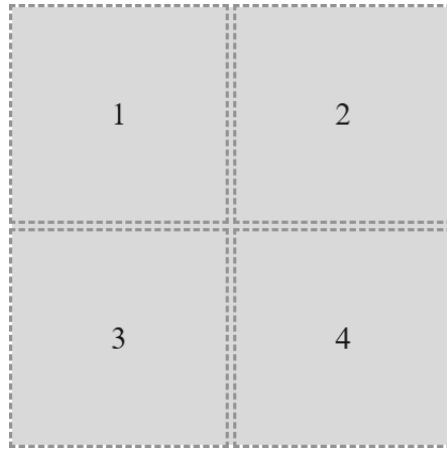
```
.my-first-grid {
    display: inline-grid;
    grid-gap: 10px;
```

```
    grid-template-rows: 200px 200px;
    grid-template-columns: 200px 200px;
    background-color: #e4e4e4;
}

[class^='grid-item'] {
    display: grid;
    align-items: center;
    justify-content: center;
    outline: 3px dashed #f90;
    font-size: 30px;
    color: #333;
}
```

> Jeśli nie wiesz, do czego służy selektor z daszkiem, nie przejmuj się — zajmiemy się tym w rozdziale 6.

Nadaliśmy własności `display` kontenera wartość `inline-grid`, a wszystkim elementom siatki wartość `grid`, po czym zastosowaliśmy własności wyrównywania `justify-content` i `align-items`.

Uzyskaliśmy tym samym rezultat widoczny na rysunku 5.3.



Rysunek 5.3. Przekształcenie dziedziców kontenera w elementy giętkie lub elementy siatki umożliwia wyśrodkowanie ich zawartości

> Kod tego przykładu znajdziesz w folderze *05-02*. W ramach ćwiczenia spróbuj przenieść elementy siatki do innych rzędów i kolumn.

Udało nam się poczynić pewne postępy. Przejdźmy teraz do kwestii jawnego i niejawnego rozmieszczania elementów.

Jawne i niejawne definicje rozmieszczenia

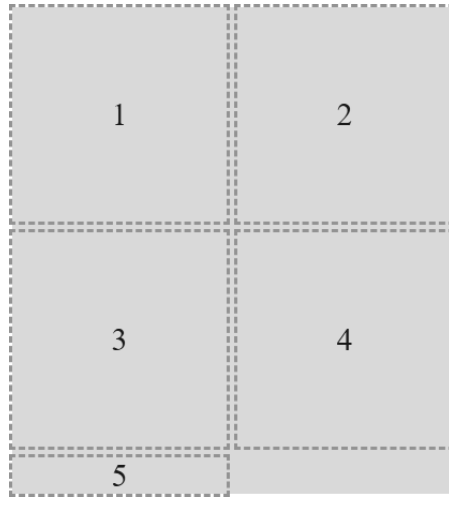
We wcześniejszej części tego rozdziału poruszyliśmy kwestię różnic pomiędzy jawnymi i niejawnymi definicjami siatki; siatka jawna jest strukturą określoną w arkuszu stylów w ramach siatki jako takiej. Kiedy w układzie pojawia się więcej treści, niż przygotowano na to miejsca, do akcji wkracza siatka „niejawna".

Przyjrzyjmy się temu bliżej, korzystając z poprzedniego przykładu.

Dodajmy kolejny element i zobaczmy, co się stanie.

```
<div class="my-first-grid">
    <div class="grid-item-1">1</div>
    <div class="grid-item-2">2</div>
    <div class="grid-item-3">3</div>
    <div class="grid-item-4">4</div>
    <div class="grid-item-5">5</div>
</div>
```

W przeglądarce otrzymujemy rezultat widoczny na rysunku 5.4.



Rysunek 5.4. Siatka dodała element, lecz nie ma on takich proporcji, jakich byśmy oczekiwali

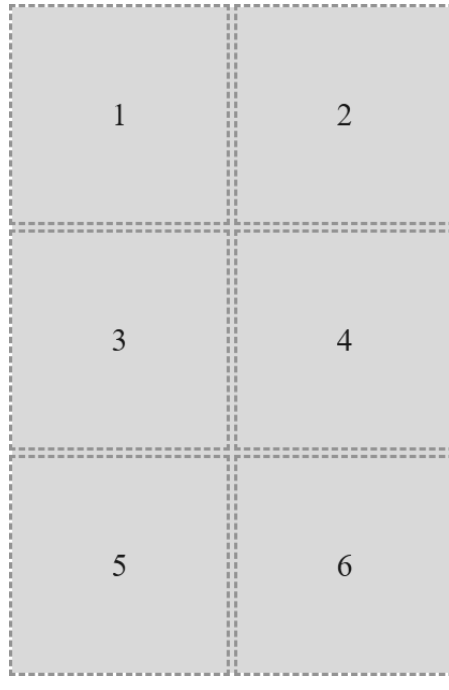
To na swój sposób przydatne; siatka utworzyła niejawne linie określające tor, na którym pojawił się nowy element. Nie wskazaliśmy jej, co powinna zrobić z dodatkowym elementem, więc postarała się odgadnąć najlepsze ustawienie. Mamy jednak możliwość kontrolowania sposobu, w jaki niejawnie zdefiniowana siatka obsługuje elementy. Umożliwiają to własności `grid-auto-rows` i `grid-auto-columns`.

Własności grid-auto-rows i grid-auto-columns

Przy użyciu własności `grid-auto-rows` i `grid-auto-columns` sprawimy, aby wszelkie dodatkowe elementy siatki miały takie same wymiary jak pozostałe.

```
.my-first-grid {
    display: inline-grid;
    grid-gap: 10px;
    grid-template-rows: 200px 200px;
    grid-template-columns: 200px 200px;
    grid-auto-rows: 200px;
    grid-auto-columns: 200px;
    background-color: #e4e4e4;
}
```

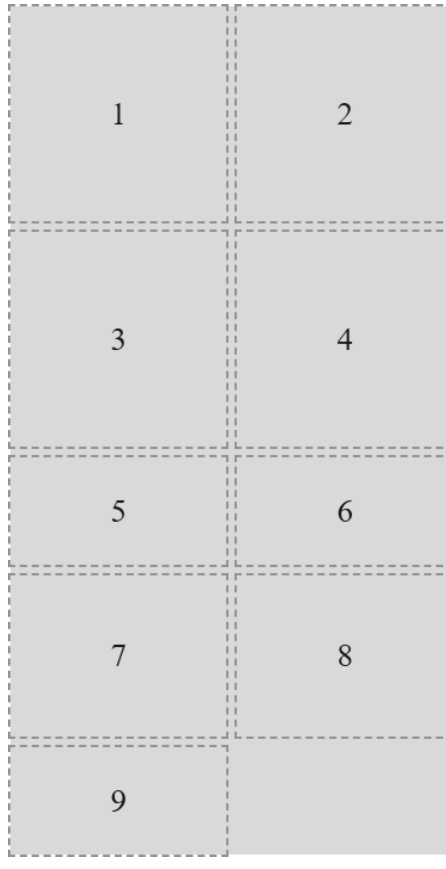
Teraz nawet bez dodatkowego kodu CSS kolejne elementy umieszczane w siatce będą miały wysokość i szerokość 200 pikseli. Wprowadziliśmy tutaj kolejny element do DOM, tworząc ich łącznie sześć (rysunek 5.5).



Rysunek 5.5. Wystarczy kilka automatycznie działających własności, aby uzyskać pożądaną wielkość elementów

Można nawet tworzyć wzorce, za sprawą których pierwszemu dodatkowemu elementowi nadaje się jeden rozmiar, a drugiemu inny. Ustawienia takich wzorców powtarzają się cyklicznie (rysunek 5.6).

```
.my-first-grid {
    display: inline-grid;
    grid-gap: 10px;
    grid-template-rows: 200px 200px;
    grid-template-columns: 200px 200px;
    grid-auto-rows: 100px 150px;
    grid-auto-columns: 100px 150px;
    background-color: #e4e4e4;
}
```



Rysunek 5.6. Możesz zdefiniować wzorzec określający wymiary automatycznie dodawanych rzędów i kolumn

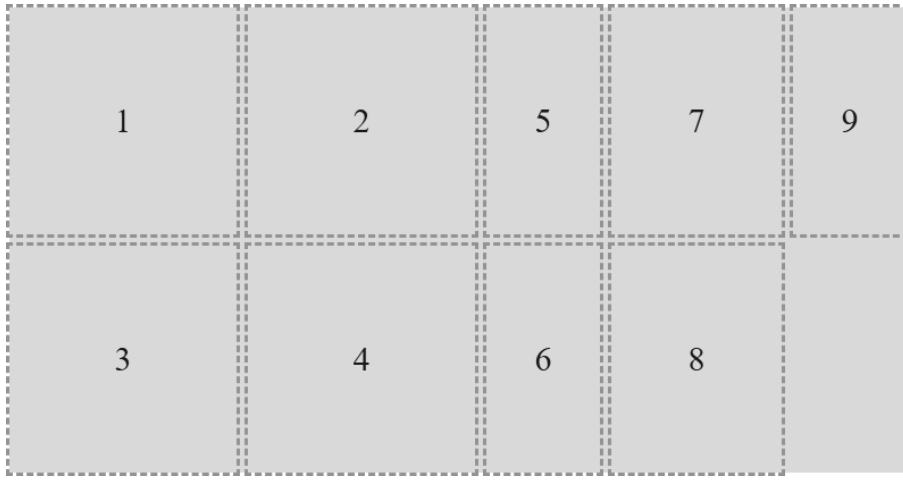
Jak widzisz, wszystkie elementy, począwszy od piątego, mają wymiary określone wartościami własności `grid-auto-rows`. Elementy w pierwszym dodanym rzędzie mają wysokość 100 pikseli, w drugim 150 pikseli, a w trzecim znowu 100 pikseli.

Do tej pory zajmowaliśmy się elementami siatki rozmieszczanymi w pionie. Z łatwością możesz jednak sprawić, aby ten układ przebiegał w poziomie. Jeśli chcesz, poeksperymentuj z kodem w folderze *05-03*.

Własność grid-auto-flow

Własność `grid-auto-flow` umożliwia zdefiniowanie kierunku, w którym dodatkowe niejawne elementy są układane w siatce. Użyj wartości `column`, jeżeli chcesz, aby dodatkowe elementy pojawiały się w dodatkowych kolumnach, a `row`, jeśli mają się pojawiać w dodatkowych rzędach.

Dodajmy teraz `grid-auto-flow: column`, aby elementy były rozmieszczane w poziomie zamiast w pionie (rysunek 5.7).



Rysunek 5.7. Siatka, w której dodatkowe elementy dostawiane są w poziomie

Do własności `grid-auto-flow: column` i `grid-auto-flow: row` można dodać słowo kluczowe `dense`, które omówimy wkrótce.

Rozmieszczanie i definiowanie rozmiaru elementów siatki

Do tej pory każdy dodany do siatki element zajął pojedynczy jej obszar. Zajmiemy się teraz kolejnym przykładem (którego kod znajdziesz w folderze *05-04*). Ta siatka będzie składała się z 20 elementów; są to przypadkowe artykuły spożywcze oznaczone liczbami porządkowymi. Kodowi CSS przyjrzymy się jednak bliżej. Zanim omówimy po kolei wszystkie nowości, rzuć okiem na kod i rysunek 5.8, aby zorientować się, w jakim stopniu rozumiesz, co się dzieje.

Warto też wspomnieć, że celowo użyłem różnych odstępów w wartościach własności. Można pisać zarówno `grid-row: 6 / span 2`, jak i `grid-row: 6/span 2` — obie formy są poprawne. Sam zdecyduj, która Ci bardziej odpowiada.

Oto kod HTML:

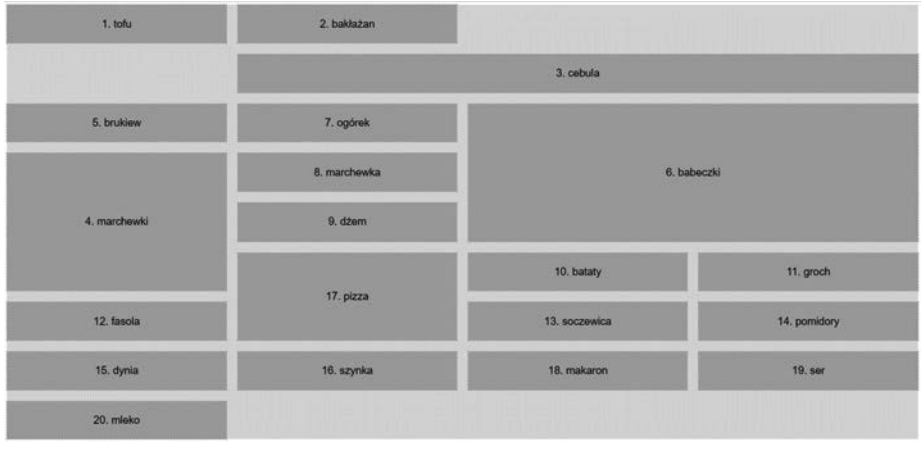
```
<div class="container">
    <div class="grid-item1">1. tofu</div>
    <div class="grid-item2">2. bakłażan</div>
    <div class="grid-item3">3. cebula</div>
    <div class="grid-item4">4. marchewki</div>
    <div class="grid-item5">5. brukiew</div>
    <div class="grid-item6">6. babeczki</div>
    <div class="grid-item7">7. ogórek</div>
    <div class="grid-item8">8. marchewka</div>
    <div class="grid-item9">9. dżem</div>
    <div class="grid-item10">10. bataty</div>
    <div class="grid-item11">11. groch</div>
    <div class="grid-item12">12. fasola</div>
    <div class="grid-item13">13. soczewica</div>
    <div class="grid-item14">14. pomidory</div>
    <div class="grid-item15">15. dynia</div>
    <div class="grid-item16">16. szynka</div>
    <div class="grid-item17">17. pizza</div>
    <div class="grid-item18">18. makaron</div>
    <div class="grid-item19">19. ser</div>
    <div class="grid-item20">20. mleko</div>
</div>
```

Oto kod CSS:

```
.container {
    font-size: 28px;
    font-family: sans-serif;
    display: grid;
    gap: 30px;
    background-color: #ddd;
    grid-template-columns: repeat(4, 1fr);
    grid-auto-rows: 100px;
    grid-auto-flow: row;
}

[class^='grid-item'] {
    outline: 1px #f90 dashed;
    display: grid;
    background-color: goldenrod;
    align-items: center;
    justify-content: center;
}

.grid-item3 {
    grid-column: 2/-1;
}

.grid-item6 {
    grid-row: 3/6;
```

```
    grid-column: 3 / 5;
}

.grid-item17 {
    grid-row: 6 / span 2;
    grid-column: 2/3;
}

.grid-item4 {
    grid-row: 4 / 7;
}
```

Na rysunku 5.8 widać układ, który pojawia się w przeglądarce.



Rysunek 5.8. Arbitralnie określone rozmiary elementów siatki

Wprowadziliśmy tutaj kilka nowych elementów, które po kolei omówimy.

Własność gap

Użyłem własności `gap` w niektórych poprzednich fragmentach kodu, ale nie wyjaśniłem, jak działa. Wybacz! Własność `gap` pozwala na określenie odstępu między torami siatki. W rzeczywistości jest to własność zbiorcza, zastępująca `row-gap` i `column-gap`. Tak samo jak w przypadku, kiedy określasz margines przy użyciu dwóch wartości, pierwsza wartość odnosi się do górnej i dolnej krawędzi (rzędu), a druga do lewej i prawej (kolumny). Pojedyncza wartość, tak jak w powyższym kodzie, odnosi się do obu tych kategorii.

> Niekiedy można się zetknąć z własnością `grid-gap` zamiast `gap`. Jest tak, ponieważ `gap` nosiła właśnie taką nazwę, zanim przeprowadzono jej rewizję, a wiele przeglądarek pierwotnie obsługiwało jedynie `grid-gap`. Jeżeli zależy Ci na jak najszerszej obsłudze kodu, możesz bezpiecznie posłużyć się własnością `grid-gap`.

Wartość repeat

Gdybyś chciał utworzyć siatkę składającą się z 30 jednakowych kolumn, zapisanie wartości `auto` 30 razy byłoby dość uciążliwe: `grid-template-columns: auto auto auto auto auto auto...;` już samo to mnie zmęczyło!

Na szczęście twórcy specyfikacji Grid CSS dali nam do rąk wartość `repeat()`, która pozwala na wygodne określenie właściwości dowolnej liczby elementów. W naszym przykładzie użyliśmy jej do utworzenia czterech kolumn o szerokości `1fr`:

```
repeat(4, 1fr);
```

Pierwsza wartość w nawiasie określa liczbę powtórzeń, a druga szerokość każdego elementu.

Nie martw się, już wkrótce wyjaśnię, czym są jednostki `fr`; na razie wystarczy, abyś wiedział, że możesz z łatwością tworzyć wiele kolumn i rzędów. Chcesz piętnaście kolumn o szerokości 100 pikseli? Wystarczy podać wartość `repeat(15, 100px)`.

Jednostki fr

Jednostka `fr` odnosi się do „elastycznej długości”, a jest skrótem „flex fraction” (ułamek elastycznej przestrzeni). Służy do określania, jaką część dostępnej, wolnej przestrzeni dany element ma zająć, co przypomina w działaniu wartość `flex-grow` flexboksa, którą omówiliśmy w rozdziale 4.

Wprawdzie niczego takiego nie ma w specyfikacji, ale sam w pracy nad układami traktuję `fr` jako jednostkę „wolnego miejsca”. W naszym przykładzie utworzyliśmy cztery kolumny, z czego każda zajmuje po jednej części wolnego miejsca.

Rozmieszczanie elementów w siatce

We wcześniejszym przykładzie umieściliśmy wszystkie elementy siatki w pojedynczych obszarach. Tutaj jednak niektórym elementom przypisaliśmy zakresy kolumn lub rzędów liczbowo.

Rozważmy przykład `grid-item3`:

```
grid-item3 {
    grid-column: 2/-1;
}
```

Własność `grid-column` służy tutaj do ustawienia punktu początkowego na drugiej linii siatki, a końcowego na linii −1. Wartość `-1` na pierwszy rzut oka wydaje się osobliwa, ale to w rzeczywistości element dobrze przemyślanej składni.

Pierwsza liczba określa punkt początkowy, a od wartości wskazującej punkt końcowy oddziela ją ukośnik. Liczby dodatnie wskazują punkt, licząc od początku — lewego końca w naszym przykładzie z kolumnami — podczas gdy liczby ujemne wskazują punkt, licząc od końca — w tym

wypadku od prawego końca. Wartość `-1` oznacza zatem ostatnią linię siatki. Ta przystępna, zwięzła składnia wskazuje zatem: „Zacznij od drugiej linii i dojdź do końca".

W niektórych regułach celowo niedbale stawiałem spacje, aby pokazać Ci, że możesz je równie dobrze pominąć — to zupełnie nieistotne.

Mamy też przykład rozciągania elementów na wiele rzędów. Spójrz na następujący kod:

```
.grid-item4 {
    grid-row: 4 / 7;
}
```

Ta reguła mówi: „Zacznij od czwartej linii rzędów i dojdź do siódmej".

Wartość span

Przyjrzyjmy się teraz regule `.grid-item17`:

```
.grid-item17 {
    grid-row: 6 / span 2;
    grid-column: 2/3;
}
```

Czy widzisz, że wartość `grid-row` wygląda nieco inaczej?

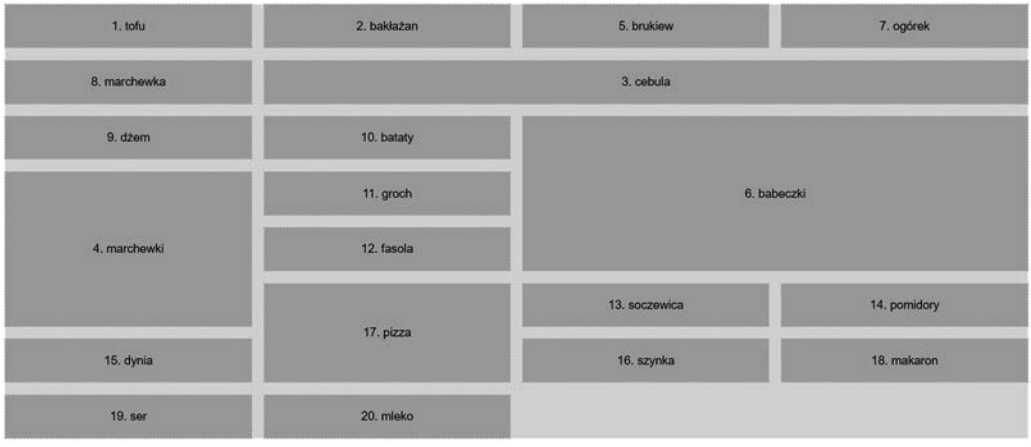
Zamiast określać konkretny początek i koniec przy rozmieszczaniu elementów siatki, możesz wskazać jeden z tych punktów i polecić, aby element rozciągał się od niego na określoną liczbę rzędów lub kolumn do przodu lub do tyłu. W naszym przykładzie element zaczyna się na szóstej linii siatki i rozciąga się na dwa rzędy.

Liczenie do tyłu wydaje mi się mniej intuicyjne, ale może coś ze mną nie tak. Na przykład ten sam efekt wizualny moglibyśmy osiągnąć, zmieniając wartość tej własności na `grid-row: span 2 / 8`. W tym przypadku definiujemy punkt końcowy, więc polecamy, aby element rozciągał się na dwa rzędy wstecz od ósmego rzędu.

Wartość dense

O słowie kluczowym `dense` wspomniałem przy okazji własności `grid-auto-flow`. To świetna okazja, aby pokazać Ci, do czego ono służy. Zmieńmy wartość `grid-auto-flow` na `row dense;`. Rezultat widać na rysunku 5.9.

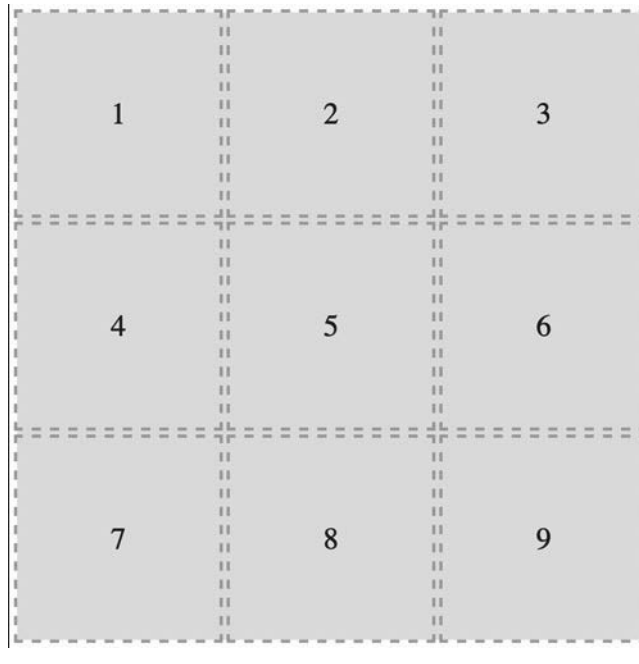
Jak widzisz, puste przestrzenie zniknęły. Do tego właśnie służy `dense`. Choć takie rozwiązanie może się wydawać bardziej estetyczne, ma to swoją cenę. Elementy ponumerowałem po to, aby zwrócić Ci uwagę, że użycie wartości `dense` poleca algorytmowi siatki przesunięcie elementów w wolne miejsca bez oglądania się na ich kolejność w dokumencie źródłowym.



Rysunek 5.9. Słowo kluczowe dense przestawia elementy siatki tak, aby usunąć puste przestrzenie

Nazwane linie siatki

Twórcy stron internetowych mogą używać siatek na różne sposoby. Jeśli np. wolisz pracować ze słowami niż z liczbami, możesz nadać liniom siatki własne nazwy. Rozważmy przykład siatki składającej się z trzech kolumn i trzech rzędów (rysunek 5.10).



Rysunek 5.10. Przestawimy nasze elementy przy użyciu nazwanych linii siatki

Kod tego przykładu znajdziesz w folderze *05-05*.

Oto kod HTML:

```
<div class="my-first-grid">
    <div class="grid-item-1">1</div>
    <div class="grid-item-2">2</div>
    <div class="grid-item-3">3</div>
    <div class="grid-item-4">4</div>
    <div class="grid-item-5">5</div>
    <div class="grid-item-6">6</div>
    <div class="grid-item-7">7</div>
    <div class="grid-item-8">8</div>
    <div class="grid-item-9">9</div>
</div>
```

Siatkę tworzymy przy użyciu poniższej reguły. Zwróć uwagę na wartości podane w nawiasach kwadratowych:

```
.my-first-grid {
    display: inline-grid;
    grid-gap: 10px;
    grid-template-columns: [left-start] 200px [left-end center-start] 200px
[center-end right-start] 200px [right-end];
    grid-template-rows: 200px 200px 200px;
    background-color: #e4e4e4;
}
```

W nawiasach kwadratowych definiujemy nazwy linii siatek. W tym przypadku linii pierwszej kolumny nadaliśmy nazwę `left-start`, a drugiej `left-end`. Zauważ, że środkowej linii siatki przydzieliliśmy dwie nazwy: `left-end` i `center-start`. Można to zrobić, oddzielając nazwy spacją. W tym przypadku ma to sens, ponieważ ta linia jest zarówno końcem lewej kolumny, jak i początkiem środkowej.

Rozwińmy deklarację `grid-template-rows` i podajmy w niej kilka nazw dla linii siatki:

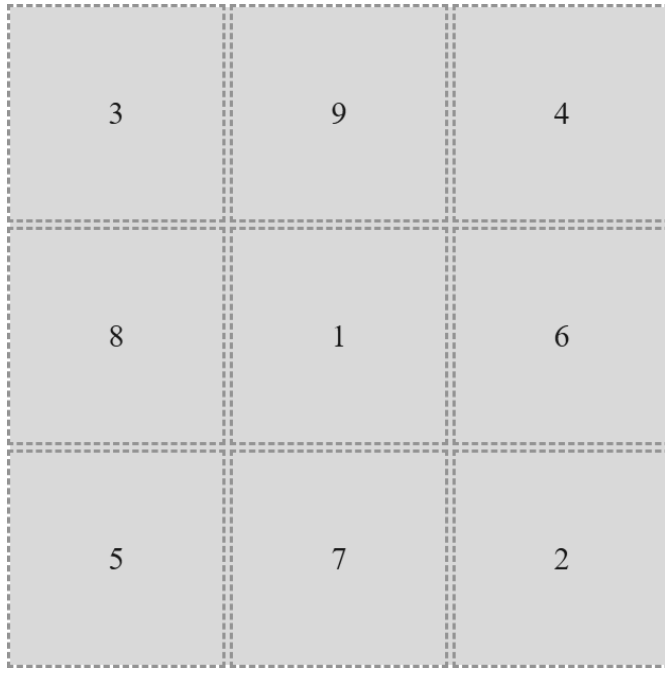
```
grid-template-rows: [top-start] 200px [top-end middle-start] 200px [middle-end
bottom-start] 200px [bottom-end];
```

Oto przykład użycia tych nazw zamiast liczb do rozmieszczenia elementów siatki. Ten fragment kodu odnosi się jedynie do trzech pierwszych elementów, które widać na rysunku 5.11:

```
.grid-item-1 {
    grid-column: center-start / center-end;
    grid-row: middle-start / middle-end;
}

.grid-item-2 {
    grid-column: right-start / right-end;
    grid-row: bottom-start / bottom-end;
}
```

```
.grid-item-3 {
    grid-column: left-start / left-end;
    grid-row: top-start / middle-start;
}
```

W przykładowym kodzie za pomocą tej techniki nadałem każdemu elementowi siatki losowe położenie. Na rysunku 5.11 widać rozmieszczenie trzech powyższych elementów.



Rysunek 5.11. Elementy możesz przemieszczać tak samo łatwo przy użyciu nazwanych linii siatki

W specyfikacji nazwy nadawane liniom siatki określa się mianem „custom ident" (własne identyfikatory). Ponieważ są to zwykłe słowa, należy unikać używania terminologii, która może kolidować ze słowami kluczowymi siatki. Nie nadawaj zatem liniom siatki nazw w rodzaju „dense", „auto-fit", czy „span"!

Grid CSS jest ponadto tak miły, że jeśli dodasz do swoich nazw linii przyrostki `-start` lub `-end`, tak jak w naszym przykładzie, siatka automagicznie (wiem, nie ma takiego słowa) utworzy nazwany obszar siatki. Zaraz, co? Zgadza się, oznacza to, że po nazwaniu linii swojej siatki możesz umieszczać elementy na niej przy użyciu zbiorczej własności `grid-area`. Oto przekształcony kod odpowiadający za rozmieszczenie trzech pierwszych elementów:

```
.grid-item-1 {
    grid-area: middle / center;
}
```

```
.grid-item-2 {
    grid-area: bottom / right;
}

.grid-item-3 {
    grid-area: top / left;
}
```

Trochę za bardzo wybiegłem naprzód, ponieważ wprowadziłem własność `grid-area` bez jakiegokolwiek wyjaśnienia. Przyjrzyjmy się jej teraz.

Własność grid-template-areas

Innym sposobem pracy z siatką jest definiowanie jej obszarów. Przekształćmy odpowiednio poprzedni przykład. W tym celu usuńmy nazwane linie siatki, aby zacząć od nowa z podstawowym arkuszem stylów siatki.

Poniższy kod znajdziesz w folderze *05-06*.

```
.my-first-grid {
    display: inline-grid;
    grid-gap: 10px;
    grid-template-columns: 200px 200px 200px;
    grid-template-rows: 200px 200px 200px;
    background-color: #e4e4e4;
}

[class^='grid-item'] {
    display: grid;
    align-items: center;
    justify-content: center;
    outline: 3px dashed #f90;
    font-size: 30px;
    color: #333;
}
```

Zdefiniujemy teraz obszary siatki, które dodamy do reguły `.my-first-grid`.

```
grid-template-areas:
    'one two three'
    'four five six'
    'seven eight nine';
```

Przy użyciu `grid-temple-areas` rzędy i kolumny definiuje się bardzo łatwo. Rząd definiuje się cudzysłowami (pojedynczymi lub podwójnymi), a poszczególne kolumny oddziela się spacją. Każdy rząd siatki definiuje się w cudzysłowie zawierającym własne identyfikatory.

> Początki obszarów siatki można definiować numerycznie, ale przy wywoływaniu ich nazw konieczne jest poprzedzenie ich znakiem ucieczki. Na przykład jeśli jeden z naszych obszarów nazywa się „9", to odniesienie do niego musi wyglądać następująco: `grid-area: "\39;"`. Moim zdaniem to dość uciążliwe, dlatego proponuję nazywać obszary słownie lub przynajmniej zaczynać nazwy wszystkich własnych identyfikatorów od liter.

Rozmieszczanie elementów przy użyciu własności `grid-area` wygląda zatem następująco:

```
.grid-item-1 {
    grid-area: five;
}

.grid-item-2 {
    grid-area: nine;
}

.grid-item-3 {
    grid-area: one;
}
```

Przyznam, że to nieskomplikowany przykład, ale może sam wymyślisz coś bardziej kreatywnego: układ bloga z nagłówkiem, paskiem bocznym po lewej, obszarem z treścią główną oraz stopką. Potrzebne do tego obszary możesz zdefiniować następująco:

```
grid-template-areas:
    'header header header header header header'
    'side side main main main main'
    'side side footer footer footer footer';
```

> Według specyfikacji dostępnej pod adresem *https://www.w3.org/TR/css-grid-1/#valdef-grid-template-areas-string* białe znaki nie generują tokenów, więc jeśli chcesz, możesz używać tabulatora, aby rozmieścić nazwy obszarów w równych kolumnach.

Przy tworzeniu obszarów siatki wcięcia i nowe linie nie mają znaczenia; gdybyś chciał, mógłbyś definicję każdego rzędu zapisać jako długą, oddzielaną spacjami listę. Dopóki nazwy obszarów są zawarte w cudzysłowie, wszystkie są oddzielone od siebie białymi znakami, a między poszczególnymi cudzysłowami znajdują się spacje, wszystko jest w porządku.

Zastosowanie nabytych umiejętności

W ramach ćwiczenia spójrz na zrzut ekranowy z witryny *https://rwd.education* (rysunek 5.12).

Rysunek 5.12. Czy potrafisz odtworzyć ten układ przy użyciu nabytej do tej pory wiedzy o siatce?

W podfolderze *Start* folderu z kodem z tego rozdziału zobaczysz, że obecnie te sekcje znajdują się jedna pod drugą. Spróbuj zmodyfikować ten kod przy użyciu CSS Grid. Możesz się np. zastanowić, jak w prosty sposób sprawić, by jedna lub dwie sekcje były widoczne, kiedy ilość miejsca jest ograniczona, a w przypadku dostatecznej ilości przestrzeni wyświetlał się cały układ.

> Istnieje już projekt roboczy modułu CSS Grid Layout poziomu 2. Największą korzyścią, jaką ma przynieść, jest możliwość tworzenia podsiatek: siatek zagnieżdżonych w siatkach, które mogą dziedziczyć wymiary torów swoich rodziców. Aktualną specyfikację znajdziesz tutaj: *https://www.w3.org/TR/css-grid-2/*.

Przejdźmy teraz do jeszcze bardziej zaawansowanych technik korzystania z siatki.

Wartości auto-fit i auto-fill

Wartości `auto-fit` i `auto-fill` są słowami kluczowymi używanymi do opisywania powtórzeń w siatce.

Sądzę, że te dwa słowa kluczowe niemal zawsze się ze sobą mylą ze względu na podobne nazwy — tak jak `cover` i `contain` przy określaniu wymiarów obrazu tła (powrócimy do nich w rozdziale 7.). Z tego względu często muszę sprawdzać, które jest którym. Dla obopólnej korzyści zajmiemy się teraz ustaleniem tego, co każde z nich robi i dlaczego warto z nich korzystać.

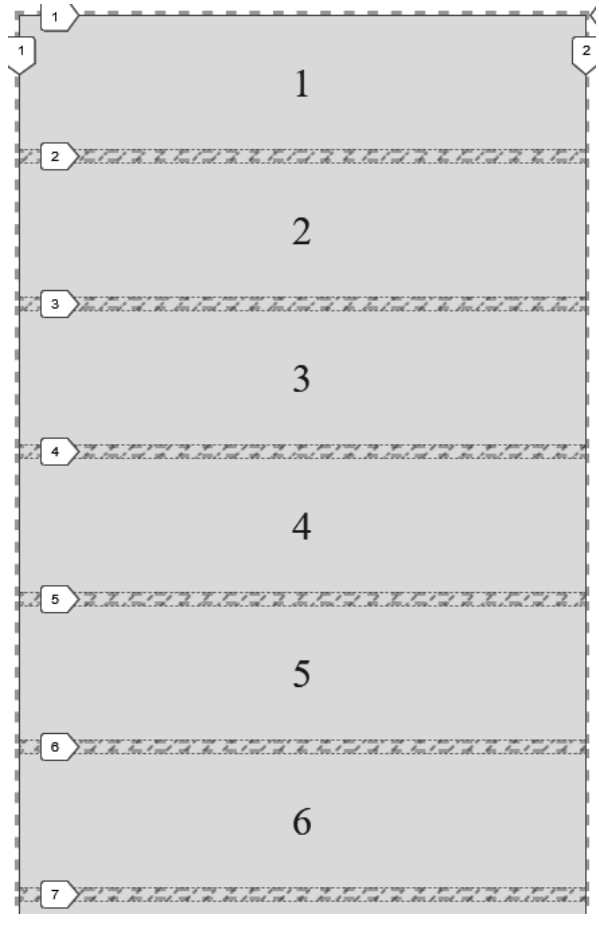
Zacznijmy od tej drugiej kwestii, skoro dotyczy obu słów kluczowych. Czy uwierzyłbyś, że za pomocą `auto-fill` lub `auto-fit` możesz utworzyć w pełni responsywną siatkę, która dodaje lub usuwa kolumny zależnie od wielkości dostępnego obszaru roboczego, a to wszystko bez jakichkolwiek zapytań medialnych?

Kusząca perspektywa, czyż nie?

Rozważmy siatkę z dziewięcioma kolumnami, każdą szeroką na 300 pikseli. W tym przypadku zacznę od pokazania Ci rozwiązania naszego problemu:

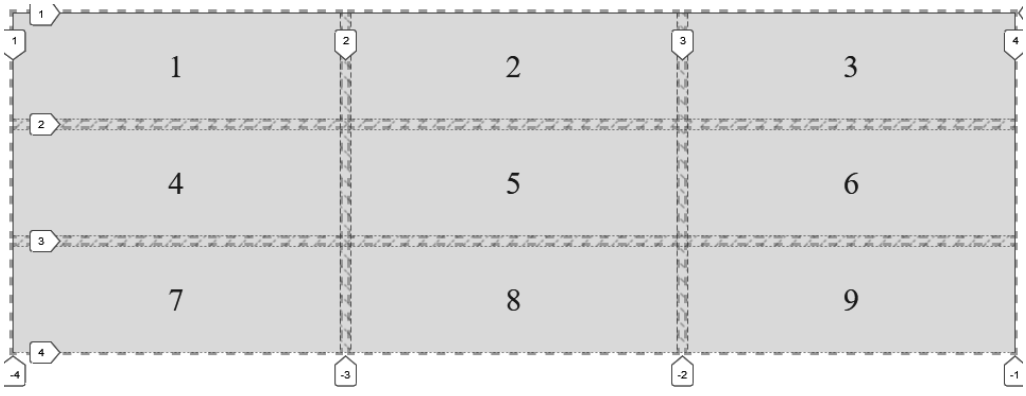
```
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));
```

Na rysunku 5.13 widać, jaki rezultat otrzymujemy w przypadku mniejszego obszaru roboczego.



Rysunek 5.13. Jeden wiersz kodu modułu siatki pozwala uzyskać układ na urządzenia mobilne…

Na rysunku 5.14 widać z kolei tę samą stronę, lecz na szerszym ekranie.



Rysunek 5.14. …a także układ właściwy dla szerszych obszarów roboczych!

Całkiem praktyczne, nie sądzisz?

Zobaczmy, jak te czary działają.

Tak jak poprzednio własności `grid-template-columns` używamy do zdefiniowania kolumn naszej siatki. Używamy funkcji `repeat()` do określenia powtarzającego się wzorca rozmieszczania kolumn, lecz zamiast podawać wartość liczbową, decydujemy się użyć słowa kluczowego `auto-fit`. Moglibyśmy tutaj także użyć wartości `auto-fill`, ale zaraz wrócimy do omówienia, czym się różnią. Na razie wskazaliśmy przeglądarce, aby cyklicznie tworzyła automatycznie dopasowywane kolumny, a ich szerokość określiliśmy przy użyciu funkcji `minmax`.

Funkcja minmax()

Jeżeli nie miałeś dotąd styczności z CSS Grid, prawdopodobnie nie korzystałeś też z `minmax()`. Ta funkcja CSS umożliwia wskazanie przeglądarce zakresu. Przy jej użyciu określasz wielkość minimalną i maksymalną, a przeglądarka oblicza zawartą w tym przedziale wartość na podstawie dostępnej przestrzeni. W naszym przykładzie podajemy funkcji `minmax()` minimalną wielkość 300 pikseli i maksymalną równą 1 `fr` (pamiętaj, że pomocne może być odczytywanie `fr` jako „wolne miejsce”).

> Jeśli wskażesz funkcji `minmax()` wielkość maksymalną mniejszą od minimalnej, zostanie ona zignorowana, a funkcja zwróci wartość minimalną jako wynik.

Po wprowadzeniu tych ustawień siatka będzie „automatycznie dopasowywać” kolumny o minimalnej szerokości 300 pikseli, które nie są jednak szersze od swojej zawartości i udziału w pozostałej przestrzeni równego `1fr`. W praktyce tworzy to responsywny układ, w którym siatka sama zmienia wymiary zależnie od szerokości obszaru roboczego.

Aby zobaczyć różnicę między `auto-fit` a `auto-fill`, podajmy 100 pikseli jako wartość minimalną:

```
grid-template-columns: repeat(auto-fit, minmax(100px, 1fr));
```

W szerokich obszarach roboczych uzyskujemy układ widoczny na rysunku 5.15.

Rysunek 5.15. Zastosowanie wartości auto-fit sprawia, że treść wypełnia dostępną przestrzeń

Zauważ, że kolumny rozciągają się na całą szerokość strony. Zmieńmy teraz wartość na `auto-fill`:

```
grid-template-columns: repeat(auto-fill, minmax(100px, 1fr));
```

Uzyskujemy efekt widoczny na rysunku 5.16.

Rysunek 5.16. Zastosowanie wartości auto-fill sprawia, że pusta przestrzeń zostaje wypełniona niewidocznymi kolumnami

Czy widzisz wolną przestrzeń na końcu? Co tu się stało?

Różnica sprowadza się do tego, czy wolne kolumny się składają, czy nie.

Kiedy przeglądarka tworzy siatkę przy użyciu dowolnej z tych wartości, początkowo rozkłada kolumny jednakowo. W przypadku `auto-fit` wszystkie kolumny, które po rozmieszczeniu treści pozostały puste, zostają złożone, zostawiając po sobie miejsce, które zostaje równo rozdzielone między wszystkimi elementami w rzędzie. W naszym przykładzie, ponieważ każdy element ma także maksymalną wielkość równą `1fr`, każdy element zajmuje równą część tej przestrzeni. W wyniku tego uzyskujemy kolumny, które rozciągają się na całą szerokość dostępnego obszaru.

W przypadku `auto-fill`, jeśli po rozłożeniu wszystkich elementów (w tym przykładzie szerokich na 100 pikseli) pozostaje jakieś wolne miejsce, pozostałe wolne kolumny nie składają się, lecz nadal są obecne w układzie, przez co nie zwalniają miejsca dla innych elementów. W wyniku tego na końcu rzędu otrzymujemy pustą przestrzeń.

Wybór słowa kluczowego jest zależny od okoliczności; pamiętaj jedynie, że możesz uzyskać dowolny z tych rezultatów.

> Na niektórych zrzutach ekranu widać wskaźniki linii siatki, które są wyświetlane przez narzędzia programistyczne Firefoksa. Kiedy piszę te słowa na początku 2020 r., Firefox moim zdaniem dysponuje najlepszymi narzędziami programistycznymi do pracy z CSS Grid.

Składnia zbiorcza

Z CSS Grid można korzystać przy użyciu różnych składni zbiorczych: jednej stosunkowo prostej, drugiej nieco mniej. Pierwsza z nich, `grid-template`, prawdopodobnie okaże się najbardziej przydatna.

Choć składnie zbiorcze bywają wspaniałe, zachęcam Cię do pisania siatek po jednej własności naraz, przynajmniej na samym początku. Kiedy będziesz już to robił z taką wprawą, że definiowanie każdej własności i wartości pojedynczo stanie się przykrą rutyną, potraktuj to jako znak, że warto poświęcić czas na naukę wariantu zbiorczego.

Skoro już udzieliłem Ci tej dobrej rady, przyjrzyjmy się owym dwóm składniom zbiorczym.

Składnia grid-template

Składnia ta umożliwia definiowanie `grid-template-rows`, `grid-template-columns` i `grid-template--areas` w jednym wierszu.

W przypadku siatki z dwoma 200-pikselowymi rzędami i trzema 300-pikselowymi kolumnami kod wyglądałby następująco:

```
grid-template: 200px 200px / 300px 300px 300px;
```

Gdybyśmy skorzystali z funkcji `repeat`, wyglądałby on tak:

```
grid-template: repeat(2, 200px) / repeat(3, 300px);
```

Człon poprzedzający ukośnik definiuje rzędy, a ten za ukośnikiem kolumny. Jeśli chcesz, możesz też wprowadzić zdefiniowane `grid-template-areas`:

```
grid-template:
    [rows-top] 'a a a' 200px
    'b b b' 200px [rows-bottom]
    / 300px 300px 300px;
```

Powyższy kod interpretowany jest przez przeglądarkę następująco:

```
grid-template-rows: [rows-top] 200px 200px [rows-bottom];
grid-template-columns: 300px 300px 300px;
grid-template-areas: 'a a a' 'b b b';
```

Odnoszę wrażenie, że kiedy zaczyna się podawać jako wartości nazwy zdefiniowanych obszarów, ma się już do czynienia z nadmiernym mętlikiem. Jednak niektórzy uwielbiają tę składnię, więc też powinieneś wiedzieć, że można z niej skorzystać.

Teraz przejdziemy na jeszcze wyższy poziom i zajmiemy się składnią `grid`.

Składnia grid

Druga składnia, `grid`, pozwala na zdefiniowanie całej siatki w jednym wierszu.

Przy użyciu tej własności możesz zdefiniować własności kontrolujące siatkę jawną: `grid-template-rows`, `grid-template-columns` i `grid-template-areas`, a także własności kontrolujące zachowanie siatki niejawnej: `grid-auto-rows`, `grid-auto-columns` i `grid-auto-flow`.

Ważnym konceptem, jakiego należy się trzymać przy korzystaniu z własności zbiorczej `grid`, jest to, że siatka może się rozrastać w sposób niejawny jedynie albo w poziomie, albo w pionie, ale nie w obu kierunkach jednocześnie. Na pierwszy rzut oka może się to wydawać dziwne, ale właściwie jak siatka, która dodawałaby i rzędy, i kolumny, mogłaby rozkładać dodatkowe elementy? W jaki sposób miałaby decydować, czy w danym przypadku dodać rząd, czy kolumnę?

Mając powyższe na uwadze, możemy zająć się niuansami składni `grid`.

Składnia zbiorcza `grid` nie jest dla ludzi o słabych nerwach, więc nie zniechęcaj się, jeśli kilka razy zostaniesz przez nią sponiewierany. Sam uważam się za osobę dość rozeznaną w CSS (wiem, tego raczej ode mnie oczekujesz), ale potrzebowałem nie minut, lecz godzin, żeby poczuć się pewnie z tym, jak ta składnia działa.

Mam nadzieję, że jesteś za pan brat z notacją BNF, bo w specyfikacji ta własność jest opisana następująco:

```
<'grid-template-rows'> / [ auto-flow && dense? ] <'grid-autocolumns'>?
↪[ auto-flow && dense? ] <'grid-auto-rows'>? / <'gridtemplate-columns'>
```

Proste, prawda?

Oczywiście żartuję. Obecnie, kiedy próbuję się zapoznać ze specyfikacją CSS, kieruję się wskazówkami przedstawionymi w tym artykule: *https://www.smashingmagazine.com/2016/05/understanding-the-css-property-value-syntax/*.

Po ponownym jego przeczytaniu dołożę teraz wszelkich starań, żeby przekształcić ten fragment specyfikacji w coś bardziej przystępnego dla zwykłego człowieka. Przede wszystkim własność `grid` może przyjmować jako wartość jedną z trzech różnych składni.

Składnia grid — opcja pierwsza

Korzystamy z tej samej wartości, której użyłbyś z własnością `grid-template`. Oto siatka z dwoma rzędami wysokimi na 100 pikseli i trzema kolumnami szerokimi na 200 pikseli:

```
grid: 100px 100px / 200px 200px 200px;
```

Podobnie jak we wcześniejszych przykładach `grid-template` możesz także skorzystać z samodzielnie zdefiniowanych obszarów `grid-template-areas`.

Składnia grid — opcja druga

W tym przypadku używa się zbioru wysokości jawnie zdefiniowanych rzędów, oddzielonych ukośnikiem od definicji obsługi niejawnie zdefiniowanych kolumn. Może to być `auto-flow` określająca `grid-auto-rows`, a także definiowanie `grid-auto-flow` przy użyciu `dense`. Można też podać wartość szerokości kolumn, aby zamiast tego zdefiniować `grid-template-columns`.

Uff, to całkiem sporo do obliczenia. Przyjrzyjmy się kilku przykładom.

Aby zatem uzyskać siatkę z dwoma jawnie zdefiniowanymi 100-pikselowymi rzędami i dowolną liczbą jawnie zdefiniowanych kolumn o szerokości 75 pikseli, użylibyśmy następującego kodu:

```
grid: 100px 100px / repeat(auto-fill, 75px);
```

W przypadku takiej siatki nadliczbowe elementy zostaną rozmieszczone w zdefiniowanych niejawnie rzędach, w których uzyskają domyślny rozmiar `auto`.

Przeglądarka przekształci tę własność zbiorczą w następujące deklaracje:

```
grid-template-rows: 100px 100px;
grid-template-columns: repeat(auto-fill, 75px);
grid-template-areas: none;
grid-auto-flow: initial;
grid-auto-rows: initial;
grid-auto-columns: initial;
```

Wypróbujmy inną deklarację. Załóżmy, że chcemy utworzyć siatkę z jednym, wysokim na 100 pikseli rzędem, lecz składającym się z dowolnej liczby kolumn, być może schodzących za krawędź kontenera:

```
grid: 100px / auto-flow;
```

Powyższa deklaracja jest interpretowana następująco:

```
grid-template-rows: 100px;
grid-template-columns: initial;
grid-template-areas: initial;
grid-auto-flow: column;
grid-auto-rows: initial;
grid-auto-columns: initial;
```

> Warto też wiedzieć, że użycie składni zbiorczej `grid` powoduje wyzerowanie wszystkich wartości, na których operuje, do ich początkowego stanu. Widać to po obliczonych wartościach stylów w narzędziach programistycznych przeglądarki.

Składnia grid — opcja trzecia

Ostatnia opcja jest w praktyce odwrotnością opcji drugiej. W tym wypadku `grid-auto-flow` nadaje się wartość w celu obsłużenia niejawnie definiowanych rzędów (z opcjonalnym słowem kluczowym `dense`) oraz opcjonalną wartość `grid-auto-rows`, określającą wielkość rzędów. Po ukośniku nadajemy wartość własności `grid-template-columns`. Takie wartości sprawiają, że siatka w razie potrzeby rozkłada elementy w rzędach zamiast w kolumnach, tak jak przy poprzedniej składni. Poniżej zapoznasz się z kilkoma przykładami.

A gdyby tak utworzyć siatkę składającą się z dowolnej liczby rzędów wysokich na 100 pikseli i pięciu kolumn o szerokości `1fr` każda?

```
grid: auto-flow 100px / repeat(5, 1fr);
```

Przeglądarka interpretuje to następująco:

```
grid-template-rows: initial;
grid-template-columns: repeat(5, 1fr);
grid-template-areas: initial;
grid-auto-flow: row;
grid-auto-rows: 100px;
grid-auto-columns: initial;
```

Albo siatkę, która tworzy pojedynczą kolumnę z taką liczbą rzędów wysokich na 100 pikseli, jaka jest potrzebna, aby pomieścić treść?

```
grid: auto-flow 100px / auto;
```

Przeglądarka interpretuje to tak:

```
grid-template-rows: initial;
grid-template-columns: auto;
grid-template-areas: initial;
grid-auto-flow: row;
grid-auto-rows: 100px;
grid-auto-columns: initial;
```

Jak widzisz, składnia zbiorcza `grid` daje niezwykle bogate możliwości, lecz nie jest zbyt zrozumiała. Niektórzy zdecydowanie wolą z niej korzystać, dla innych jest źródłem frustracji. Nie ma obiektywnie poprawnych preferencji — jest tylko to, co Ci odpowiada lub nie.

Podsumowanie

Jeżeli tworzenie stron internetowych jest dla Ciebie stosunkowo nowym zajęciem, nauka korzystania z CSS Grid będzie niemal ułatwiona — masz świeży, chłonny umysł początkującego. Natomiast dla ludzi, którzy od lat tworzyli układy przy użyciu innych technik, oduczenie się wszystkiego, co wiedzą o tworzeniu układów w CSS, może sprawiać niemałą trudność.

Po przeczytaniu tego rozdziału powinieneś orientować się, jakie możliwości daje siatka i jak możesz je wykorzystać.

Ponadto, jeśli udało Ci się uporać z ćwiczeniami, możesz sobie pogratulować. Za pierwszym podejściem do korzystania z siatki trzeba rozważyć wiele spraw. Jeśli udało Ci się osiągnąć zadowalający rezultat, oznacza to, że poradziłeś sobie znakomicie.

W tym miejscu jeszcze raz powtórzę, że korzystanie z CSS Grid jest początkowo skomplikowane. Wiąże się z wieloma możliwościami, ale też wymaga zapoznania się z nową terminologią i koncepcjami. Licz się z tym, że będziesz musiał zaliczyć kilka podejść. Obiecuję jednak, że kiedy już wypracujesz w tym zakresie kompetencję, włożony przez Ciebie wysiłek przyniesie szczodre korzyści.

Ostatnie dwa rozdziały traktowały o dość rozległych zagadnieniach: tworzeniu układów przy użyciu najnowszych technik i posługiwaniu się obrazami responsywnymi. Kolejny rozdział będzie ukierunkowany bardziej szczegółowo. CSS pozwala na wykorzystanie wielu ciekawych sztuczek i technik — pod tym względem jest workiem pełnym miłych niespodzianek. W rozdziale 6. zapoznasz się z tematyką selektorów, typografii i trybów koloru.

Współczesny HTML i CSS? Mogą więcej, niż myślisz!

Jeszcze dziesięć lat temu responsywność strony internetowej była traktowana jako ciekawostka. Dziś jest powszechnym standardem. Znane od dawna klasyczne technologie HTML i CSS wciąż zachowują świeżość i wyjątkową przydatność w tworzeniu nowoczesnych stron internetowych. Społeczności skupione wokół tych narzędzi starannie dbają o ich nieustanny rozwój, dzięki czemu za ich pomocą nadal można osiągać imponujące efekty i spełniać coraz wyższe standardy. Oznacza to, że każdy szanujący się programista front-endu powinien na bieżąco śledzić nowości i uczyć się korzystania z sukcesywnie pojawiających się funkcjonalności HTML i CSS.

Ta książka jest kolejnym, uaktualnionym i uzupełnionym wydaniem lubianego podręcznika projektowania responsywnych stron internetowych. Omówiono w niej wszystkie nowości i ulepszenia z dziedziny projektowania responsywnych stron internetowych, w tym zapewnianie lepszej dostępności, fonty zmienne czy kontrola przewijania strony. Szczegółowo opisano moduł CSS Grid i mechanizm tworzenia układów Flexbox. Zaprezentowano wiele praktycznych informacji o SVG, wymogach dostępności, efektach w CSS, definiowaniu przejść, transformacji i animacji oraz włączaniu do kodu zapytań medialnych. Znakomitym uzupełnieniem treści są autorskie wskazówki i uwagi do programowania front-endu.

Najważniejsze zagadnienia:

- dostosowywanie arkuszy stylów do potrzeb różnych urządzeń
- pisanie przejrzystego, szybkiego i bogatego semantycznie kodu HTML
- grafiki w formacie SVG w projektach responsywnych
- najnowsze możliwości CSS: własności użytkownika, fonty zmienne i siatka
- weryfikacja danych w HTML i inne przydatne funkcje formularzy
- filtry, cienie, animacje i inne efekty wzbogacające interfejs

Ben Frain

jest programistą front-endów. Strony internetowe tworzy od 1996 roku. Nim odkrył swoje powołanie do pisania kodu, był niedocenionym aktorem telewizyjnym i twórcą niedocenianych scenariuszy. Uwielbia swoją pracę i grę w halową piłkę nożną (o ile dopisuje mu kondycja).

Helion

helion.pl

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

ISBN 978-83-283-7419-5

9 788328 374195

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 69,00 zł

Packt