БИБЛИОТЕКА ПРОГРАММИСТА

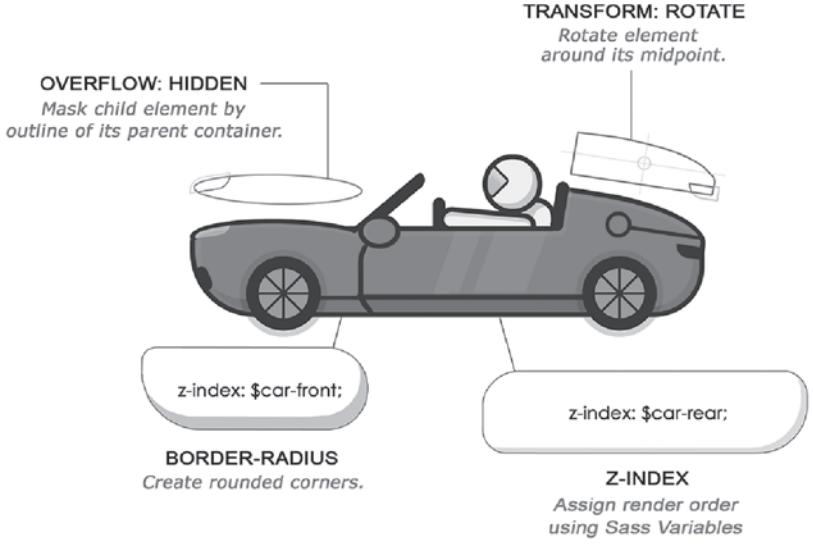
Грег Сидельников

НАГЛЯДНЫЙ CSS

CSS

VISUAL DICTIONARY

FLEX GRID TRANSFORMS ANIMATION SCSS



Learning Curve Books™

БИБЛИОТЕКА ПРОГРАММИСТА

НАГЛЯДНЫЙ CSS

Грег Сидельников

Санкт-Петербург • Москва • Минск

2021

ББК 32.988-02-018
УДК 004.738.5
С34

Сидельников Грег

С34 Наглядный CSS. — СПб.: Питер, 2021. — 224 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1618-8

На 1 июня 2018 года CSS содержал 415 уникальных свойств, относящихся к объекту style в любом элементе браузера Chrome. Сколько свойств доступно в вашем браузере на сегодняшний день? Наверняка уже почти шесть сотен. Наиболее важные из них мы и рассмотрим.

Грег Сидельников упорядочил свойства по основной категории (положение, размерность, макеты, CSS-анимация и т. д.) и визуализировал их работу.

Вместо бесконечных томов документации — две с половиной сотни иллюстраций помогут вам разобраться во всех тонкостях работы CSS. Эта книга станет вашим настольным справочником, позволяя мгновенно перевести пожелания заказчика и собственное видение в компьютерный код!

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.988-02-018
УДК 004.738.5

ISBN 978-1983065637 англ.
ISBN 978-5-4461-1618-8

Оглавление

Предисловие

На создание книги, которую вы держите в руках (или читаете в электронном виде), ушло несколько месяцев. Действительно, *«Наглядный CSS»* — это результат любви и большого труда. Я написал эту книгу, чтобы пополнить и усовершенствовать ваши знания и навыки для работы с CSS. CSS (Cascading Style Sheets — каскадные таблицы стилей) — это язык стилей, отвечающий за визуальное представление HTML-документов.

Я искренне надеюсь, что в дальнейшем книга станет вашим верным помощником.

1 Свойства и значения CSS

На 1 июня 2018 года CSS содержал **415** уникальных свойств, относящихся к объекту `style` в любом элементе браузера Chrome. По состоянию на 21 декабря 2018 года насчитывалось **522** уникальных свойства. Всего за семь месяцев в Chrome была добавлена поддержка более 100 новых свойств. Это будет происходить постоянно, так как спецификация CSS продолжает развиваться.

Сколько свойств доступно в вашем браузере на сегодняшний день? Вы можете проверить это самостоятельно с помощью простого фрагмента кода JavaScript.

```
001 // Создание нового HTML-элемента
002 let element = document.createElement("div");
003
004 let p = 0; // Создание счетчика
005 for (index in element.style)
006     p++;
007
008 // Выводит 522 в Chrome по состоянию на 21 декабря 2018 года
009 console.log( p );
```

Для просмотра всех доступных в вашем браузере свойств CSS запустите данный код JavaScript (быстрее всего проверить CSS и JavaScript можно с помощью codepen.io). Результаты могут различаться для разных браузеров и версий.

В процессе создания книги все свойства были распечатаны и упорядочены по основным категориям (*положение*, *размерность*, *разметка*, *CSS-анимация* и т. д.). Затем для каждого свойства, которое каким-то важным образом отображает или изменяет визуальный вывод, была создана схема с кратким описанием названия и значения.

Здесь не описаны редко используемые свойства CSS (или те, которые на момент написания книги не имели полной поддержки основных браузеров). Они бы только все запутали.

Мы сосредоточимся только на свойствах, которые в настоящее время широко применяются веб-дизайнерами и разработчиками. Много усилий было уделено созданию схем **grid-** и **flex-верстки**. Помимо этого, я включил краткое руководство по **SASS/SCSS**, но выбрал только наиболее важные функции, о которых вы должны знать.

1.1. Внешнее размещение

Код CSS можно сохранить в отдельном внешнем файле (например, `style.css`) и включить с помощью HTML-элемента `link`.

Исходный код файла `style.css`:

```
001 body p
002 {
003     background: white;
004     color: black;
005     font-family: Arial, sans-serif;
006     font-size: 16px;
007     line-height: 1.58;
008     text-rendering: optimizeLegibility;
009     -webkit-font-smoothing: antialiased;
010 }
```

Пример ссылки на внешний CSS-файл:

```
001 <html>
002     <head>
003         <title>Добро пожаловать на сайт.</title>
004         <link rel = "stylesheet"
005               type = "text/css"
006               href = "style.css" />
007      </head>
008      <body>
009          <p>CSS-стили записаны в файле style.css
010             и применяются к содержимому этой страницы.</p>
011      </body>
012 </html>
```

1.2. Внутреннее размещение

Вы можете ввести CSS-код непосредственно в HTML-документ между двумя тегами элемента `style`:

```
<html>
  <head>
    <style type = "text/css">
        body p
        {
            background: white;
            color: black;
            font-family: Arial, sans-serif;
            font-size: 16px;
            line-height: 1.58;
            text-rendering: optimizeLegibility;
            -webkit-font-smoothing: antialiased;
        }
    </style>
  </head>

  <body>
      <p>CSS-стили внутри элемента style
         и применяются к этому абзацу
         в HTML-коде веб-страницы</p>
  </body>
</html>
```

1.3. Строковое размещение

Строковое размещение CSS-кода с использованием атрибута `style` в элементе HTML:

```
<html>
  <head></head>
  <body style = "font-family: Arial;">
      <p>При выводе в браузере этот абзац
         наследует форматирование шрифтом Arial
         из строкового стиля
         в родительском элементе.</p>
  </body>
</html>
```

1.4. Селекторы

Теперь мы знаем, где CSS-код находится в HTML-документе. Но прежде, чем мы начнем рассматривать каждое свойство по отдельности, полезно ознакомиться с грамматикой языка CSS — правилами синтаксиса для определения свойств и значений.

Наиболее распространенный селектор — само *имя HTML-элемента* (напомню, что за редким исключением HTML-элемент состоит из двух тегов, открывающего и закрывающего).

Использование имени тега приведет к выделению всех элементов данного типа. Выберем элемент `body` по его имени:

```
001 body { /* Сюда помещаются свойства CSS */ }
```

На первый взгляд, поскольку в HTML-документе есть только один элемент `body`, это единственное, что будет выбрано.

Однако из-за каскадной специфики CSS любое свойство, которое мы берем в скобки, также будет применяться ко всем его потомкам (дочерним элементам, содержащимся в элементе `body`, даже если мы не станем явно указывать их стиль).

Это пустой селектор. Он выбирает элемент `body`, но пока не назначает ему никаких свойств.

Ниже приведены несколько других примеров выбора объектов по имени их HTML-элемента. Это самые распространенные приемы.

```
001 /* Выбираем все элементы абзаца, p */
002 p { }
003
004 /* Выбираем все элементы div */
005 div { }
006
007 /* Выбираем все элементы p, только если они находятся
008    в элементах div */
009 div p { }
```

Ваши стили CSS будут заключены в `{ ...`*`эти`*`... }` скобки.

Инструкция CSS состоит из *селектора* и пары `свойство: значение;`. Несколько свойств должны быть разделены *точкой с запятой*. Начнем с одного свойства, просто чтобы посмотреть, как выглядит синтаксис CSS-свойства:

```
001 <div id = "box">контент<div>
```

В CSS идентификатор указывается в виде символа *хештега* `#`:

```
001 #box { свойство: значение; }
```

Используйте идентификаторы (`id`) для маркировки элементов всякий раз при наличии *уникального* контейнера.

Не форматируйте каждый отдельный HTML-элемент с помощью идентификаторов, задействуйте их для именования глобальных родительских элементов или для более значимых элементов (например, тех, которые необходимо часто обновлять из-за изменения содержимого).

Что, если мы хотим выбрать несколько элементов одновременно?

```
001 <ul>
002     <li class = "элемент">1<li>
003     <li class = "элемент">2<li>
004     <li class = "элемент">3<li>
005 </li>
```

Аналогично атрибут класса (`class`) обозначается селектором точки (`.`):

```
001 .item { line-height: 1.80; }
```

В данном примере точка используется для выбора нескольких элементов, имеющих одно и то же имя класса. Так свойству `line-height` (высота строки) присваивается значение `1.50` (что примерно соответствует 150 % высоты шрифта).

Специальные правила CSS: селектор `:root` применяет их ко *всем* HTML-элементам. Вы можете использовать `:root`, чтобы установить значения CSS *по умолчанию* для всего документа.

Установим Arial в качестве шрифта по умолчанию для всего документа или sans-serif, если шрифт Arial недоступен; вы можете указать столько шрифтов, сколько пожелаете:

```
001 :root { font-family: Arial, sans-serif; }
```

Селектор `:root` также часто используется для *глобального* хранения CSS-переменных.

Создайте CSS-переменную с именем `--red-color` и присвойте ей значение цвета `red`:

```
001 :root { --red-color: red; }
```

Учтите, что все имена переменных CSS должны начинаться с двойного дефиса `--`.

Теперь вы можете использовать CSS-переменную `--red-color` в качестве значения в стандартных селекторах CSS:

```
001 div { color: var(--red-color) ; }
```

Мы изучили, как селектор `:root` позволяет сохранить CSS-переменные, и узнали, что он также может сбросить до значений по умолчанию *все форматирование документа*.

Селектор звездочка (`*`) выполняет те же функции.

Можно использовать селектор * для достижения эффекта, вызываемого применением `:root`. Единственное отличие состоит в том, что селектор * нацелен абсолютно на все элементы в документе, а `:root` — только на контейнер документа без его дочерних элементов:

```
001 * { font-family: Arial, sans-serif; }
```

Несмотря на то что добавление селектора * дает тот же эффект, менее целесообразно использовать его для применения стилей ко всему документу (вместо этого задействуйте `:root`).

Лучше всего селектор * подходит для пакетного выбора «всех элементов» в пределах определенного родительского элемента.

Селектор `#parent *` может использоваться для выбора всех потомков родительского элемента независимо от их типа:

```
001 <div id = "parent">
002     <div>A</div>
003     <div>B</div>
004     <ul>
005         <li>l</li>
006         <li>2</li>
007     </ul>
008     <p>Текст.</p>
009 </div>
```

Продолжая экспериментировать с селекторами, вы заметите, что можно выбирать одни и те же HTML-элементы, используя разные *комбинации* селекторов.

Например, все следующие комбинации выбирают один и тот же набор элементов (все потомки родительского элемента, без учета самого предка). Селектор `#parent *` может использоваться для выбора всех потомков родительского элемента независимо от их типа:

```
001 /* Выбираем все дочерние элементы #parent */
002 #parent * { color: blue; }
003
004 /* Объединяем несколько селекторов, используя запятую */
005 #parent div,
006 #parent ul,
007 #parent p { color: blue; }
008
009 /* Применяем псевдоселекторы :nth-child */
010 #parent nth-child(1),
011 #parent nth-child(2),
012 #parent nth-child(3),
013 #parent nth-child(4) { color: blue; }
```

Конечно, наличие возможности не означает, что так нужно делать. Это лишь пример.

Наиболее целесообразным решением в данном случае является селектор `#parent *`. Но каждый проект, сайт или приложение требуют разметки, уникальной по своей структуре и назначению.

Поначалу создание селекторов может показаться простой задачей. Но это до тех пор, пока вы не углубитесь в более сложные примеры пользовательского интерфейса. С каждым разом ваш CSS-код будет становиться все сложнее и сложнее.

Сложность CSS-кода тесно связана со структурой самого HTML-документа. Поэтому даже некоторые из самых «умных» селекторов часто могут «пересекаться» с селекторами, созданными впоследствии, вызывая конфликты. Изучение CSS — настоящее искусство. Ваши навыки создания CSS-селекторов будут улучшаться только в процессе регулярной практики!

При работе над реальным проектом и ввиду постоянного усложнения макета приложений вы не сможете даже предположить, как часто конкретное CSS-свойство не будет работать нужным образом.

Когда пропущен определенный сценарий и допущена ошибка, разработчики часто используют ключевое слово `!Important`, чтобы быстро исправить проблему.

Вы можете переопределить любой стиль CSS, добавив ключевое слово `!Important` в конец кода CSS.

```
001 /* Выбираем все дочерние элементы #parent */
002 #parent * { color: blue; }
003
004 /* Выбираем только div в #parent и меняем цвет на красный */
005 #parent div { color: red; }
006
007 /* Проверяем, что все div во всем документе зеленого цвета */
008 div { color: green !important; }
```

Весьма заманчиво использовать ключевое слово `!Important` для принудительной установки стиля CSS. Но обычно такая практика считается порочной, поскольку игнорируется каскадная логика таблиц стилей!

ПРЕДОСТЕРЕЖЕНИЕ

Директиву !Important лучше вообще не использовать. Даже если вам покажется, что вы решаете проблему, применение директивы может значительно усложнить обслуживание вашего кода CSS.

Идеальный вариант, когда CSS-селекторы максимально простые и эффективные. Однако такой баланс не всегда легко сохранить. Я обычно начинаю с набросков своего CSS-кода на бумаге. Потратив немного больше времени на обдумывание структуры приложения и написание заметок, вы быстрее создадите наиболее оптимальные селекторы.

1.5. Взаимосвязь между свойствами и значениями

Не все CSS-свойства одинаковы. В зависимости от типа свойства значение может быть **мерой пространства**, заданного в *пикселах*, единицах *pt*, *em* или *fr*, **цветом**, указанным в виде имени (*red*, *blue*, *black* и т. д.), шестнадцатеричного значения (*#0F0* или *#00FF00...*) или rgb (*r*, *g*, *b*).

В других случаях значение уникально для конкретного свойства, и его нельзя использовать с любым другим свойством. Например, CSS-свойство `transform` может принимать значение, указанное с помощью ключевого слова `rotate`.

В данном случае принимается угол в градусах — CSS требует добавления букв `deg` к числовому значению градуса.

```
001 /* поворот этого элемента на 45 градусов по часовой стрелке */
002
003 #box {
004     transform: rotate(45deg);
005 }
```

Однако это не единственный способ указать угол. CSS предлагает еще три *типа* единиц, специально предназначенных для указания угла поворота: `grad`, `rad` и `turn`.

```
001 /* 200 градиан (град)*/
002 transform: rotate(200grad);
003
004 /* 1,4 радиан */
005 transform: rotate(1.4rad);
006
007 /* 0,5 оборота или 180 градусов (1 оборот = 360 градусов) */
008 transform: rotate(0.5turn);
```

В данном случае мы используем `grad` (градианы), `rad` (радианы) и `turn` (повороты) в качестве альтернативного способа задания угла поворота HTML-элемента.

Альтернативные способы указания *значений* не редкость для многих других CSS-свойств. Например, `#F00`, `#FF0000`, `red`, `rgb(255, 255, 255)` и `rgba(255, 255, 255, 1.0)` задают один и тот же цвет.

1.6. Комментарии в CSS-коде

Для создания комментариев в коде CSS поддерживается только синтаксис блочных комментариев.

Это делается путем вставки блока текста с использованием символов `/* комментарий */`.

```
/* Установить белый цвет шрифта, используя
   шестнадцатеричное значение */
p { color: #FFFFFF; }

/* Установить белый цвет шрифта, используя сокращенное
   шестнадцатеричное значение */
p { color: #FFF; }

/* Установить белый цвет шрифта, используя название цвета */
p { color: white; }

/* Установить белый цвет шрифта, используя значение RGB */
p { color: rgb(255,255,255); }

/* Создать переменную CSS --white-color
   (обратите внимание на двойной дефис) */
:root { --white-color: rgba(255, 255, 255, 1.0); }

/* Установить белый цвет шрифта,
   используя CSS-переменную */
p { color: var(--white-color); }
```

Обратите внимание, как один и тот же цвет свойства может принимать различные типы значений. При использовании переменных CSS имя переменной предваряет двойной дефис (`--`).

Вы также можете закомментировать раздел CSS-кода целиком. Далее показано временное отключение блока кода CSS для тестирования нового кода или будущей ссылки и т. д.:

```
001 /* Временно отключить данный блок CSS
002     content:"hello";
003     border: 1px solid gray;
004     color: tfFFFFFF;
005     line-height: 48px;
006     padding: 32px;
007 */
```

CSS не поддерживает *// встроенные комментарии*, или, скорее, они не оказывают влияния на CSS-интерпретатор браузера.

1.7. Оформление стилей

В CSS имеется множество свойств, связанных с габаритами и размерами (`left`, `top`, `width`, `height` и др.). Было бы излишним перечислять их все. Поэтому далее в примерах я буду использовать слово *свойство*.

Для указания значения служит шаблон *свойство*: *значение*. Такая комбинация позволит установить фоновые изображения, цвет и другие основные свойства HTML-элементов.

В качестве альтернативы можете использовать шаблон *свойство*: *значение значение значение* для установки нескольких значений одному свойству. Это и есть *сокращенная форма записи*. Значения разделяются пробелом.

Без сокращенной формы записи вы бы указали каждую часть свойства в отдельной строке.

```
001 /* Фон */
002 background-color:    black;
003 background-image:    url("image.jpg");
004 background-position: center;
005 background-repeat:   no-repeat;
006
007 /* Сокращенная форма записи, только одна строка кода! */
008 background: black url("image.jpg") center no-repeat;
```

За все эти годы в CSS произошли значительные изменения. Прежде чем изучать визуальные схемы, описывающие каждое свойство, необходимо понять, как CSS интерпретирует шаблоны свойств и значений.

Большинство свойств используют следующие шаблоны:

```
/* Самая распространенная форма */
свойство: значение;

/* Значения, разделенные пробелом */
свойство: значение значение значение;

/* Значения, разделенные запятой */
свойство: значение, значение, значение;
```

Свойства, связанные с размером, могут быть рассчитаны с помощью ключевого слова `calc`:

```
/* Вычисление */
свойство: calc(значениепх);

/* Вычисление значений в % и px – ок. */
свойство: calc(значение% - значениепх);

/* Вычисление значений в % и % – ок. */
свойство: calc(значение% - значение%);

/* Сложение px и px – ок. */
свойство: calc(значениепх + значениепх);
```

Вычитание, умножение и деление выполняются по той же схеме. Только не пробуйте делить на значение в пикселах.

```
/* Вычитание px из px – ок. */
свойство: calc(значениепх - значениепх);

/* Умножение px на число – ок. */
свойство: calc(значениепх * число});

/* Деление px на число – ок. */
свойство: calc(значениепх / число});

/* Деление числа на px – ошибка. */
свойство: calc(число / значениепх});
```

Последний пример выдаст ошибку. Используя ключевое слово `calc`, вы не можете разделить число на значение, указанное в пикселах (px).

1.8. CSS-переменные

Вы можете задействовать CSS-переменные, чтобы избежать многократного определения одних и тех же значений в различных CSS-селекторах. Имена переменных CSS всегда начинаются с двух дефисов.

Определить CSS-переменную в глобальной области видимости можно с помощью селектора `:root`. Здесь элемент используется только в качестве заполнителя, в реальной же работе он будет заменен допустимым именем HTML-элемента:

```
001 /* Определяем переменную --default-color */
002 :root { --default-color: yellow; }
003
004 /* Определяем переменную --variable-name */
005 :root { --variable-name: 100px; }
006
007 /* Устанавливаем значение переменной
008    цвета фона --default-color */
009 элемент { background-color: var(--default-color); }
010
011 /* Устанавливаем ширину 100px */
012 элемент { width: var(--variable-name); }
```

Локальные переменные

Вы можете создавать локальные переменные, содержащиеся только в определенном родительском элементе. Таким образом, они не проникают в глобальную область видимости и не получают другие определения переменных, потенциально объявленных с тем же именем.

Хорошая идея — сохранять определения переменных, скрытых для области, в которой они используются. Обычно это эффективно для любого языка программирования, например JavaScript, но годится и для CSS:

```
001 // Определение локальной переменной
002 .notifications { --notification-color: blue; }
003
004 // Локализация переменной для дочерних элементов
005 .notifications div {
006   color: var(--notification-color);
007   border: 1px solid var(--notification-color);
008 }
```

1.9. Метаязык SASS

Syntactically Awesome Stylesheet, или *SASS*, — это препроцессор CSS, добавляющий новые функции, которые в настоящее время недоступны в стандартной версии CSS.

SASS — расширенный набор стандартных CSS. То есть все, что работает в CSS, будет работать в SASS.

Мы больше не используем прежний синтаксис SASS с файлами в формате `.SASS`. Вместо этого применяется формат `.scss` — новая (и улучшенная) версия SASS.

SCSS рекомендуется для опытных специалистов CSS. Возможно, есть кто-то, способный оценить всю красоту использования цикла `for` в качестве директивы стиля CSS.

Обратите внимание: с 10 декабря 2018 года SASS/SCSS не встраивается в браузеры. Для включения такой функции на своем веб-сервере вам необходимо установить компилятор SASS из командной строки.

Если хотите начать экспериментировать с SASS, то зайдите на сайт www.codepen.io, где сможете легко начать использовать интерпретацию SASS без предварительной настройки. *CodePen* — это среда разработки для дизайнеров и программистов.

Имена переменных SASS определяются начальным символом `$`, так же как в языке PHP!

```
001 $font: Helvetica, sans-serif;
002 $dark-gray: #333;
003
004 body {
005   font: 16px $font;
006   color: $dark-gray;
007 }
```

Что можно сделать с помощью SASS?

```
001 $a: #E50C5E;
002 $b: #E16A2E;
003
004 .mixing-colors {
005     background-color: mix($a, $b, 30%);
006 }
```

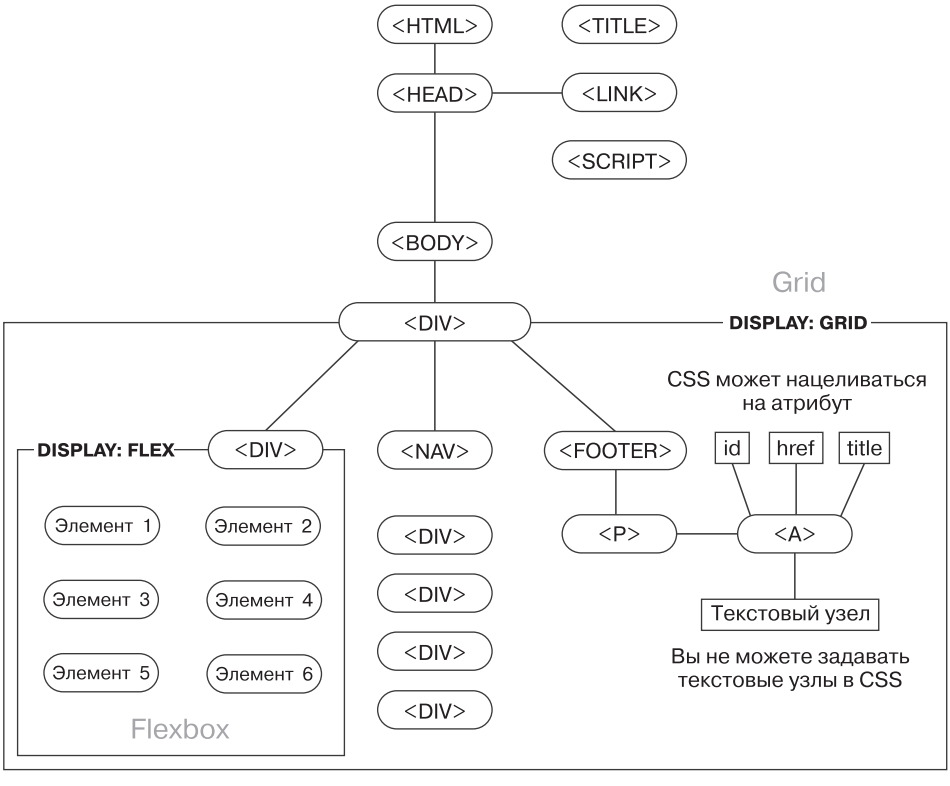
В данном случае SASS использовался для смешивания двух цветов, определенных в переменных SASS `$a` и `$b`.

Я предлагаю вам продолжить изучение SASS/SCSS самостоятельно, но только после того, как вы освоите стандартный CSS, описанный в книге.

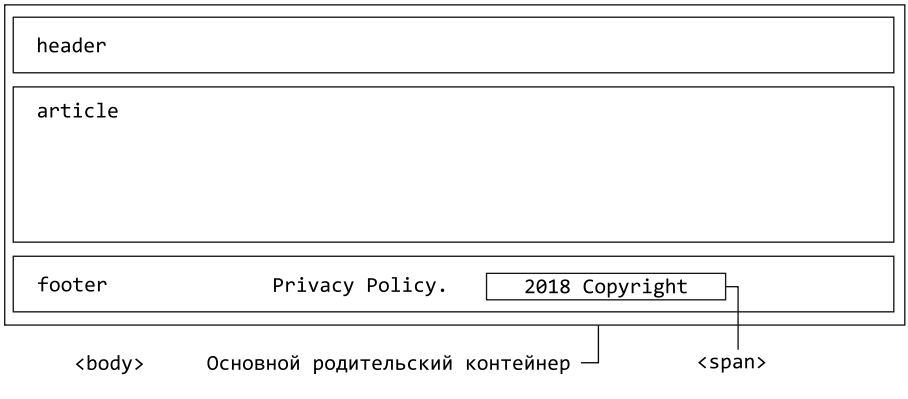
1.10. Суть каскадных таблиц стилей

Каскадные таблицы стилей названы так по определенной причине. Представьте водопад с водой, разбивающейся внизу о камни. Каждый камень, на который упала вода, становится влажным. Точно так же каждый стиль CSS наследует стили, уже примененные к его родительскому элементу HTML.

Посмотрите на следующую схему. Стили CSS буквально «сползают» по иерархии DOM, представляющей древовидную структуру вашего сайта. Язык CSS (с помощью некоторых селекторов) позволяет контролировать этот обычно странный процесс.



Чтобы представить основную концепцию CSS, рассмотрим простую структуру сайта.



На схеме представлено несколько элементов, вложенных в основной контейнер сайта. CSS, подобно пинцету, помогает выбирать элементы, к которым мы хотим применить определенный стиль

Если вы примените черный фон к элементу `body`, то все вложенные элементы в нем автоматически унаследуют черный фон:

```
001 body { background: black color: white; };
```

Данный стиль будет «каскадно» перемещаться по родительской иерархии, заставляя все следующие HTML-элементы наследовать белый текст на черном фоне.

Базовая структура HTML:

```
001 <body>
002   <header>
003     <p>Шапка сайта</p>
004   </header>
005   <article>
006     <p>Основной контент/p>
007   </article>
008   <footer>
009     <p>Privacy Policy. <span>&copy;
010        2019 Copyright</span></p>
011   </footer>
012 </body>
```

В том случае, если вы хотите оформить подвал (footer) и выделить слова Privacy Policy красным цветом, а 2018 Copyright — зеленым, можете расширить каскадный принцип, добавив следующие CSS-команды.

```
001 body        { background: black; color: white; }
002 footer      { color: red; }
003 footer span { color: green; }
```

Обратите внимание: между словами `footer` и `span` есть пробел. В CSS он выступает актуальным символом селектора CSS. Это означает следующее: «найти в ранее указанном теге» (в данном примере `footer`).

1.11. Селекторы CSS

Основные CSS-селекторы:

```
001 /* Выбираем один элемент с идентификатором id */
002 #id { }
003
004 /* Выбираем все элементы с классом class */
005 .class { }
006
007 /* Выбираем все элементы с классом class1, создающие иерархию
008    с другим родительским элементом с идентификатором parent */
009 #parent .class1 { }
```

1.12. Лояльность CSS

Поскольку разработка CSS велась для сред, в которых загрузка полной копии сайта не всегда гарантирована, он является одним из самых лояльных языков, похожих на HTML. Если вы допустите ошибки или по какой-то причине страница полностью не загрузится, то CSS-код будет постепенно утрачивать свойства настолько, насколько возможно. Это значит, что вы все еще можете использовать *// встроенные комментарии*, но, вероятно, не стоит этого делать.

1.13. Распространенные комбинации

Рассмотрим некоторые из наиболее распространенных комбинаций CSS-свойств и значений:

```
001 /* Устанавливаем белый цвет шрифта */
002 color: #FFFFFF;
003
004 /* Устанавливаем черный цвет фона */
005 background-color: #000000;
006
007 /* Создаем вокруг элемента синюю границу толщиной 1 пиксел */
008 border: 1px solid blue;
```

```
001 /* Устанавливаем белый цвет шрифта */
002 font-family: Arial, sans-serif;
003
004 /* Устанавливаем размер шрифта 16px */
005 font-size: 16px;
006
007 /* Добавляем отступы размером 32px */
008 padding: 32px;
009
010 /* Добавляем вокруг области контента отступ размером 16px */
011 margin: 16px;
```

1.14. Сокращенные нотации

Назначим три различных свойства, способствующих появлению фонового изображения HTML-элемента:

```
001 background-color: #000000;
002 background-image: url("image.jpg");
003 background-repeat: no-repeat;
004 background-position: left top;
005 background-size: cover;
006 background-atachment: fixed;
```

То же самое можно сделать, используя одно сокращенное свойство `background`, разделяя значения пробелом:

```
background: background-color background-image background-repeat;
```

Остальные комбинации настроек фона рассматриваются в главе 14.

```
001 background: #000000 url("image.jpg") left top no-repeat fixed;
```

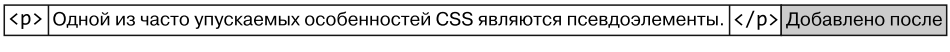
Сокращения также применимы к различным свойствам grid- и flex-верстки.

2 Псевдоэлементы

Псевдоэлементы начинаются с двойного двоеточия `::`. В данном контексте *псевдо* означает, что они не ссылаются на явные элементы DOM, вручную добавленные в документ HTML, например элемент выделения текста.

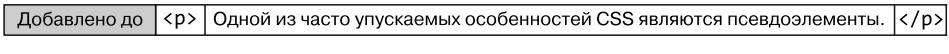
2.1. ::after

```
p::after { content: "Добавлено после"; }
```



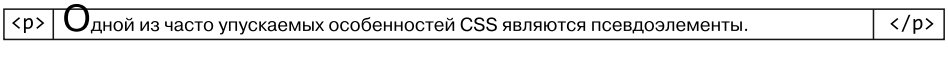
2.2. ::before

```
p::before { content: "Добавлено до"; }
```



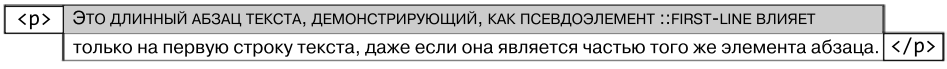
2.3. ::first-letter

```
p::first-letter { font-size: 200%; }
```



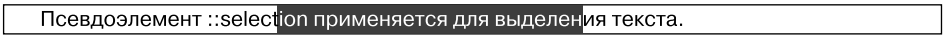
2.4. ::first-line

```
p::first-line { text-transform: uppercase; }
```



2.5. ::selection

```
::selection { background: black; color: white; caret-color: blue; }
```



2.6. ::slotted(*)

Псевдоселектор `slotted` работает только в контексте HTML-элемента `template` для выбора элементов `slot`.

`::slotted(*)` или `::slotted(`*`имя-элемента`*`)`

```
<template>
  <div>
  <slot name = "animal"></slot>
    <ul>
      <li><slot name = "kind">Кот</slot></li>
      <li><slot name = "name">Феликс</slot></li>
    </ul>
  </div>
</template>
```

3 Псевдоселекторы

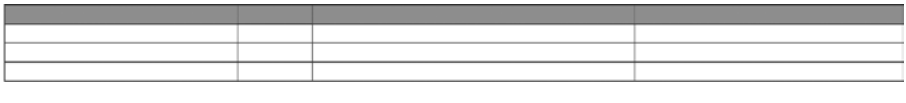
В CSS *псевдоселектор* — это любой селектор, который начинается с символа двоеточия (`:`) и обычно добавляется в конец имени другого элемента — часто родительского контейнера. Псевдоселекторы также известны как *псевдоклассы*.

Псевдоселекторы `:first-child` и `:last-child` используются для выбора самого первого или самого последнего элемента из списка потомков в родительском элементе.

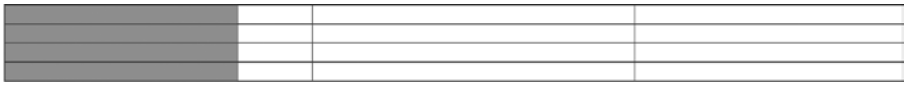
Псевдоселектор `:nth-child` служит для выбора серии элементов, принадлежащих строке или столбцу в списке элементов или даже в таблице HTML.

Далее рассмотрим несколько случаев, демонстрирующих применение псевдоселекторов. Они эффективны при использовании вместе с другими селекторами элементов. Увидев, как псевдоселекторы влияют на таблицу HTML, легко понять их принцип работы, поскольку таблица имеет дочерние элементы, охватывающие оба измерения (строки × столбцы).

Вы можете использовать код `table tr:first-child` для выбора всех элементов в первой строке:



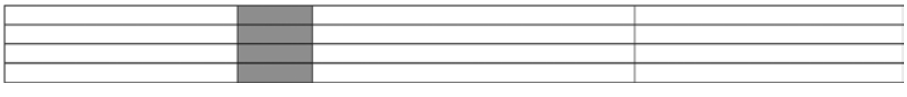
Выберите первый столбец с помощью селектора `table td:first-child`:



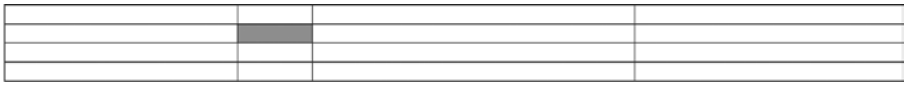
Обратите внимание: между `td` и `:first-child` нет пробела. Это важно, поскольку `td :first-child` (с пробелом) — совершенно другой селектор. Изменение несущественно, однако результаты разные, так как ваш результат будет эквивалентен `td *:first-child`.

Помните, что символ пробела является селектором иерархии элементов? Примеры ниже комбинируют псевдоселекторы с `tr` и `td` для выбора определенного столбца или строки.

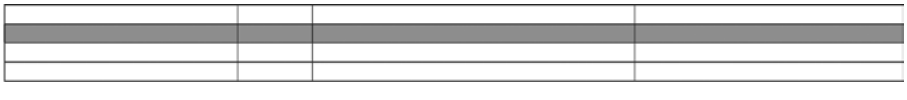
```
table tr td:nth-child(2)
```



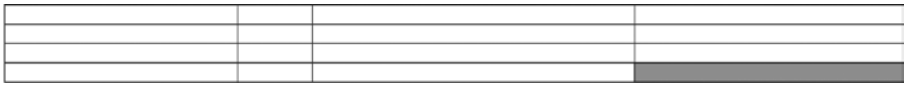
```
table tr:nth-child(2) td:nth-child(2)
```



```
table tr:nth-child(2)
```



```
table tr:last-child td:last-child
```



Те же самые правила `nth-child` применяются и ко всем другим *вложенным группам элементов*, таким как, например, `ul` и `li`, и к любой другой произвольной комбинации «родитель/потомок».

Обратите внимание: сам символ пробела тоже является частью селектора. Это поможет вам детализировать иерархию родительских элементов.

3.1. :link

`:link` похож на `a[href]`.

`<a href = "http://www.google.com/">`	Текст ссылки	`</a>`

`:link` не выбирает элементы `href-less`.

`<a> href-less </a>`

3.2. :visited

`:visited` выбирает посещенные ссылки в текущем браузере.

`<a href = "http://www.google.com/">`	Посещенная ссылка	`</a>`

3.3. :hover

`:hovered` выбирает элемент ссылки, на которую наведен указатель мыши.

`<a href = "http://www.google.com/">`	Ссылка, на которую установлен указатель мыши	`</a>`

3.4. :active

`:active` выбирает активную или нажатую ссылку.

3.5. :focus

`:focus` выбирает элементы с текущим фокусом, включая ссылки, элементы ввода и текстовые элементы.

```
<input type = "text" />
```

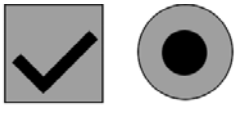
```
<textarea>
</textarea>
```

3.6. :enabled

`:enabled` позволяет активизировать включенные элементы или навести на них фокус. Элемент также может быть отключен, и тогда его невозможно активизировать или навести на него фокус.

3.7. :disabled

`:disabled` выбирает элемент, который нельзя активизировать или на который не получится навести фокус (например, флажок или переключатель).



`:checked` — флажок или переключатель

3.8. :default

`input:default` выбирает элемент по умолчанию (например, флажок или переключатель).


```
<form>
<input type = "radio" name = "answer" value – "YES" checked>ДА</br>
<input type = "radio" name = "answer" value – "NO">НЕТ</br>
<input type = "radio" name = "answer" value – "MAYBE">ВОЗМОЖНО</br>
</form>
```

3.9. :indeterminate

`:indeterminate` выбирает флажок или переключатель, которым не назначено состояние по умолчанию.

3.10. :required

`:required` выбирает элемент ввода с обязательным атрибутом.

```
<input type = "text"  required  />
```

3.11. :optional

`:optional` выбирает элемент ввода без обязательного атрибута.

```
<input type = "text"            />
```

3.12. :read-only

`:read-only` и `read-write` выбирают элементы с атрибутами `readonly` и `disabled`.

`<input type = "text"`	`disabled`	`readonly`	`/>`

3.13. :root

`:root` выбирает корневой элемент DOM (`html`).

`<html>` `</html>`

3.14. :only-of-type

`li:only-of-type`

`<div>`				
`<ul>`				
	`<li>`	`<span>`	Содержимое `</span>`	`</li>`
	`<li>`	`<span>`	Содержимое `</span>`	`</li>`
	`<li>`	`<span>`	Содержимое `</span>`	`</li>`
`</ul>`				
`</div>`				

3.15. :first-of-type

`div ul li:first-of-type`

`<div>`				
`<ul>`				
	`<li>`	`<span>`	Содержимое `</span>`	`</li>`
	`<li>`	`<span>`	Содержимое `</span>`	`</li>`
	`<li>`	`<span>`	Содержимое `</span>`	`</li>`
`</ul>`				
`</div>`				

3.16. :nth-of-type()

`li:nth-of-type(2)`

			Содержимое	
			Содержимое	
			Содержимое	

3.17. :last-of-type

`div ul li:last-of-type`

			Содержимое	
			Содержимое	
			Содержимое	

3.18. :nth-child()

`li:nth-child(1)`

			Содержимое	
			Содержимое	
			Содержимое	

3.19. :nth-last-child()

```
span:nth-last-child(1)
```

			Содержимое		
			Содержимое		
			Содержимое		

3.20. :nth-child(odd)

```
span:nth-child(odd)
```

	Содержимое	
	Содержимое	
	Содержимое	
	Содержимое	

3.21. :nth-child(even)

```
span:nth-child(even)
```

	Содержимое	
	Содержимое	
	Содержимое	
	Содержимое	

3.22. :not()

```
not(.excluded)
```

tr#first td	td.excluded
td.excluded	td.default
td.excluded	td.excluded
td.default	td.excluded

3.23. :empty

```
p::first-line { text-transform: uppercase; }
```

 ЭТО ДЛИННЫЙ АБЗАЦ ТЕКСТА, ДЕМОНСТРИРУЮЩИЙ, КАК ПСЕВДОЭЛЕМЕНТ ::FIRST-LINE
влияет только на первую строку текста, даже если она — часть того же элемента абзаца.

3.24. Вложенные псевдоселекторы

```
p::first-letter { font-size: 200%; }
```

 Псевдоселекторы могут быть связаны.

3.25. :dir(rtl) и :dir(ltr)

```
dir(rtl) или :dir(ltr)
```

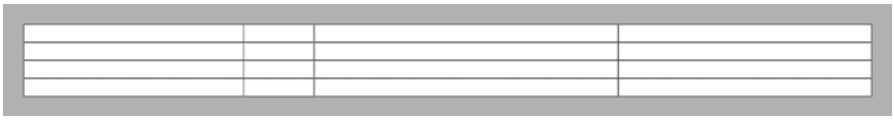
```
<div dir = "rtl">Справа налево</div>
<div dir = "ltr">Слева направо</div>
<div dir = "auto">בארץ טאָכטעס</div>
```

3.26. :only-child

`:only-child`

Что, если вам нужно выбрать абсолютно все элементы на странице или внутри какого-либо родительского элемента? Нет проблем! Селектор звездочка (`*`) выделяет все элементы в родительском элементе. В этом случае используется селектор `table *`:

Разница между селектором звездочка (`*`) и `:root` заключается в том, что последний выбирает только основной контейнер DOM без дочерних элементов:

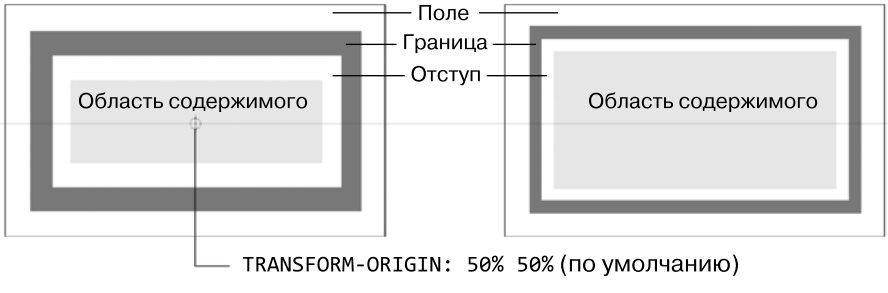


За все эти годы язык CSS очень изменился. В одной из его последних спецификаций проводится четкое различие между псевдоселекторами (известными как псевдоклассы) и псевдоэлементами, начинающимися с двойного двоеточия `::`.

Псевдоселекторы/классы обычно выбирают существующий элемент в DOM, тогда как псевдоэлементы обычно ссылаются на элементы, которые не указаны непосредственно. Например, вы можете изменить цвет фона выделенного текста, добавить содержимое к предполагаемым элементам `::before` и `::after` и др.

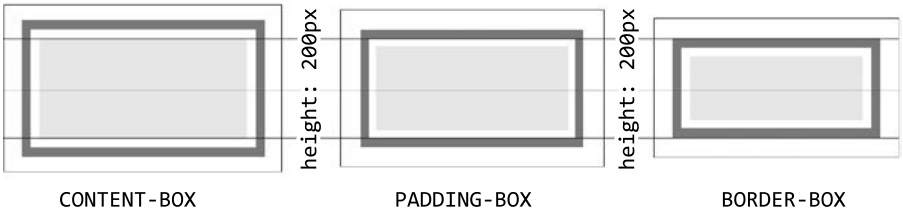
4 Блочная модель CSS

Блочная модель CSS — фундаментальная структура HTML-элемента. Модель состоит из области содержимого с тремя дополнительными слоями пространства вокруг него: *отступами*, *границами* и *полями*.



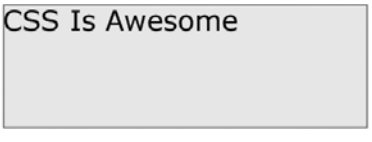
Стандартная блочная модель CSS включает в себя поле, границу, отступы и область содержимого (контента) (см. рисунок, *справа*). Если вы планируете вращать HTML-элемент с помощью свойства `transform`, то вращение будет происходить вокруг центра элемента, поскольку по умолчанию оно составляет `50% 50%`. Изменение его на `0 0` приведет к сбросу точки вращения в верхний левый угол элемента

Наиболее важная особенность блочной модели состоит в том, что по умолчанию ее свойству `box-sizing` присвоено значение `content-box`. Это работает для текстового контента, но я думаю, что данный вариант блокировки элементов в целом немного неудачный, поскольку означает изменение физических размеров области блокировки после добавления отступов, границ или полей. Вот почему в CSS-сетке по умолчанию реализуется модель `border-box`.



Обратите внимание: значение `200px` свойства `height` элемента не меняется, но его физические размеры изменяются в зависимости от `box-sizing: [content-box | padding-box | border-box]`.

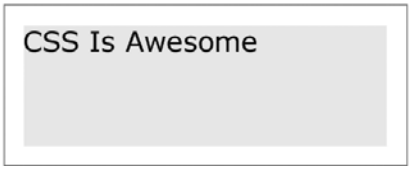
Отсутствует область содержимого `margin-box`, так как поля по определению окружают ее.



Граница влияет на исходный размер элемента, если по умолчанию задано значение content-box

```
border:     1px solid gray;
width:      300px;
height:     100px;
background: #eee;
box-sizing: content-box;
```

Здесь значения свойств `width` (ширина) и `height` (высота) увеличились на 2 пиксела с каждой стороны, так как при использовании модели `content-box` по умолчанию для каждой из четырех сторон была добавлена граница толщиной `1px`.

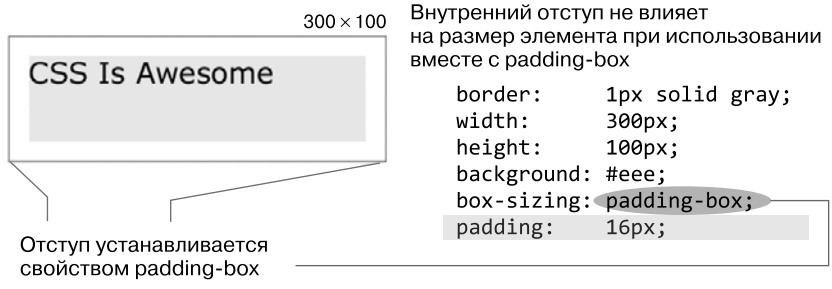


Граница влияет на исходный размер элемента, если по умолчанию задано значение content-box

```
border:     1px solid gray;
width:      300px;
height:     100px;
background: #eee;
box-sizing: content-box;
padding:    16px;
```

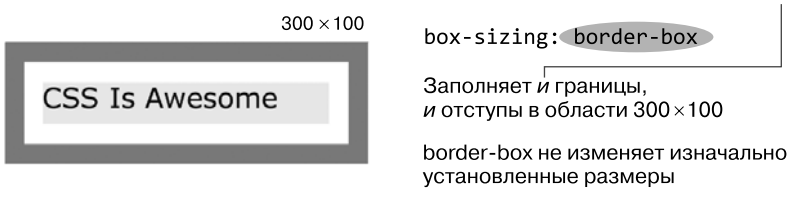
При наличии границ и отступов фактическая физическая ширина становится `334px` × `134px`. Это на 34 пиксела больше, чем исходные размеры (1 пиксел × 2 + 16 пикселов × 2 = 34 пиксела).

Значение `padding-box` определяет отступ как часть содержимого. Теперь исходные размеры сохраняются, но содержимое еще включает отступ:



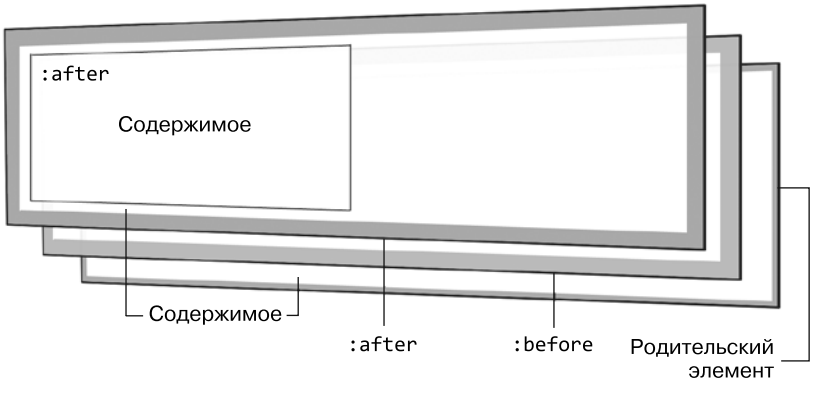
Далее мы перезаписываем исходное значение `border: 1px solid gray` на `border: 16px`, и вместе с `padding: 16px` исходная ширина и высота элемента теперь дополняются 32 пикселами с каждой стороны, прибавляя всего 64 пиксела для каждого размера элемента:

Использование `border-box` инвертирует `border` (границы) и `padding` (отступы), сохраняя исходную `width` (ширину) и `height` (высоту) элемента. Данная опция полезна, когда необходимо убедиться, что элемент сохранит идеальные размеры в пикселах, независимо от величины его границы или внутреннего отступа:

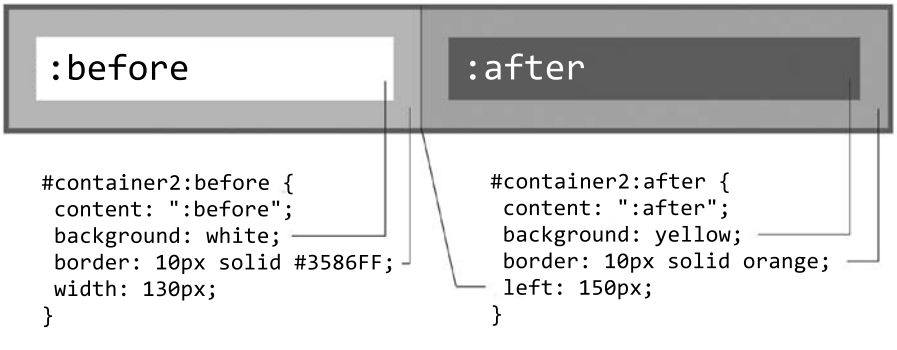


В CSS нет `margin-box`, поскольку внешние отступы по определению всегда относятся к пространству, окружающему содержимое:

HTML-элемент сложнее, чем кажется на первый взгляд:

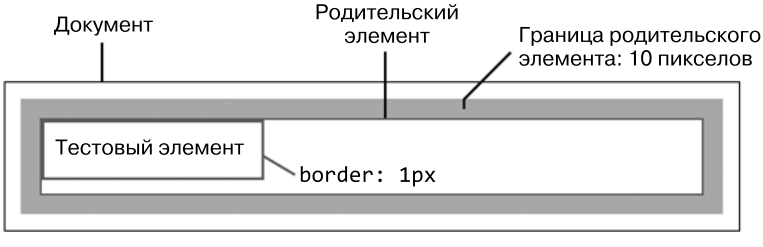


Элементы `:before` и `:after` являются частью одного HTML-элемента. Вы даже можете применить к ним `position: absolute` и организовать их вокруг без создания каких-либо новых элементов!



5 Позиционирование

5.1. Тестовый элемент



Обратите внимание: на самом деле здесь три элемента. Во-первых, сам документ. Но теоретически это может быть `html`, или `body`, или любой другой родительский контейнер. Фактические стили будут применены к тестовому элементу в данном родительском контейнере. Данный образец в качестве примера будет использоваться в главе 6, касающейся позиции элемента.

Позиционирование элементов в CSS может зависеть от свойств родительского контейнера. Для представления различных вариантов данная конкретная настройка будет полезна без отображения полного сайта или макета приложения.

Доступно пять типов позиционирования: `static` (статичное) (по умолчанию), `relative` (относительное), `absolute` (абсолютное), `fixed` (фиксированное) и `sticky` («липкое»). Мы рассмотрим их на протяжении всей этой главы.

По умолчанию для всех элементов используется статичное позиционирование:

Относительное позиционирование практически такое же, как и статичное:

5.2. Статичное и относительное позиционирование

По умолчанию свойство `position` установлено в `static`, то есть элементы отображаются в том порядке, в котором они были указаны в вашем HTML-документе, в соответствии с нормальным потоком HTML-страницы.

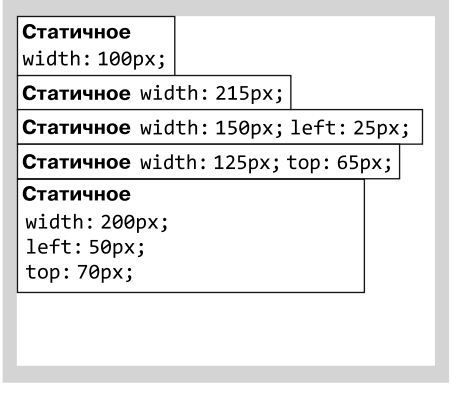
На статично позиционированные элементы не влияют свойства `top`, `left`, `right` и `bottom`.

Чтобы понять разницу, создадим несколько основных стилей CSS:

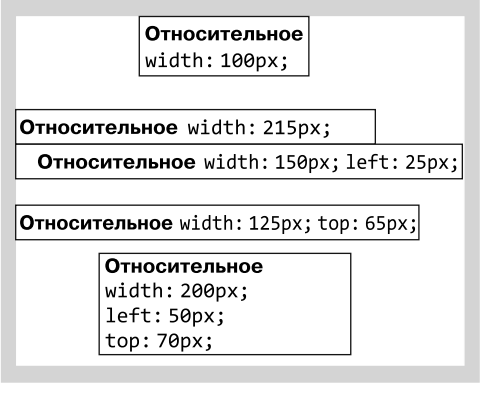
```
001 /* Применить границу ко всем элементам <div> */
002 div { border: 1px solid gray; }
003
004 /* Установить произвольные значения ширины и положения */
005 #A { width: 100px; top: 25px; left: l00px; }
006 #B { width: 215px; top: 50px; }
007 #C { width: 250px; top: 50px; left:25px; }
008 #D { width: 225px; top: 65px; }
009 #E { width: 200px; top: 70px; left:50px; }
```

Граница `1px solid gray` применена ко всем элементам `div`, поэтому теперь легче увидеть фактические размеры каждого HTML-элемента при отображении его в браузере.

Далее мы применим свойства `position: static` и `position: relative` к элементу `div`, чтобы увидеть разницу между статичным и относительным позиционированием.



`position: static;`



`position: relative;`

По сути, элементы с позиционированием `static` и `relative` одинаковы, за исключением того, что элементы `relative` могут иметь `top` (верхнюю) и `left` (левую) позиции относительно их исходного местоположения. Относительные элементы также могут иметь `right` (правое) и `bottom` (нижнее) положение.

Относительное позиционирование хорошо подходит для оформления текста. Хотя достичь того же эффекта более правильно с помощью свойства `padding` и `margin`. Вы обнаружите, что *относительного* позиционирования недостаточно для размещения блокирующих элементов, таких как изображения, в определенном месте внутри области родительского элемента.

Следовательно, свойство `position: relative` не гарантирует полную точность при необходимости разместить элемент в идеальном месте в его родительском контейнере. Для такой цели больше всего подходит свойство `position: absolute`.

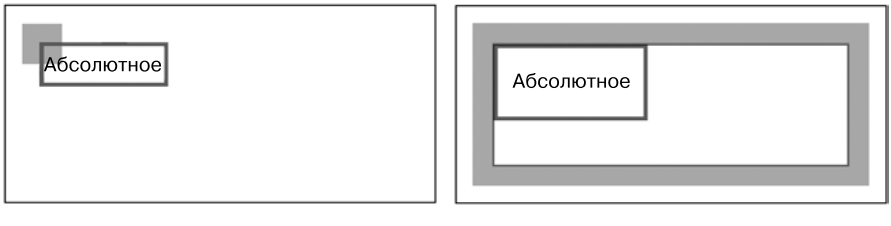
5.3. Абсолютное и фиксированное позиционирование

Абсолютное позиционирование используется для идеального размещения пикселов внутри родительского контейнера. Фиксированные элементы практически идентичны позиционированным абсолютно. За исключением того, что они не реагируют на изменения положения ползунка полосы прокрутки.

Это пример того, как элементы с позиционированием `absolute` и `fixed` *схлопывают* родительский элемент, если для родительского контейнера не заданы размеры. Это может казаться неважным, однако при верстке макетов вы часто будете сталкиваться с такими случаями, особенно при переключении элементов с позиционирования `relative` на `absolute`.

В данной главе мы рассмотрим более приближенные к реальности примеры.

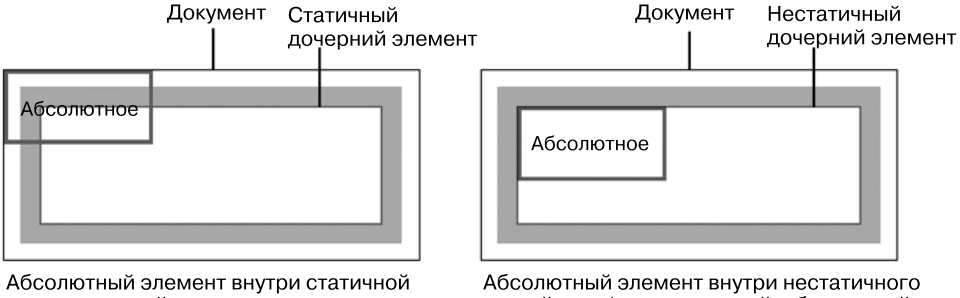
Обратите внимание: если свойства `width` и `height` родителя не указаны явно, то применение позиционирования `absolute` (или `fixed`) к его единственному дочернему элементу преобразует его размеры в 0×0, однако данный элемент все равно будет позиционироваться относительно него:



На предыдущей схеме слева абсолютно позиционированные элементы не заполняют свой родительский контейнер содержимым. Они как бы плавают над ним, сохраняя положение относительно своего элемента-контейнера. Справа размеры родительского элемента заданы явно. Технически для потомка со свойством `position: absolute` не задан никакой эффект, его точка поворота все еще находится в положении 0×0 родительского элемента.

Чтобы элементы со свойством `position: absolute` были выровнены относительно их родителя, его свойство `position` не должно быть установлено в `static` (по умолчанию):

Для того чтобы понять, как абсолютное позиционирование влияет на элемент, к которому применяется, нужно провести черту между часто происходящими *двумя уникальными случаями*.

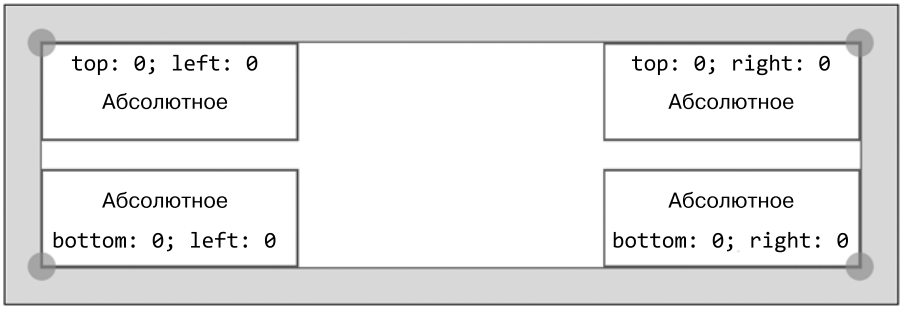


Абсолютный элемент внутри статичной позиции контейнера по-прежнему определяется относительно документа («проваливается»)

Абсолютный элемент внутри нестатичного контейнера (относительный, абсолютный, фиксированный, «липкий»). Положение определяется относительно нестатичного контейнера

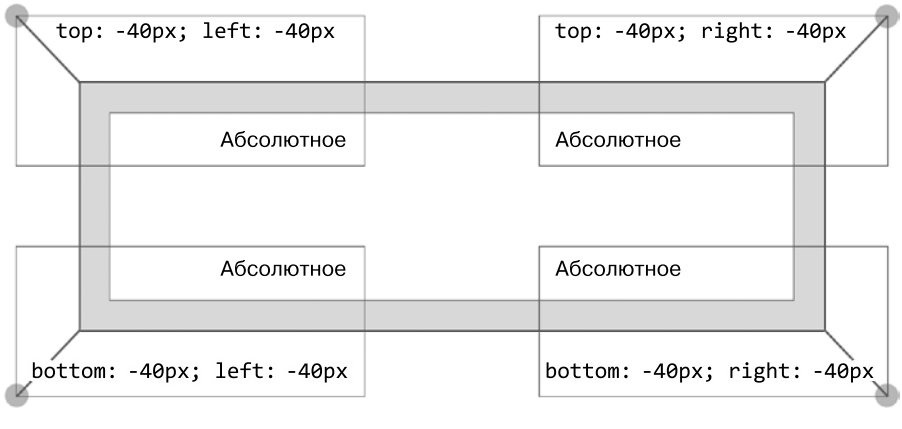
Как видите, элементы с абсолютным позиционированием ведут себя по-разному в зависимости от того, внутри какого контейнера они находятся: статичного или нестатичного.

Использование свойства `position: absolute` для выравнивания элементов по углам родителя:



Изменить начальную точку, из которой будет рассчитываться смещение, можно, комбинируя положения `top`, `left`, `bottom` и `right`. Однако не получится одновременно использовать положения `left` и `right`, так же как и `top` и `bottom`. При таком применении один элемент перекроет другой.

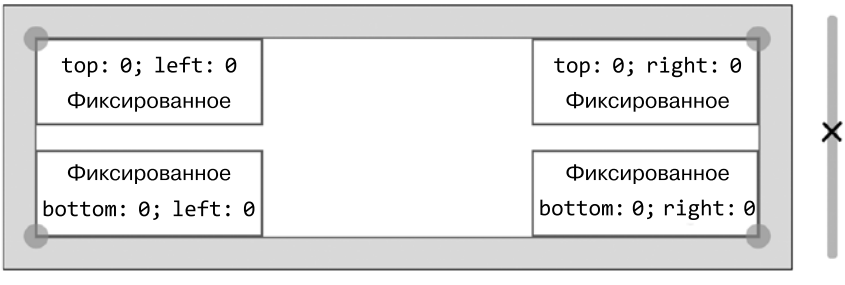
Использование свойства `position: absolute` с отрицательными значениями:



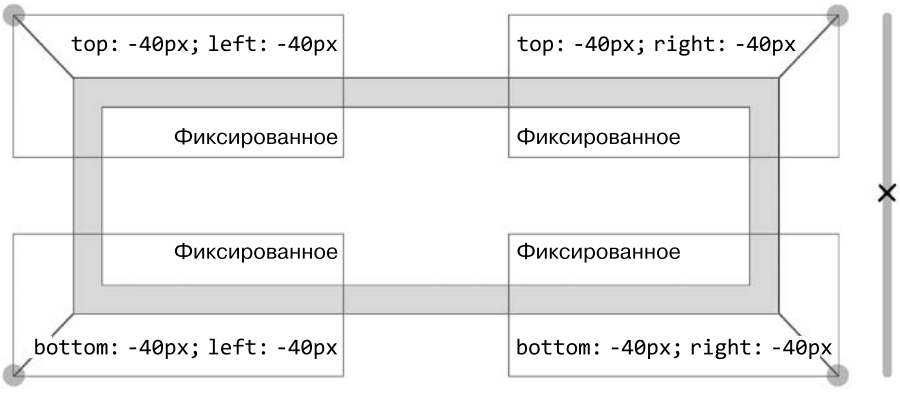
5.4. Фиксированное позиционирование

Данное позиционирование работает идентично абсолютному, за исключением того, что такие элементы не реагируют на полосу прокрутки. Элементы остаются в том месте на экране (относительно документа), где они были размещены, независимо от текущей позиции полосы прокрутки.

Использование свойства `position: fixed` для размещения элементов в фиксированном месте на экране относительно документа:



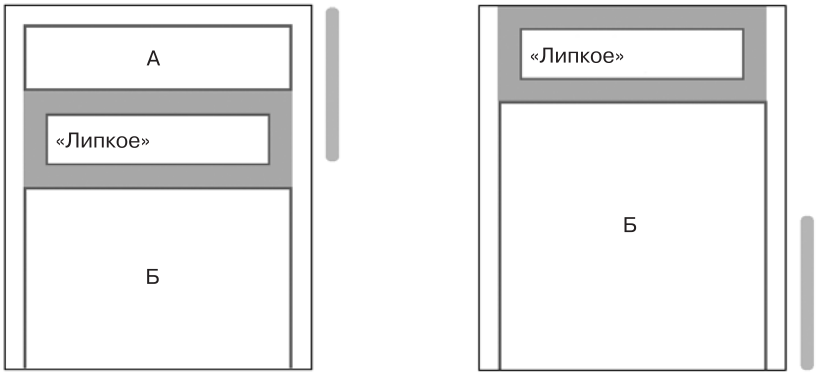
Использование свойства `position: fixed` с отрицательными значениями:



5.5. «Липкое» позиционирование

Это позиционирование было одним из последних дополнений в CSS. Ранее для достижения того же эффекта вам приходилось писать собственный код JavaScript или мультимедийный запрос.

«Липкое» позиционирование часто используется для создания плавающих панелей навигации:



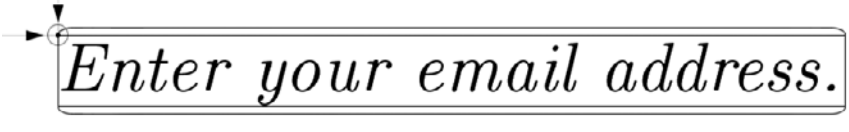
Далее приведен простой код, чтобы навигационная панель «прилипала» к верхней (`top: 0`) границе экрана. Обратите внимание: добавлен код `-webkit-sticky` для совместимости с браузерами на движке Webkit (такими как Chrome):

```
001 .navbar {
002     /* Определение некоторых основных настроек */
003     padding: 0px;
004     border: 20px solid silver;
005     background-color: white;
006     /* Добавить липкость */
007     position: -webkit-sticky;
008     position: sticky;
009     top: 0;
010 }
```

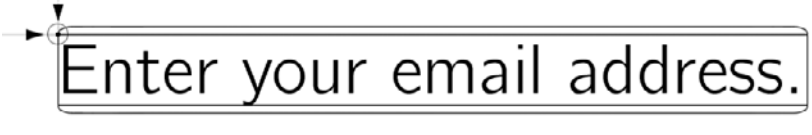
6 Работа с текстом

Мы не станем тратить много времени на рассмотрение схематичных рисунков для текста, поскольку вы видели примеры практически повсюду на сайтах или в социальных сетях. Основными свойствами для оформления текста в CSS являются: `font-family`, `font-size`, `color`, `font-weight` (`normal` или `bold`), `font-style` (например, `italic`) и `text-decoration` (`underline` или `none`).

Далее приведен пример свойства `font-family: "CMU Classical Serif"`. Это один из лучших шрифтов.

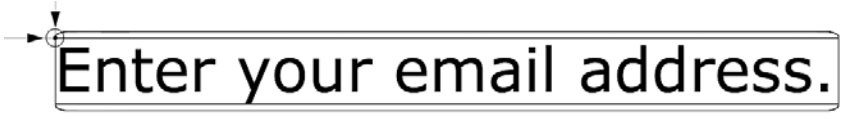


А вот свойство `font-family: "CMU Bright"`. Еще один красивый шрифт!

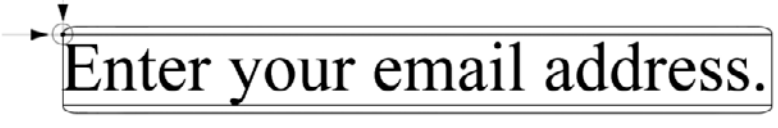


Свойство `font-family: Arial, sans-serif` используется в Google:

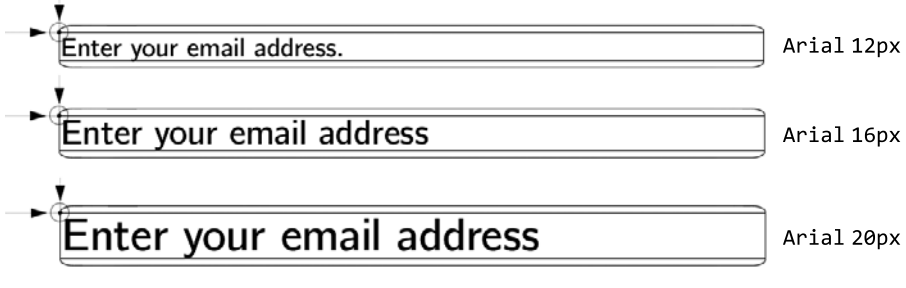
Свойство `font-family: Verdana, sans-serif`:



Обратите внимание: шрифт `sans-serif` здесь используется в качестве резервного. Можно указать еще больше шрифтов, разделяя их запятыми. Если первый шрифт в списке недоступен или не может быть отображен текущим браузером, то CSS обратится к следующему доступному шрифту в списке. Шрифт Times New Roman, используемый в примере ниже, будет задействован, если другой не был найден.



Можно изменить размер шрифта с помощью свойства `font-size`. `16px` — это размер по умолчанию (`medium`):



Размер шрифта можно указать с помощью единиц измерения pt, px, em или %. По умолчанию 100 % соответствует `12pt`, `16px` или `1em`. Зная это, вы можете предугадать значения для получения либо большего, либо меньшего шрифта относительно размера по умолчанию.

pt	px	em	%	размер	sans-serif по умолчанию
6pt	8px	0.5em	50%		Sample text
7pt	9px	0.55em	55%		Sample text
7.5pt	10px	0.625em	62.5%	x-small	Sample text
8pt	11px	0.7em	70%		Sample text
9pt	12px	0.75em	75%		Sample text
10pt	13px	0.8em	80%	small	Sample text
10.5pt	14px	0.875em	87.5%		Sample text
11pt	15px	0.95em	95%		Sample text
12pt	16px	1em	100%	medium	Sample text
13pt	17px	1.05em	105%		Sample text
13.5pt	18px	1.125em	112.5%	large	Sample text
14pt	19px	1.2em	120%		Sample text
14.5pt	20px	1.25em	125%		Sample text
15pt	21px	1.3em	130%		Sample text
16pt	22px	1.4em	140%		Sample text
17pt	23px	1.45em	145%		Sample text
18pt	24px	1.5em	150%	x-large	Sample text
20pt	26px	1.6em	160%		Sample text
22pt	29px	1.8em	180%		Sample text
24pt	32px	2em	200%	xx-large	Sample text
26pt	35px	2.2em	220%		Sample text
27pt	36px	2.25em	225%		Sample text
28pt	37px	2.3em	230%		Sample text
29pt	38px	2.35em	235%		Sample text
30pt	40px	2.45em	245%		Sample text
32pt	42px	2.55em	255%		Sample text
34pt	45px	2.75em	275%		Sample text
36pt	48px	3em	300%		Sample text

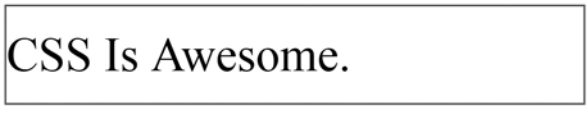
100% = 16px = medium

Далее свойство `font-weight` показано на пользовательском шрифте Raleway, доступном в Google Fonts:

font-weight	Raleway
100	Thin
200	Extra-Light
300	Light
400	Regular
500	Medium
600	**Semi-Bold**
700	**Bold**
800	**Extra-Bold**
900	**Black**

6.1. Свойство text-align

Выравнивание текста внутри HTML-элемента — одно из самых простых действий, доступных в CSS.



`text-align: left`
(по умолчанию)

`text-align: center`

`text-align: right`

6.2. Свойство text-align-last

Данное свойство аналогично `text-align`, за исключением того, что относится только к последней строке текста в абзаце.

CSS Is Awesome, that much we know. However, we need to write a bit more text here, in order to demonstrate how the CSS property text-align-last works, justifying only the last line of text in a paragraph.

`text-align-last: left` (по умолчанию)

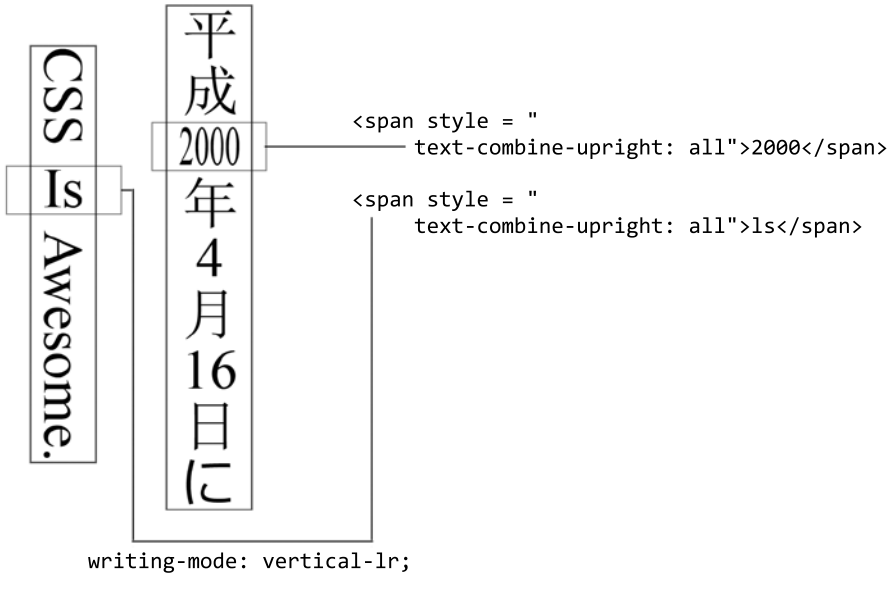
CSS Is Awesome, that much we know. However, we need to write a bit more text here, in order to demonstrate how the CSS property text-align-last works, justifying only the last line of text in a paragraph.

`text-align-last: center`

CSS Is Awesome, that much we know. However, we need to write a bit more text here, in order to demonstrate how the CSS property text-align-last works, justifying only the last line of text in a paragraph.

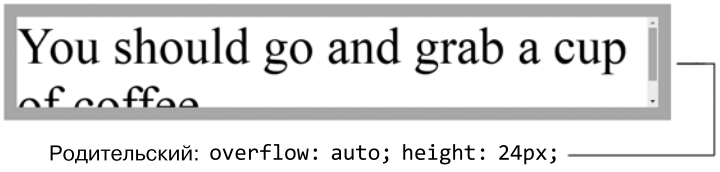
`text-align-last: right`

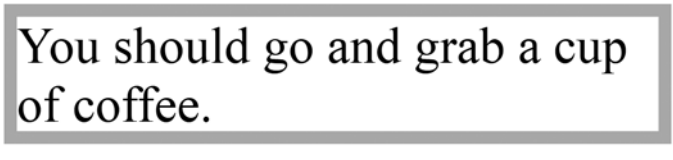
Присваивая свойству `writing-mode` значения `vertical`, вы также можете использовать свойство `text-combine-upright: all`:

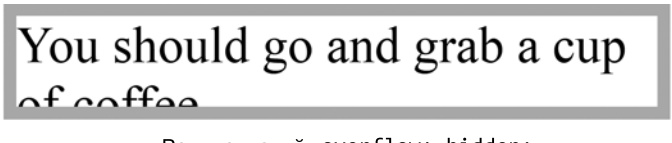


6.3. Свойство overflow

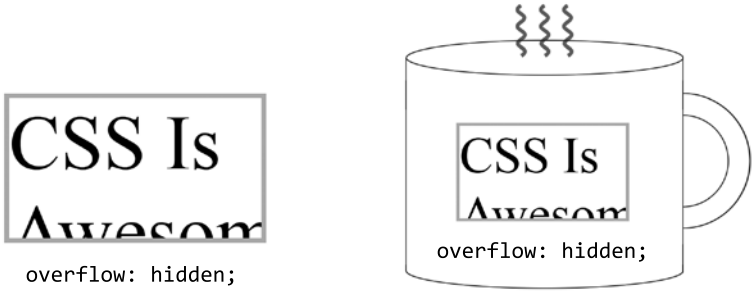
При вложении текста в родительский элемент вы можете сделать его прокручиваемым, применив свойство `overflow: scroll` к родительскому элементу:

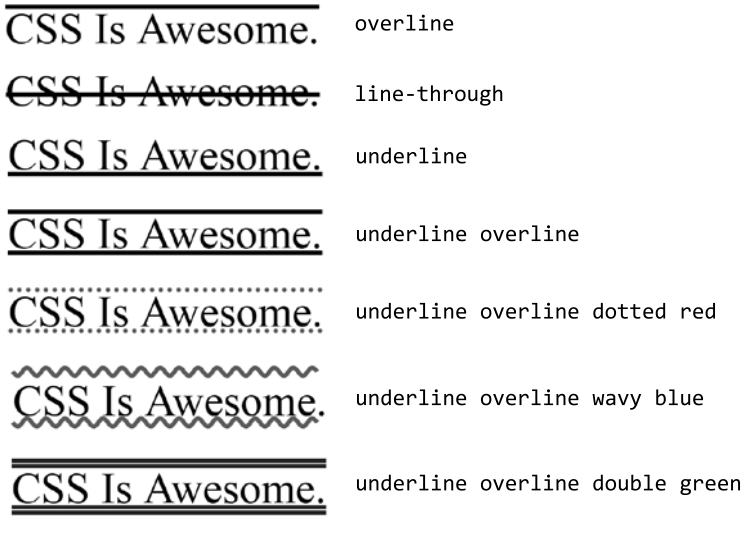






Классический вариант — `overflow: hidden`. Вам стоит пойти и выпить чашечку кофе.

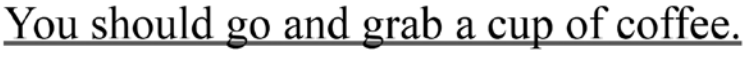




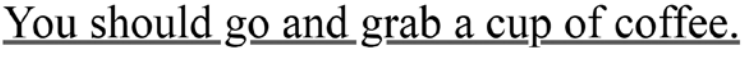
Обратите внимание: отдельные значения разделены пробелами. Вы заметите это во всем спектре комбинаций значений CSS, обычно использующихся в качестве сокращенного варианта написания для отдельных свойств. С помощью свойства `text-decoration` vожно добавить подчеркивание к тексту в верхней и нижней части текста. Хотя данное свойство в макете встречается редко, необходимо понимать, что оно существует и поддерживается всеми браузерами.

6.4. Свойство text-decoration-skip-ink

Свойство `text-decoration-skip-ink` отвечает за отображение линий под/над текстом, когда они пересекают глифы. Это на самом деле полезно для улучшения визуальной целостности заголовков страниц или любого подчеркнутого текста, в котором используются большие буквы.




```
text-decoration: underline solid blue
    text-decoration-skip-ink: none
```



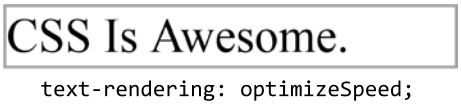

```
text-decoration: underline solid blue
    text-decoration-skip-ink: auto
```

6.5. Свойство text-rendering

Данное свойство, возможно, не вызовет заметных различий в четырех его проявлениях (`auto`, `optimizeSpeed`, `optimizeLegibility` и `geometryPrecision`). Но считается, что в некоторых браузерах, использующих значение `optimizeSpeed` этого свойства, улучшается скорость рендеринга больших блоков текста. Значение `optimizeLegibility` — единственное, на самом деле создававшее физическое различие в отображении текста в наших экспериментах с браузером Chrome, смещая слова ближе друг к другу в отдельных комбинациях символов.

Названия четырех использованных значений отражают их суть:


```
text-rendering: auto;
```



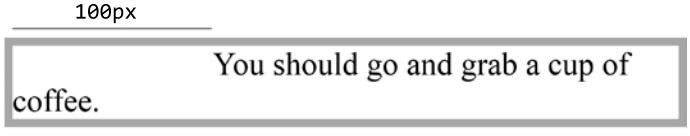

```
text-rendering: optimizeSpeed;
```


```
text-rendering: optimizeLegibility;
```

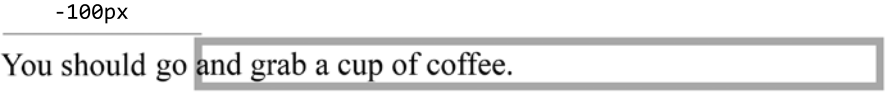

```
text-rendering: geometricPrecision;
```

6.6. Свойство text-indent

Данное свойство отвечает за выравнивание текста. Применяется редко, но в некоторых случаях, например на новостных сайтах или в текстовых редакторах, может оказаться полезным.



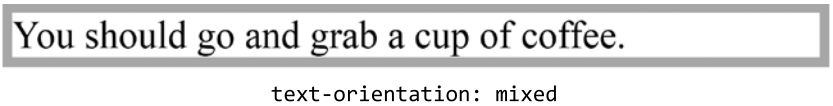
`text-indent: 100px`



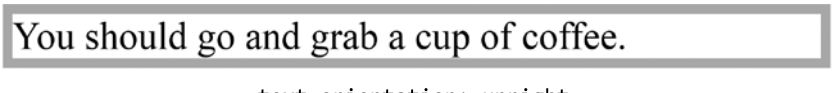
`text-indent: -100px`

6.7. Свойство text-orientation

Данное свойство определяет направление текста. Может быть полезно для рендеринга разных языков, в которых поток текста проходит справа налево или сверху вниз. Часто применяется вместе со свойством `writing-mode`.

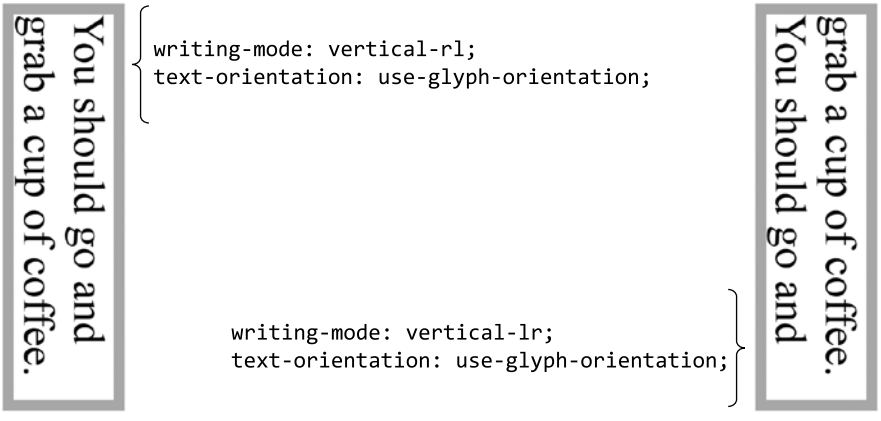


`text-orientation: mixed`



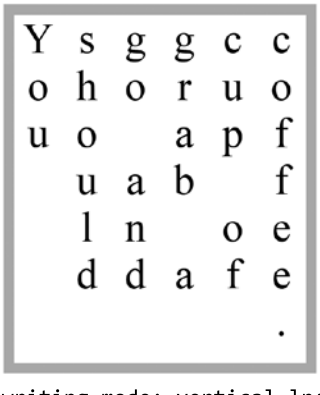
`text-orientation: upright`

Вместе со свойством `writing-mode: vertical-rl` (справа налево) или `writing-mode: vertical-lr` (слева направо) свойство `text-orientation` можно использовать для выравнивания текста практически в любом направлении.



Далее свойству `text-orientation` присваивается значение `upright`. Применимо к SVG-элементам свойство `use-glyph-direction` заменяет устаревшие свойства `use-direction-vertical` и `use-direction-horizontal`.


```
writing-mode: vertical-rl;
text-orientation: upright;
```

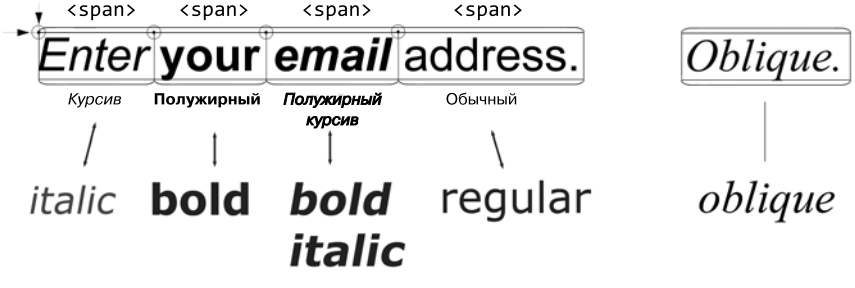



```
writing-mode: vertical-lr;
text-orientation: upright;
```

Чтобы центрировать текст по вертикали в любом элементе, установите его высоту строки с помощью свойства `line-height: 60px`. Размер шрифта и значение свойства `line-height` не всегда совпадают:

Посмотрите на лигатуры с настройками `font-feature-settings: "liga" 1` и `font-feature-settings: "liga" on`:

Распространенные текстовые эффекты (`italic` (курсив), `bold` (полужирный) и `oblique` (наклонный)) достигаются с помощью свойств `font-style` и `font-weight`:



Свойства `text-align` и `line-height` часто используются для центрирования текста на кнопках:

Стиль по умолчанию не очень корректно работает для кнопок

Выравнивание по центру и line-height можно установить для точности положения текста

6.8. Свойство text-shadow

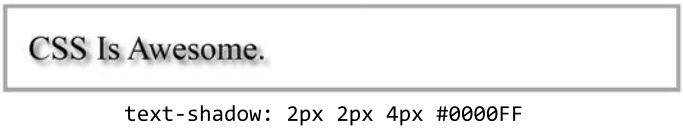

```
text-shadow: 0px 0px 0px #0000FF
```


```
text-shadow: 0px 0px 1px #0000FF
```


```
text-shadow: 0px 0px 2px #0000FF
```


```
text-shadow: 0px 0px 3px #0000FF
```


```
text-shadow: 0px 0px 4px #0000FF
```

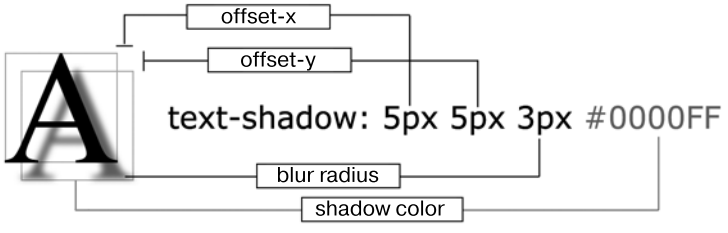



```
text-shadow: 2px 2px 4px #0000FF
```

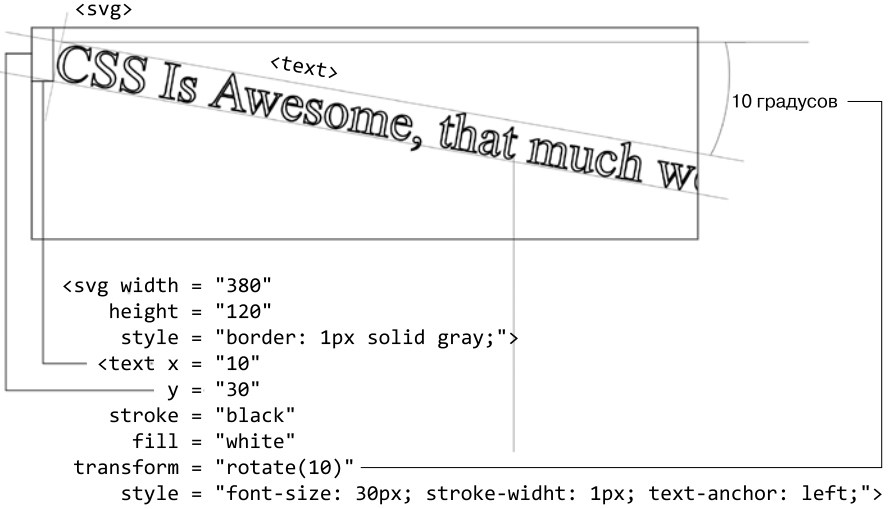

```
text-shadow: 3px 3px 4px #0000FF
```


```
text-shadow: 5px 5px 4px #0000FF
```

Можно добавить тень к тексту с помощью свойства `text-shadow`. Это свойство задает смещение вдоль осей *X* и *Y*, можно также настроить радиус размытия и цвет тени:

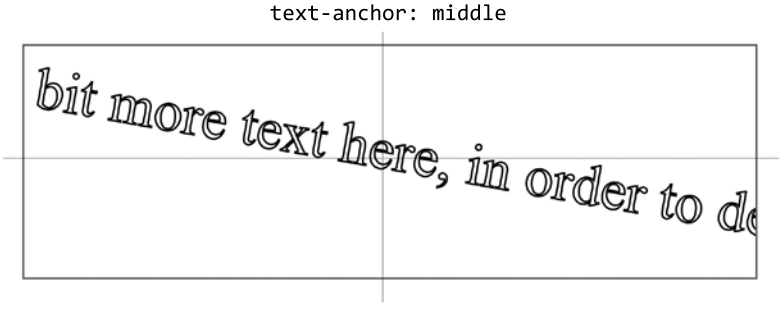


Мы не станем глубоко вдаваться в SVG-контент, который тоже форматируется свойствами CSS. На эту тему можно написать целую книгу. Но в качестве краткой вставки — повернутый текст SVG создается следующим образом.

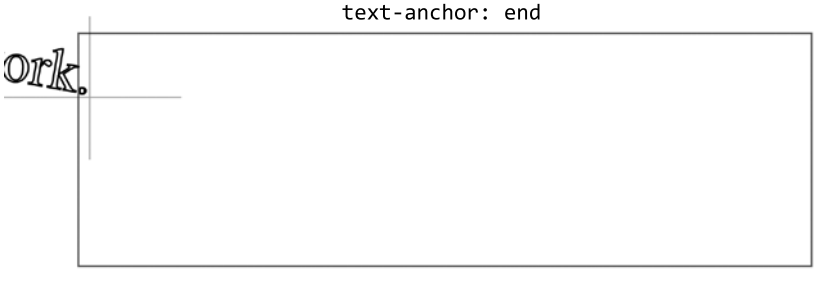



```
<svg width = "380"
    height = "120"
     style = "border: 1px solid gray;">
  <text x = "10"
        y = "30"
   stroke = "black"
     fill = "white"
transform = "rotate(10)"
    style = "font-size: 30px; stroke-widht: 1px; text-anchor: left;">
       CSS Is Awesome, that much we know. However, we need to write
       a bit more text here, in order to demonstrate how SVG
       rotations work.</text>
</svg>
```

С помощью свойства `text-anchor` можно установить центральную точку, вокруг которой текст будет вращаться:



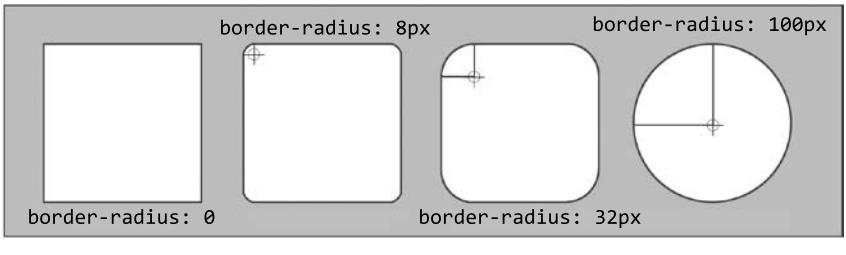
Чтобы установить центр поворота в самый конец текстового блока, свойству `text-anchor` следует задать значение `end`. Мы увидим похожее поведение в свойстве `transform`, которое можно использовать для поворота целых HTML-элементов и текста внутри них:



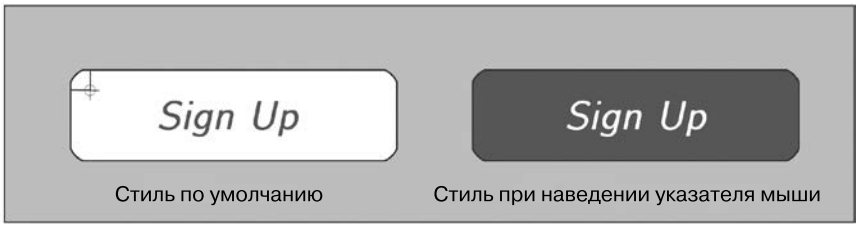
7 Свойства margin, border-radius, box-shadow и z-index

Ниже в произвольном порядке изображены несколько объектов для того, чтобы наглядно продемонстрировать часто используемые свойства CSS.

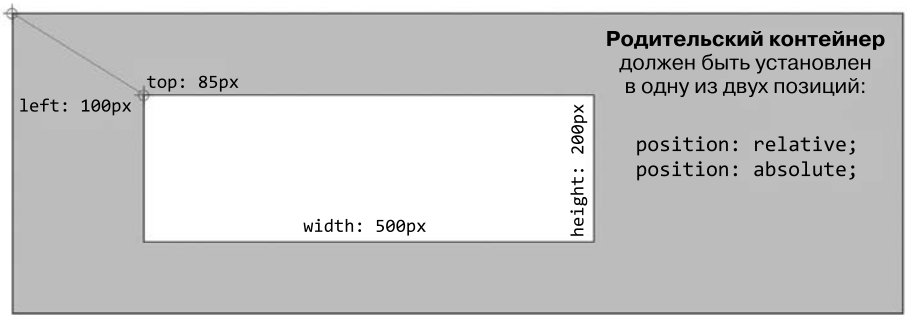
Свойство `border-radius` используется для добавления скругленных углов к квадратным или прямоугольным элементам HTML:



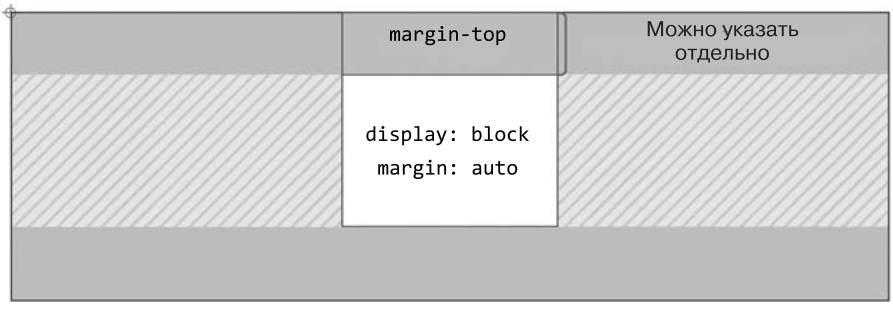
Используя псевдоселектор `:hover`, вы увидите что произойдет, если на элемент навести указатель мыши:



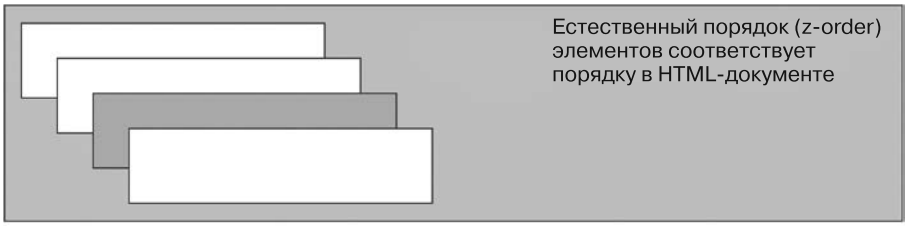
Для родительского контейнера должно быть явно задано свойство `position: relative` либо `position: absolute`, чтобы использовать в нем дочерний элемент, который также задействует свойство `position: absolute`:

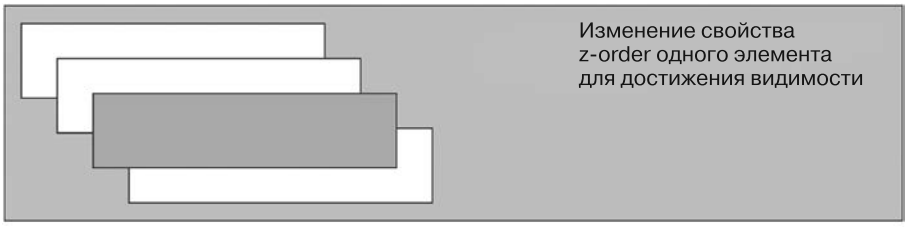


Для выравнивания элемента по горизонтали можно использовать свойство `margin: auto`. При этом свойство `display` должно иметь значение `block`. Свойство `margin-top` подойдет для смещения элемента с помощью пространства в его верхней части. Вы также можете использовать свойства `margin-left`, `margin-right`, `margin-bottom` (устанавливают величину отступа от левого, правого, нижнего края элемента соответственно).

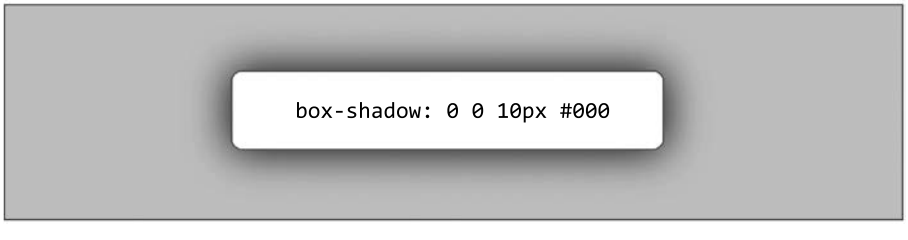


Чтобы можно было определить порядок рисования элемента в большинстве распространенных браузеров, свойство `z-index` принимает числовое значение в диапазоне от 0 до 2 147 483 647. В Safari 3 максимальное значение `z-index` составляет 16 777 271.

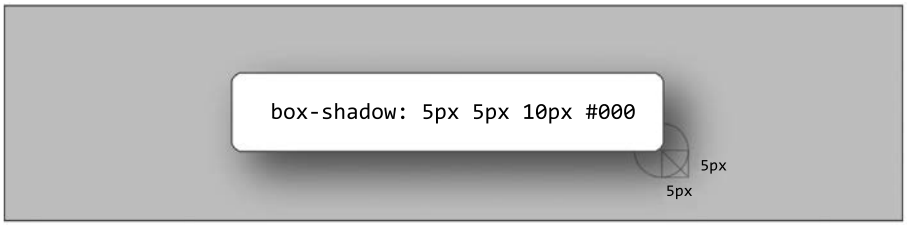




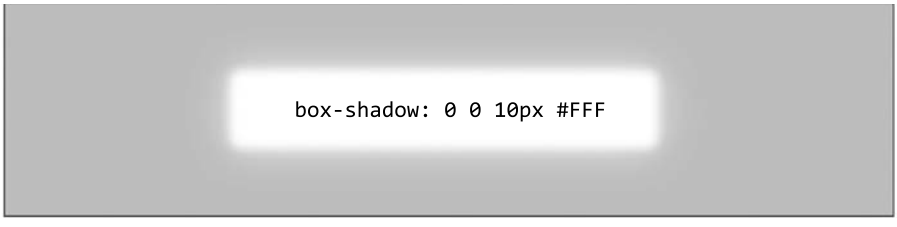
В следующем примере свойство `box-shadow` используется для добавления тени вокруг широкого элемента. Он принимает те же параметры, что и `text-shadow`, например: `box-shadow: 5px 5px 10px #000` (смещение по осям *X* и *Y*, радиус и цвет тени).



На следующей схеме `box-radius` управляет радиусом кривой угла по осям *X* и *Y*:



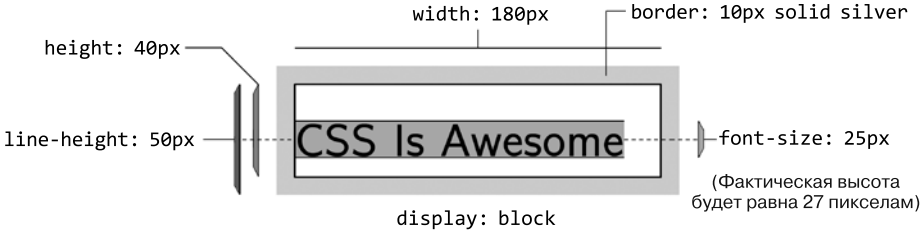
Используя яркие цвета со свойством `box-shadow`, можно создать эффект свечения вокруг HTML-элементов:



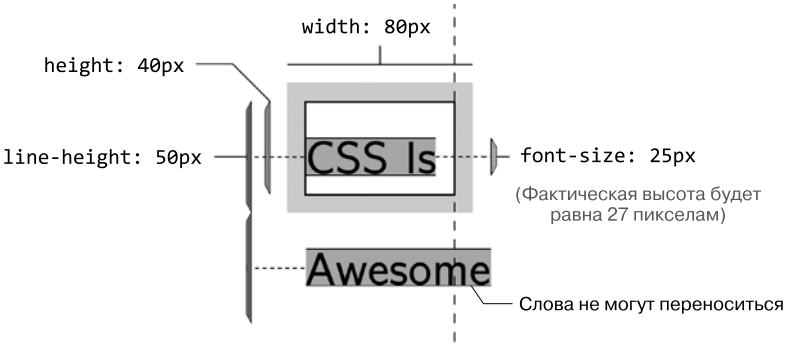
То, что вы ожидаете от простого блокирующего элемента:

Когда ширина элемента становится меньше, чем ширина его текстового содержимого, текст автоматически перемещается на следующую доступную строку, даже если выходит за пределы границы элемента:

Рассмотрим более подробно предыдущие примеры.

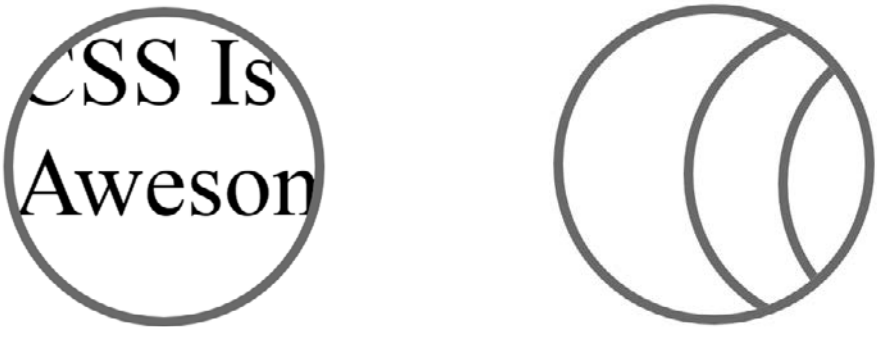


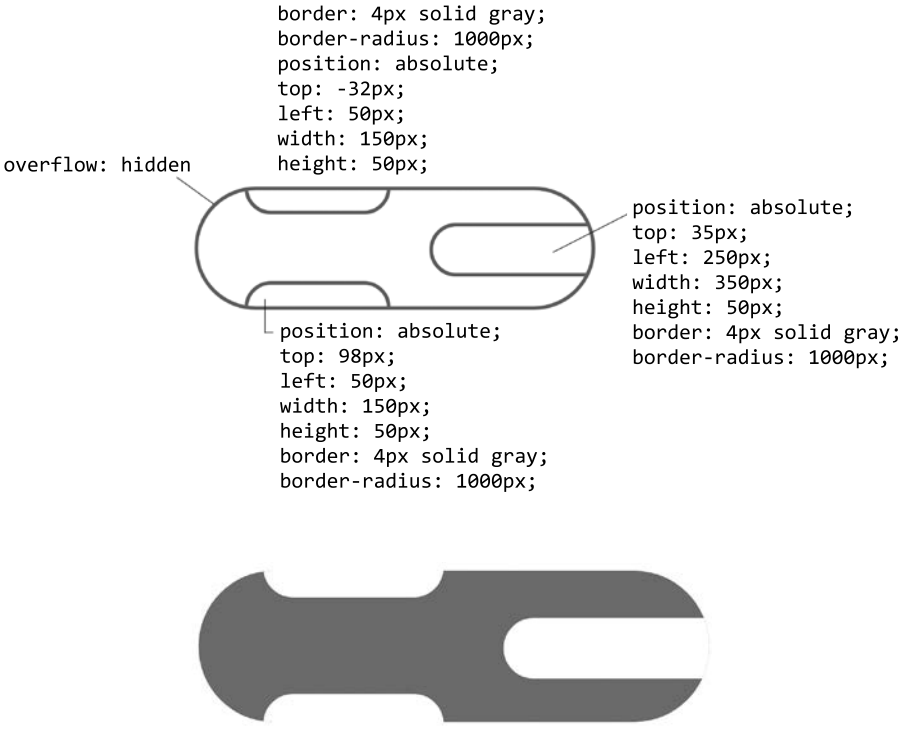
Здесь физическая высота текста в действительности будет равна 27 пикселам, на 2 больше, чем 25 пикселов — исходное установленное значение. Значение, установленное свойством `line-height`, может выходить за пределы области содержимого.



Здесь мы видим, что слово Awesome перешло на следующую строку. Вдобавок обратите внимание: одно слово не может обернуться вокруг элемента контейнера, даже если его ширина меньше. Другими словами, свойство `overflow` принимает значение `visible` по умолчанию.

Вы можете эффективно вырезать содержимое вне его контейнера, установив свойство `overflow: hidden`. Это будет работать даже на элементах с закругленными углами:

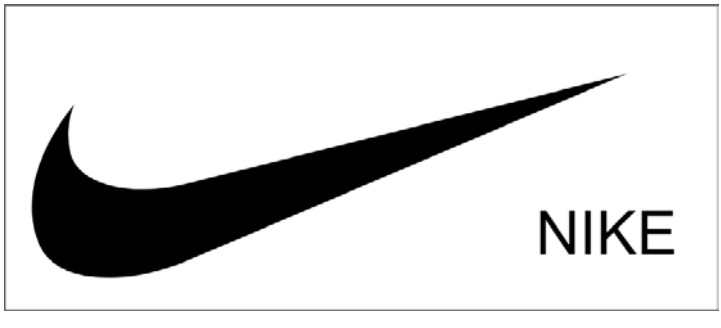




На этом рисунке показано то же, что и в примере выше, за исключением следующего: свойству `background` родительского контейнера присвоено значение `gray`, а свойству `background` элементов в нем — значение `white`. По желанию всегда можно творчески подойти к созданию некоторых интересных объектов. В конце книги мы рассмотрим оригинальный пример с автомобилем.

8 Логотип Nike

Комбинируя приемы из предыдущей главы со свойством `transform: rotate` (далее будет рассматриваться более подробно) и наши знания псевдоселекторов `:before` и `:after`, можно создать логотип Nike из одного HTML-элемента.



Определим наш основной контейнер:

```
 1 #nike {
 2   position: absolute;
 3   top: 300px; left: 300px;
 4   width: 470px; height: 200px;
 5   border: 1px solid gray;
 6   overflow: hidden;
 7   font-family: Arial, sans-serif;
 8   font-size: 40px;
 9   line-height: 300px;
10   text-indent: 350px;
11   z-index: 3;
12 }
```

Обратите внимание: свойство `overflow: hidden` здесь используется для того, чтобы отсечь все находящееся за пределами контейнера.

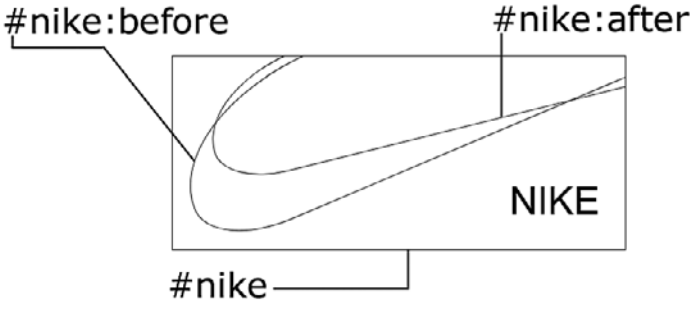
С помощью псевдоэлементов `#nike:before` и `#nike:after` создадим основу логотипа — длинную черную полосу. Скругленные углы используются в данном примере для создания знаменитой кривой Nike:

```
 1 #nike:before {
 2   content: " ";
 3   position: absolute;
 4   top: -250px;
 5   left: 190px;
 6   width: 150px;
 7   height: 550px;
 8   background: black;
 9   border-top-left-radius: 60px 110px;
10   border-top-right-radius: 130px 220px;
11   transform: rotate(-113deg);
12   z-index: 1;
13 }
```

Точно так же мы создадим еще одну изогнутую рамку. Белый фон послужит маской для обработки остальной части логотипа. В данном примере угол поворота имеет решающее значение. Именно он формирует узнаваемую кривую логотипа. Для обеспечения правильного наложения элементов мы также использовали свойство `z-index` со значениями `1`, `2` и `3` соответственно.

```
 1 #nike:after {
 2   content: " ";
 3   position: absolute;
 4   top: -235px;
 5   left: 220px;
 6   width: 120px;
 7   height: 500px;
 8   background: black;
 9   border-top-left-radius: 60px 110px;
10   border-top-right-radius: 130px 220px;
11   background: white;
12   transform: rotate(-104deg);
13   z-index: 2;
14 }
```

Вот еще один вариант логотипа, состоящего из трех элементов (один HTML-элемент и два псевдоэлемента). На этот раз с прозрачным фоном.



Фактический HTML-код — это всего лишь один элемент `div` с идентификатором `nike`.

```
1 <div id = "nike">NIKE</div>
```

9 Свойство display

Свойства CSS используются для установки значения HTML-элементов, определяющих отображение на экране. Рисунки в этой главе иллюстрируют, каким образом свойства влияют на каждый элемент.

Для определения размещения отдельных элементов свойству `display` можно задавать любое из значений `inline`, `block`, `inline-block` или `float`.

Здесь задано свойство `display: inline`. Это значение устанавливается по умолчанию, используется для `span`, `b`, `i` и некоторых других HTML-элементов, предназначенных для работы с отображением текста внутри родительских контейнеров с неопределенной шириной.

Здесь каждый элемент размещается непосредственно справа после того, как длина его содержимого (или его ширина) в предыдущем элементе была превышена, что делает его естественным вариантом для отображения текста.

ПРИМЕЧАНИЕ

Длинные встроенные элементы автоматически переносятся в следующий ряд.

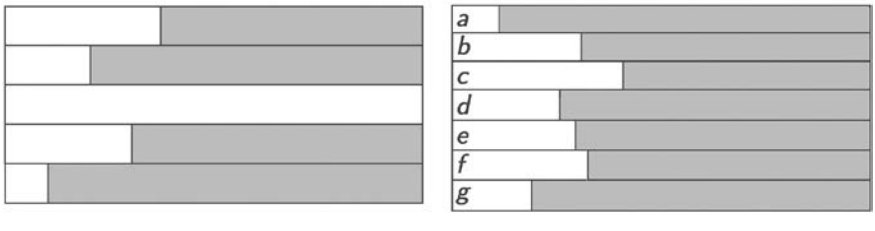
Позже, когда мы перейдем к главам, описывающим grid- и flex-верстку, вы увидите, как применение значений `flex` либо `grid` к свойству `display` может изменить поведение его *элементов*, находящихся внутри элемента контейнера, часто называемого их *родительским*.

Свойство `display: block`, в отличие от `inline`, автоматически блокирует весь ряд пространства независимо от ширины его содержимого. HTML-элементы `div` по умолчанию являются блокирующими элементами:

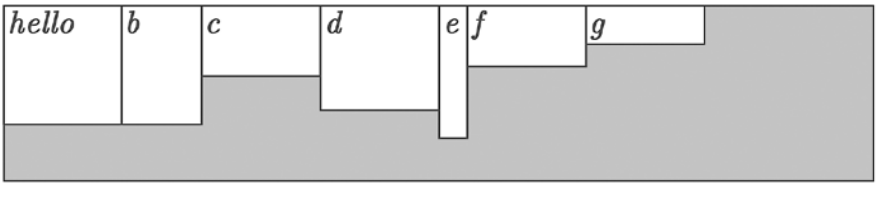
`block`

Свойство `display: block` с явно заданной шириной элемента показывает распознавание ширины контейнера элемента и ширины его содержимого:

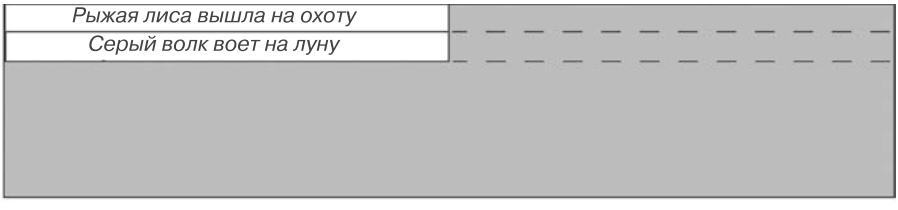
`div { width: n }`, где *n* — числовое значение в пикселах или процентах от ширины контейнера



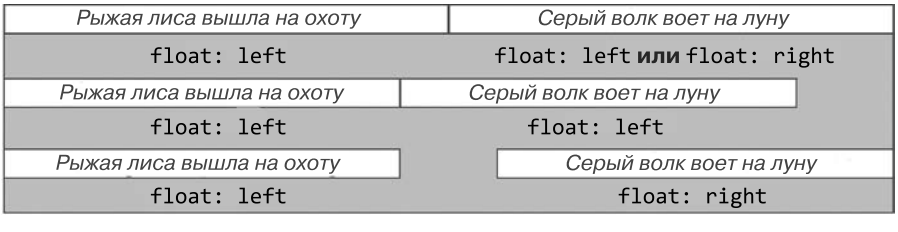
Свойство `display: inline-block` сочетает в себе поведение `inline` и `blocking`, чтобы включить пользовательский размер для встроенных элементов:



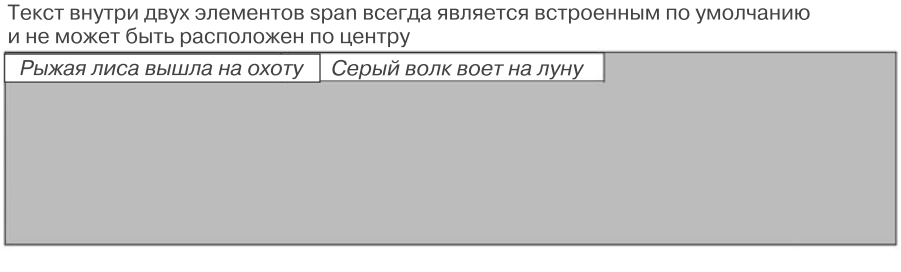
Центрированный текст (`text-align: center`) внутри двух блокирующих элементов шириной, равной 50 % контейнера. Обратите внимание: при блокировании всей строки родительского элемента область содержимого растягивается только до 50 % его ширины. Блокирующий элемент не определяется шириной его содержимого:



Два блокирующих элемента с явно заданной шириной около 50 % и `text-align: center` могут имитировать встроенные элементы, если также применяется `float: left` и/или `float: right`. Однако, в отличие от встроенных элементов, одиночный блокирующий элемент никогда не может перейти на следующую строку:



Встроенные элементы всегда ограничены шириной их содержимого, поэтому текст внутри них не может быть отцентрирован:

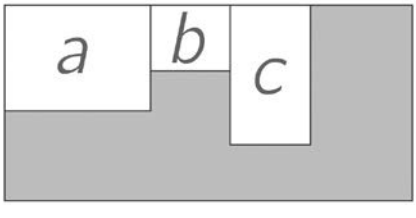


10 Свойство visibility

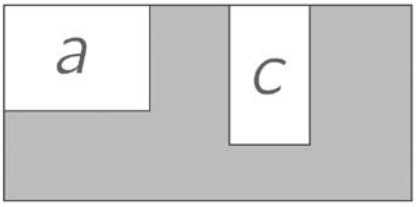
Свойство `visibility` скрывает или показывает элемент без изменения разметки документа.

Далее показан результат присвоения свойству `visibility` элемента `b` значения `hidden`. По умолчанию установлено значение `visible` (аналогично `unset`, или `auto`, или `none`).

```
.b {visibility: visible}
```

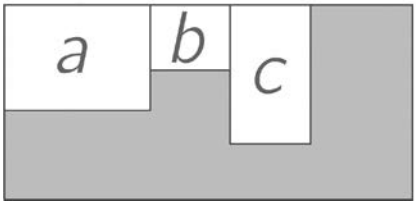



```
.b {visibility: hidden}
```

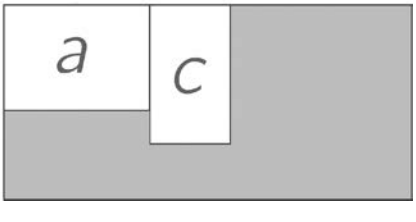


Здесь свойство `display: none` полностью удаляет элемент.

```
.b {display: block}
```

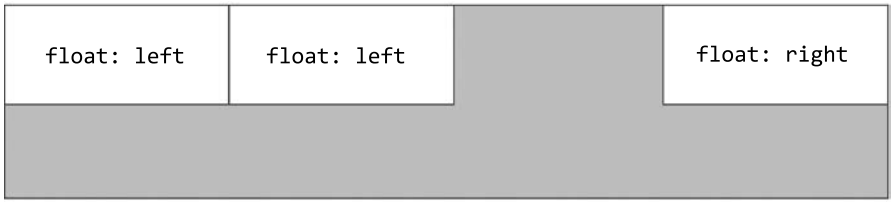



```
.b {display: none}
```

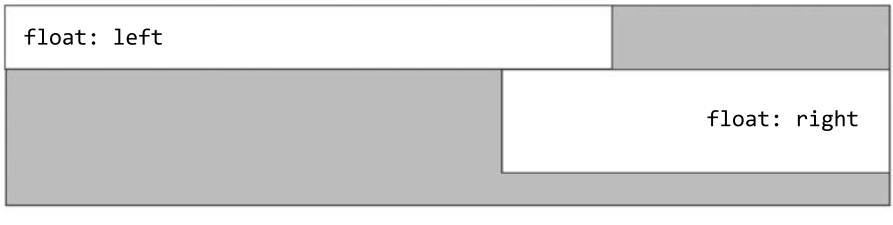


11 Плавающие элементы

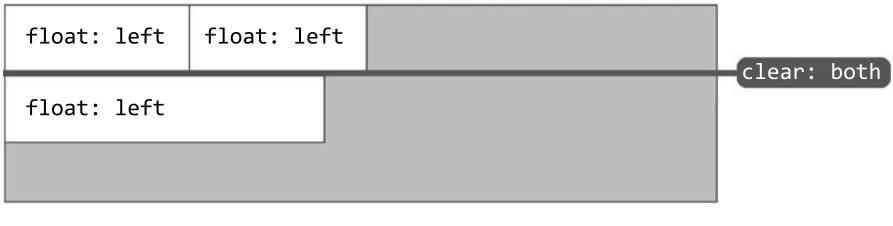
Блокирующие элементы со свойствами `float: left` и `float: right` появляются в одной строке, если сумма их ширины меньше, чем ширина родительского элемента:



Если общая сумма двух плавающих элементов превышает ширину родительского элемента, то один из них будет заблокирован другим и перейдет к следующей строке:



Очистить плавающие элементы и начать новую плавающую строку можно с помощью свойства `clear: both`:



12 Цветовые градиенты

Градиенты могут применяться по разным причинам. Наиболее распространенная из них — обеспечение плавного эффекта затенения в области какого-либо элемента пользовательского интерфейса. Вот еще несколько причин использования цветового градиента.

- **Плавная заливка фона цветом** делает HTML-элементы более привлекательными.
- **Экономия трафика** является еще одной важной причиной использования градиентов, поскольку они автоматически генерируются в браузере с помощью эффективного алгоритма затенения. Это значит, что их можно применять вместо изображений, намного дольше загружающихся с веб-сервера.
- **Простые определения** могут использоваться в свойстве `background` для создания интересных эффектов. Вы будете присваивать минимально необходимые значения свойствам `linear-gradient` (линейный градиент) или `radial-gradient` (радиальный градиент), чтобы создать эффект, продемонстрированный в следующем разделе.

12.1. Общие сведения

При черно-белой печати разница между градиентами не видна. Но для их освоения действительно нужно хорошо понять их направление и тип. Выделяют четыре типа: линейный `linear-gradient`, радиальный `radial-gradient`, повторяющийся линейный `repeating-linear-gradient` и повторяющийся радиальный `repeating-radial-gradient`. Следующий рисунок дает хорошее представление о разнообразии градиентов, которые можно создать для HTML-элементов с помощью CSS:


gradient-1.png

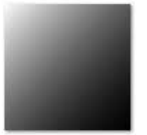

gradient-2.png

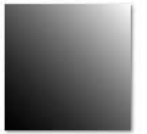

gradient-3-toleft.png

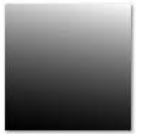

gradient-3-toright.png


gradient-4-to-bottomright.png

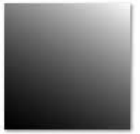

gradient-5-to-bottomleft.png


gradient-6-to-topleft.png


gradient-7-to-topright.png


gradient-8-10 deg.png

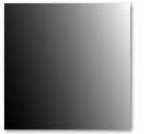

gradient-8-20 deg.png


gradient-8-30 deg.png


gradient-8-40 deg.png


gradient-8-50 deg.png

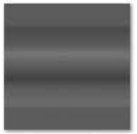

gradient-8-60 deg.png

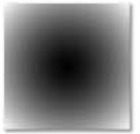

gradient-8-70 deg.png


gradient-8-80 deg.png


gradient-8-90 deg.png

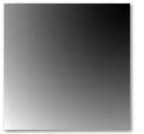

gradient-8-hsl-3-red-yellow-green.png

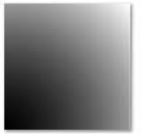

gradient-8-hsl-all-300-50-interval.png

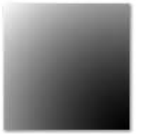

gradient-8-radial-black-white.png

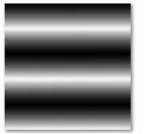

gradient-8-radial-white-black.png

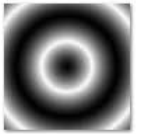

gradient-9-radial-attopleft.png


gradient-9-radial-attopright.png


gradient-9-radial-bottomleft.png


gradient-9-radial-bottomright.png


gradient-9-repeating-linear-gradient.png


gradient-9-repeating-radial-gradient.png

Здесь я немного схитрил... Изображения выше — файлы из моей папки градиентов, которую я создал во время работы над книгой. Но как мы на самом деле создаем градиенты с помощью команд CSS? Именно это мы и обсудим в следующем подразделе.

Образец для отображения градиентов

Мы проведем наши эксперименты с градиентами фона, используя простой элемент `div`. Сначала установим некоторые основные свойства, в том числе `width=500px` и `height=500px`.

Сейчас нам нужен простой квадратный элемент. Вставьте этот код между тегами элемента `head` в вашем HTML-документе.

```
001 <style type = "text/css">
002 div {
003 position: relative;
004 display: block;
005 width: 500px;
006 height: 500px;
007 }
008 <style>
```

Этот код CSS преобразует каждый элемент `div` на экране в квадрат размером 500 × 500 пикселов. О свойствах `position` и `display` будет рассказано несколько позже.

В качестве альтернативы мы можем назначить градиенты только одному HTML-элементу. В этом случае вы можете назначить CSS-стили отдельному элементу `div`, используя уникальный идентификатор, такой как `#my-gradient-box`, *или любой другой по вашему желанию*.

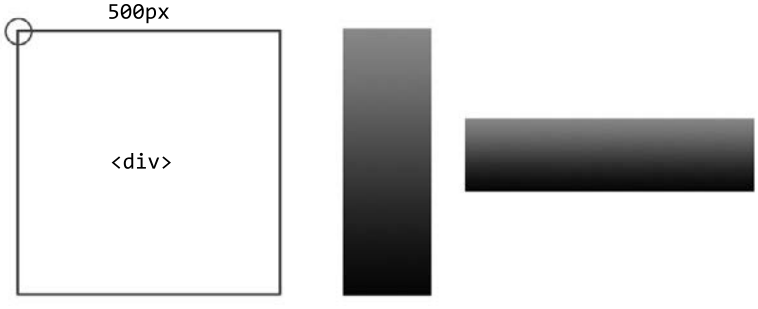
```
001 <style type = "text/css">
002 div#my-gradient-box { position: relative; display: block;
      width: 500px; height: 500px; }
003 <style>
```

А затем добавьте его в элемент `body`:

```
001 <!-- Эксперимент с цветными градиентными фонами в HTML -->
002 <div id = "my-gradient-box"></div>
```

Или можете ввести те же команды CSS непосредственно в атрибут `style` HTML-элемента, к которому хотите применить цветовой градиент:

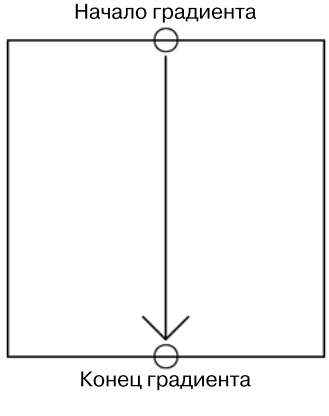
```
001 <div style = "position: relative; display: block; width: 500 px;
height: 500px;"></div>
```



Здесь показан элемент `div` размером 500 × 500 пикселов. Строка и столбец с правой стороны демонстрируют, как градиенты автоматически адаптируются к размеру элемента. Свойство градиента здесь осталось прежним. Изменились только размеры элемента, и градиент сразу выглядит совсем иначе. Имейте это в виду при создании собственных градиентов!

CSS-градиенты автоматически адаптируются к ширине и высоте элемента. *Можно произвести немного другой эффект.*

Основная идея градиентов — интерполировать как минимум два цвета. По умолчанию, без присвоения каких-либо дополнительных значений, предполагается вертикальное направление. Начальный цвет начинается сверху элемента, постепенно смешиваясь со 100 % второго цвета внизу. Можно создавать градиенты, комбинируя более двух цветов. Далее мы это и рассмотрим.



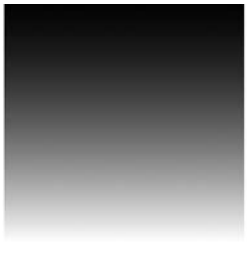
Все значения градиента CSS передаются в свойство `background`. Вот пример создания простого линейного градиента:

```
001 background: linear-gradient(black, white);
```

Ниже можно будет увидеть, как с помощью этой функции и ее значений создается градиент.

12.2. Типы градиентов

Рассмотрим разные стили градиента и визуализируем тип эффектов градиента, а также рассмотрим их применение.

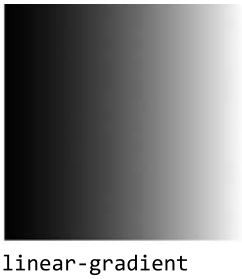


`linear-gradient(black, white)`

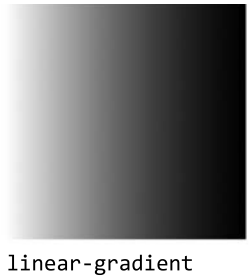
`linear-gradient(yellow, red)`

Это простой линейный градиент. *Слева:* от черного к белому. *Справа:* от желтого к красному.

Горизонтальные градиенты можно создать, указав начальное значение `to left` (налево) или `to right` (направо), в зависимости от желаемого направления:

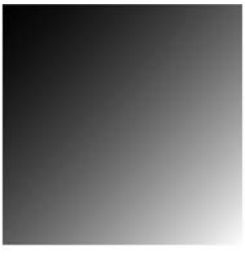


`linear-gradient`
`(to left, black, white)`

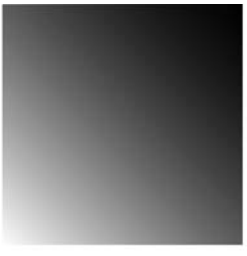


`linear-gradient`
`(to right, black, white)`

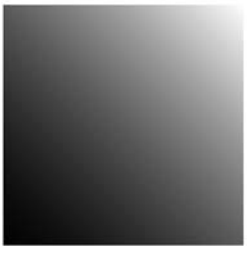
Можно запускать градиенты и в углах, чтобы создавать диагональные цветовые переходы. Достичь этого эффекта позволят значения `to top left` (вверх и налево), `to top right` (вверх и направо), `to bottom left` (вниз и налево) и `to bottom right` (вниз и направо):



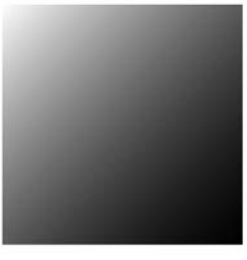
```
linear-gradient
(to top left, black, white)
```



```
linear-gradient
(to top right, black, white)
```



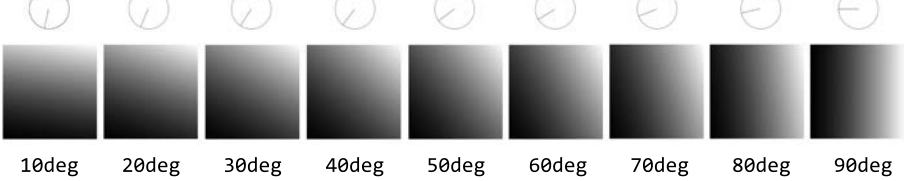
```
linear-gradient
(to bottom left, black, white)
```



```
linear-gradient
(to bottom right, black, white)
```

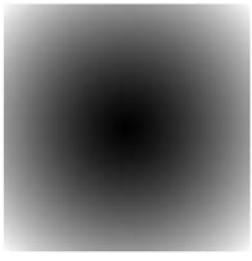
Если угла 45 градусов недостаточно, то можно указать произвольный градус в диапазоне от 0 до 360 непосредственно для линейного градиента `linear-gradient(30deg, black, white)`.

Обратите внимание: в следующем примере градиент постепенно меняет направление от потока вниз, к левой стороне, когда угол изменяется в прогрессии от 10 до 90 градусов:

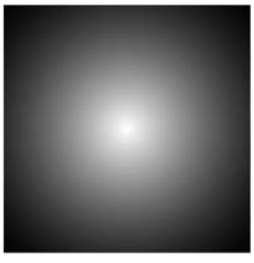


`10deg` `20deg` `30deg` `40deg` `50deg` `60deg` `70deg` `80deg` `90deg`

Радиальные градиенты могут быть созданы с помощью свойства `radial-gradient`. Смена цветов вокруг производит обратный эффект:



`radial-gradient(black, white)`

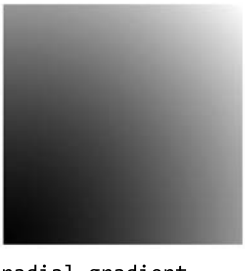


`radial-gradient(white, black)`

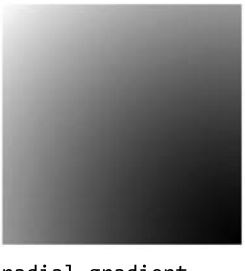
Аналогично линейным градиентам радиальные градиенты также могут начинаться в любом из четырех углов HTML-элемента:

`radial-gradient`
`(at top left, black, white)`

`radial-gradient`
`(at top left, white, black)`

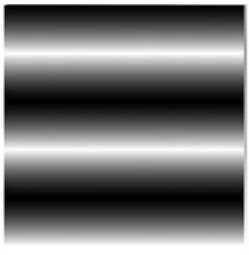


`radial-gradient`
`(at bottom left, black, white)`

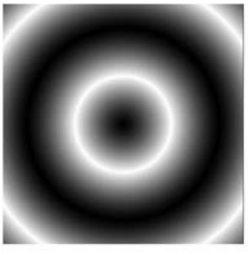


`radial-gradient`
`(at bottom right, white, black)`

Повторяющиеся узоры для линейных и радиальных градиентов могут быть созданы с помощью свойств `repeating-linear-gradient` и `repeating-radial-gradient` соответственно. Вы можете указать столько повторяющихся значений цвета в строке, сколько необходимо. Просто не забудьте разделить их запятой!

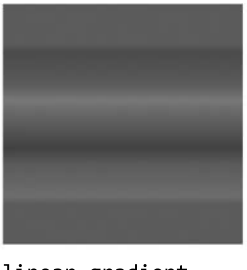


```
repeating-linear-gradient
(white 100px;
 black 200px;
 white 300px);
```



```
repeating-radial-gradient
(white 100px;
 black 200px;
 white 300px);
```

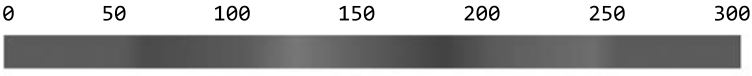
Наконец, самый усовершенствованный тип градиента можно создать с помощью ряда значений HSL. Значения HSL не имеют именованных или RGB-эквивалентов, они рассчитываются по шкале от 0 до 300. См. пояснение далее.



```
linear-gradient
  hsl(0,100%,50%),
  hsl(50,100%,50%),
  hsl(100,100%,50%),
  hsl(150,100%,50%),
  hsl(200,100%,50%),
  hsl(250,100%,50%),
  hsl(300,100%,50%)
```

```
linear-gradient
  hsl(0,100%,50%),
  hsl(50,100%,50%),
  hsl(300,100%,50%)
```

Вы можете выбрать любой цвет, используя значения от 0 до 300:



Мы уже рассмотрели примеры значений свойств, связанных с каждым градиентом. Поэкспериментируем со значениями и посмотрим, какие эффекты они оказывают на ваши элементы пользовательского интерфейса:

```
001 background: linear-gradient (yellow, red);
002 background: linear-gradient (black, white);
003 background: linear-gradient (to right, black, white);
004 background: linear-gradient (to left, black, white);
005 background: linear-gradient (to bottom right, black, white);
006 background: linear-gradient (90deg, black, white);
007 background: linear-gradient (
008 hsl (0, 100%, 50%),
009 hsl (50, 100%, 50%),
010 hsl (100, 100%, 50%),
011 hsl (150, 100%, 50%),
012 hsl (200, 100%, 50%),
013 hsl (250, 100%, 50%),
014 hsl (300, 100%, 50%));
015 background: radial-gradient (black, white);
016 background: radial-gradient (at bottom right, black, white);
017 background:
018 repeating-linear-gradient
019 (white 100px, black 200px, white 300px);
020 background:
021 repeating-radial-gradient
022 (white 100px, black 200px, white 300px);
```

13 Фильтры

Фильтры CSS изменяют внешний вид изображения (или любого HTML-элемента, имеющего графический вывод некоторых видов), регулируя его цветовые значения. Фильтры применяются с помощью свойства `filter`, принимающего функцию с одним аргументом.

Значение функций фильтра, таких как `blur` (размытие), `contrast` (контрастность), `brightness` (яркость) и т. д., можно указывать в процентах, в виде числа или значения в пикселах (px).

13.1. Фильтр blur()

Возможно, наиболее полезным из всех CSS-фильтров является эффект размытия.

```
001 .blur { filter: blur(100px); }
```

filter: blur(0);

filter: blur(100px);

Фильтр размытия довольно универсален, поскольку способен работать вместе с изображениями, содержащими прозрачные области, создавая множество возможностей для создания интересных визуальных эффектов и переходов пользовательского интерфейса. К сожалению, большинство других CSS-фильтров практически не используются (например, вы когда-нибудь видели фото с эффектом сепии на каких-либо сайтах?). Однако мы должны иметь о них представление.

13.2. Фильтр brightness()

```
001 .blur { filter: blur(100px); }
```

Регулирует яркость HTML-элемента (или изображения). Значения присваиваются в формате плавающей точки `0.0` — `1.0`, где `1.0` соответствует исходному содержанию изображения. Допускаются значения больше `1.0`. Они просто сделают ваше изображение ярче исходного.

13.3. Фильтр contrast()

Меняет контрастность изображения/элемента HTML. Уровень контрастности указывается в процентах.

```
001 .contrast { filter: contrast(120%); }
```

13.4. Фильтр grayscale()

Извлекает все цвета из картинки, делая на выходе черно-белое изображение. Значение задается в процентах.

```
001 .grayscale { filter: grayscale(100%); }
```

13.5. Фильтр opacity()

Добавляет прозрачность элементу (аналог свойства `opacity`).

```
001 .opacity { filter: opacity(50%); }
```

13.6. Фильтр hue-rotate()

Меняет цвета изображения в зависимости от заданного угла поворота в цветовом круге. Значение задается в градусах (от `0deg` до `360deg`).

```
001 .hue-rotate { filter: hue-rotate(180deg); }
```

13.7. Фильтр invert()

Инвертирует цвета. Значение указывается в процентах. Значение `50%` создает серое изображение.

```
001 .invert { filter: invert(100%); }
```

13.8. Фильтр saturate()

Управляет насыщенностью цветов: значение указывается в виде числа от `0` до `100`. Значение больше `100` обычно приводит к чрезмерно насыщенному изображению.

```
001 .saturate { filter: saturate(7); }
```

13.9. Фильтр sepia()

Эффект, имитирующий старину (похоже на старую фотографию).

```
001 .sepia { filter: sepia(100%); }
```

13.10. Фильтр drop-shadow()

Действует подобно свойству `box-shadow`.

```
001 .shadow { filter: drop-shadow(8px 8px 10px green); }
```

14 Фоновые изображения

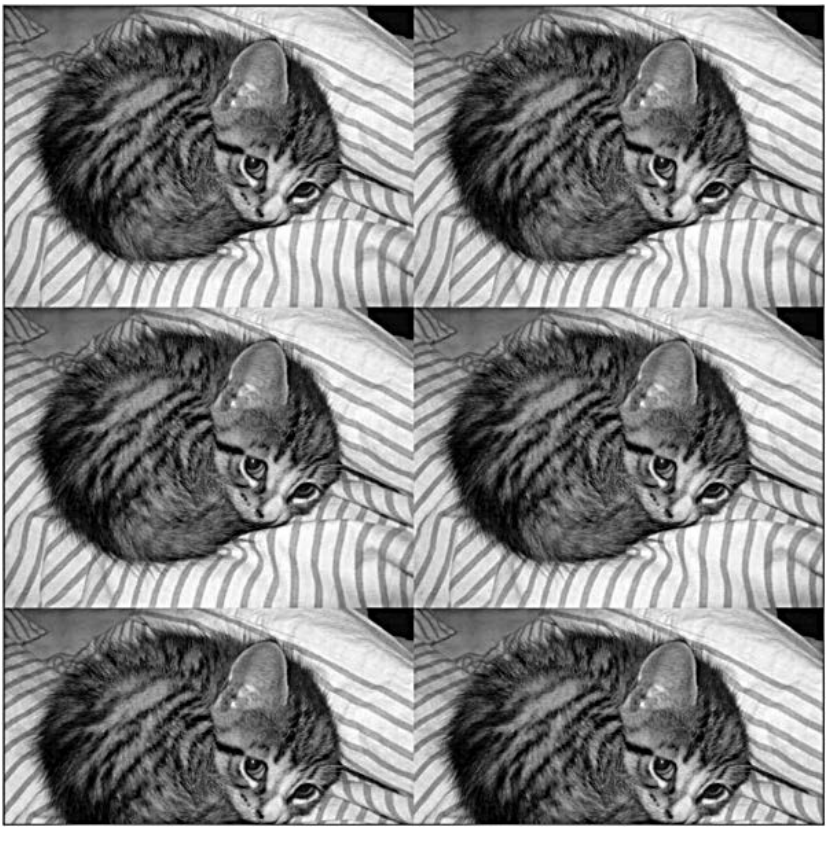
Так вы думаете, что знаете HTML-фоны? Ну, может, знаете, а может, и нет. Данная глава, надеюсь, познакомит читателя с общей картиной. Мы рассмотрим несколько свойств CSS, способных помочь изменить настройки фонового изображения для любого элемента HTML.

Пример использования свойства `background: url("image.jpg")` или `background-image: url("image.jpg")`:

Изображение, используемое в этой главе, — очаровательный котенок на полосатом фоне.

Если размеры элемента превышают размеры исходного изображения, то оно будет повторяться внутри тела данного элемента, повторяя

заполнение оставшихся сторон элемента содержимым изображения. Это почти как растягивание конечных обоев по всей области элемента.



Чтобы установить фоновое изображение для любого элемента, можно использовать следующие команды CSS:

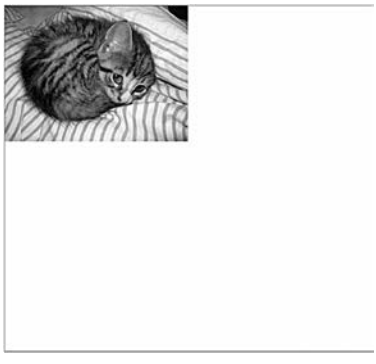
```
001 background: url("kitten.jpg");
```

или...

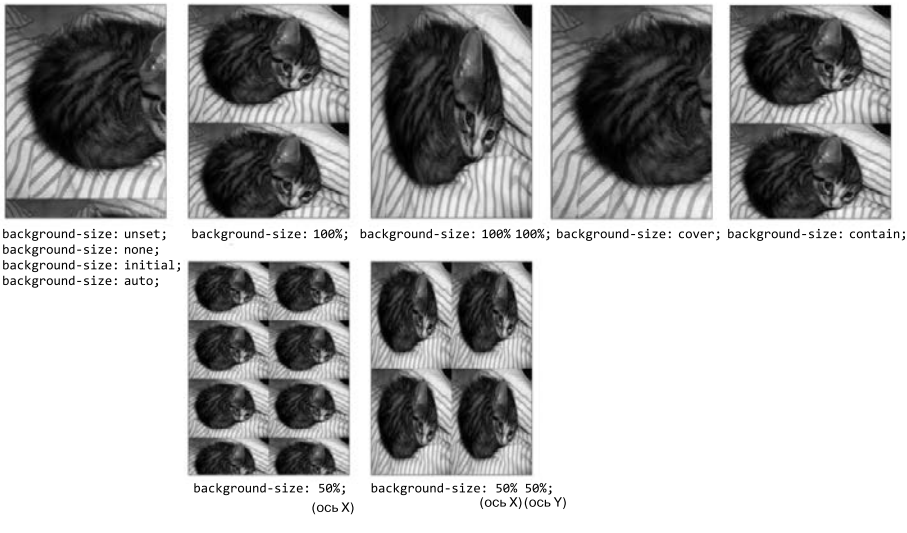
```
001 background-image: url("kitten.jpg");
```

Кроме того, можно задействовать внутренний CSS-код, поместив его между тегами `<style></style>`.

Посмотрим на тот же фон котенка... только с установленным значением `background-repeat: no-repeat`:



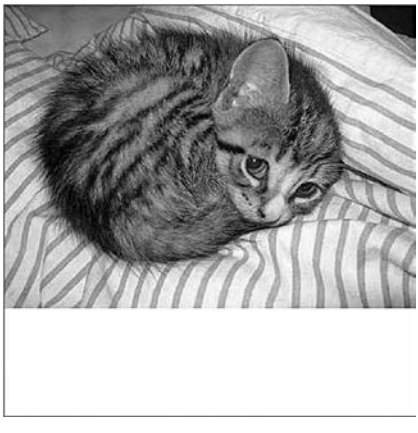
Далее показаны результаты, созданные с помощью свойства `background-size`. Слева направо приведены следующие примеры: (`unset` — `none` — `initial` — `auto`), которые приводят к значению по умолчанию.



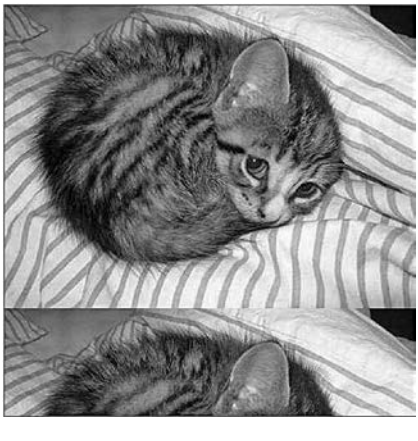
Значение `100%` растянет изображения по горизонтали, но не по вертикали. Значение `100% 100%` растянет изображение по всему доступному пространству. Значение `cover` растянет изображение по всему вертикальному пространству элемента, обрежет все в горизонтальном

направлении, как при переполнении. Значение `contain` гарантирует, что изображение растягивается по горизонтали по ширине элемента, и, сохраняя исходную пропорцию, растягивает его по вертикали столько, сколько потребуется, повторяя изображение до тех пор, пока оно не заполнит нижнюю часть элемента.

Объединяя свойства `background-repeat: no-repeat` и `background-size: 100%`, можно растянуть изображение только по горизонтали, по всей ширине элемента:



Что, если вы хотите повторить фон по вертикали, но держать его растянутым по ширине? Все возможно, просто удалите значение `no-repeat` из предыдущего примера. В конечном итоге получится вот что:



Как было упомянуто выше, такой фоновый метод HTML/CSS используется для сайтов, чье содержимое растягивается вертикально на большую область пространства. Я думаю, что одна из итераций сайта Blizzard задействовала его ранее. Иногда вы захотите обрезать содержимое и сделать статичным. В других случаях пожелаете, чтобы оно продолжалось все время по вертикали. Все будет зависеть от вашего видения макета.

Бывает необходимо растянуть изображение поперек, чтобы установить ограничивающую рамку элемента. Однако это зачастую приводит к некоторым искажениям. CSS автоматически растянет изображение в соответствии с некоторым автоматически рассчитанным процентным значением.

Нет необходимости говорить, что такой эффект будет наблюдаться только в том случае, если HTML-элемент и размер изображения не совпадают.

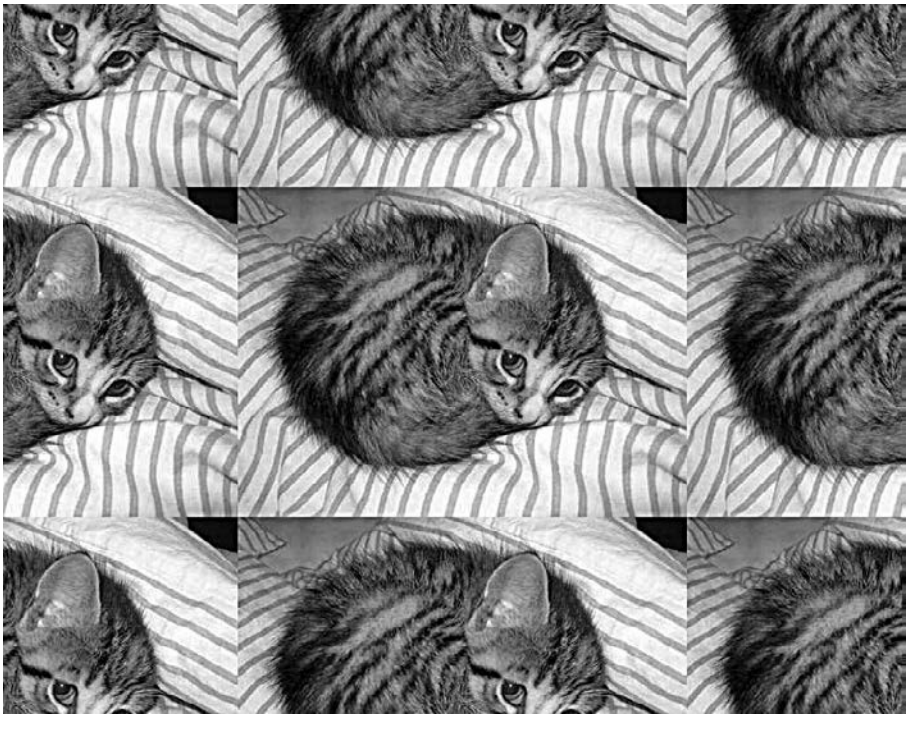
Как упоминалось выше: чтобы растянуть изображение, следует установить свойство `background-size: 100% 100%`. Обратите внимание: значение `100%` повторяется дважды. Первое значение растягивает изображение по вертикали, а второе делает то же самое по горизонтали. Вы можете использовать значения в диапазоне от `0` до `100%`, хотя я не вижу в этом необходимости.

14.1. Указание нескольких значений

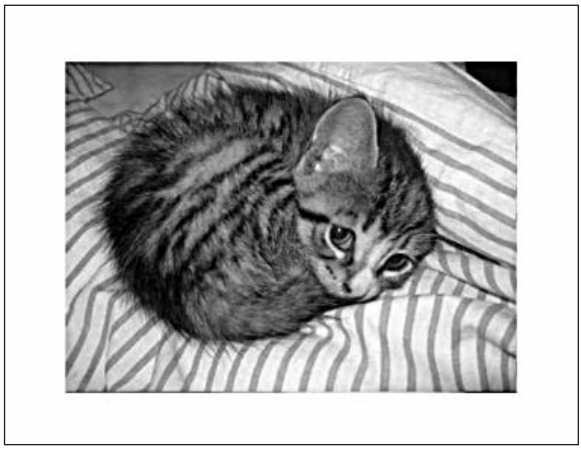
Всякий раз, когда вам нужно указать в HTML несколько значений, они часто разделяются пробелом. Вертикальные координаты (*ось Y*, или *высота*) всегда указываются первыми. Иногда значения разделяются запятой. Например, при необходимости указать несколько фонов они обычно разделяются запятой, а не пробелом. (В последнем разделе я покажу, как это делается.)

14.2. Свойство background-position

Далее приведен пример `background-position: center center`:



Вы можете центрировать изображение, но потерять повторяемость шаблона, указав значение `no-repeat` для свойства `background-repeat`.



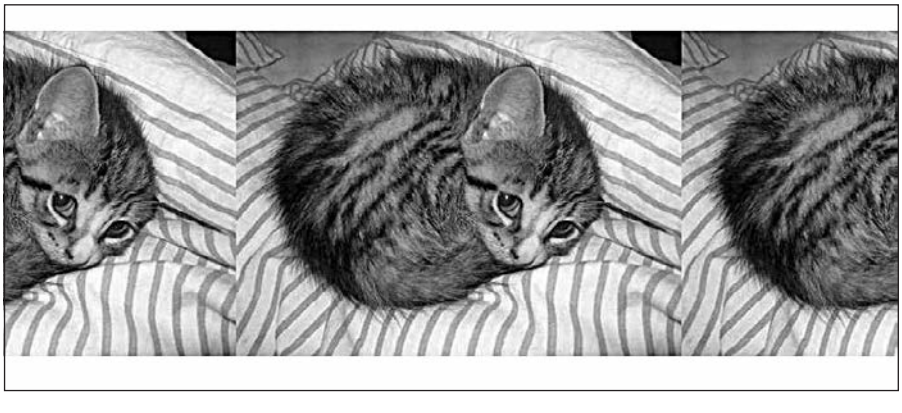
Центрирование изображения:

```
001 background-position: center center;
```

Отключение повторения:

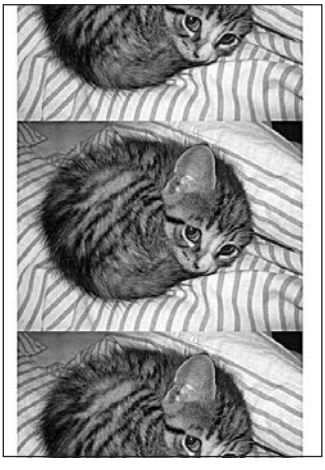
```
001 background-repeat: no-repeat;
```

Вы можете повторить изображение только по оси *X* (по горизонтали), используя значение `repeat-x`:



Вы можете легко отцентрировать и повторить изображение только по горизонтали, указав `repeat-x` в качестве значения свойства `background-repeat`.

Для того же эффекта, но по оси *Y* служит свойство значение `repeat-y`:



Как и любым другим свойством CSS, вы должны манипулировать значениями для достижения желаемых результатов. Я думаю, мы рассмотрели почти все, касающееся фонов. За исключением одной последней темы...

14.3. Фон из нескольких изображений

Можно добавить более одного фона к одному и тому же HTML-элементу. Процесс довольно прост.

Рассмотрим изображения, хранящиеся в двух отдельных файлах.

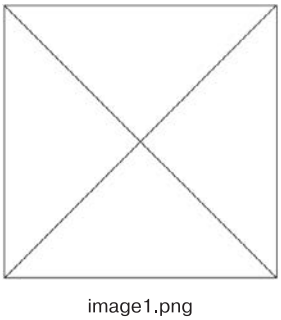


image1.png

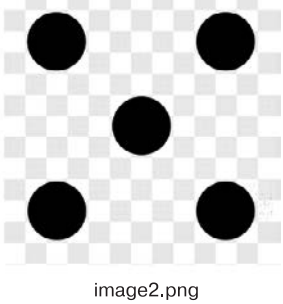


image2.png

Инструмент Magic Eraser (Волшебный ластик) в программе Photoshop можно выбрать на панели инструментов:

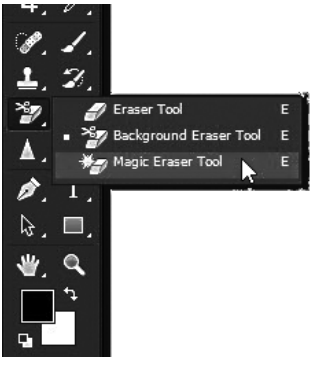


Рисунок в виде шахматной доски справа на рисунке используется только для обозначения прозрачности. Белые и сероватые квадраты не являются реальной частью самого изображения. Это та самая прозрачная область, которая обычно видна в программном обеспечении для цифровых манипуляций.

Когда изображение справа помещается поверх других HTML-элементов или изображений, клетчатая область не будет блокировать данный контент внизу. И в этом заключается вся идея множественных фонов в HTML.

14.4. Прозрачность фона

Чтобы полностью использовать преимущества множественных фонов, одно из фоновых изображений должно иметь прозрачную область. Как это сделать? В нашем примере второе изображение, `image2.png`, содержит пять черных точек на прозрачном фоне, обозначенном клетчатым узором.

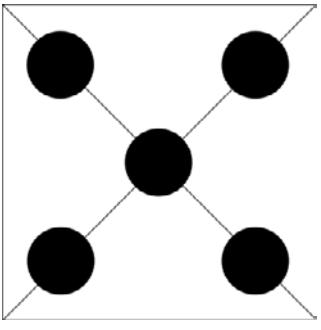
Как и многие другие свойства CSS, которые принимают несколько значений для настройки множественных фонов, они предоставляют для свойства фона набор значений, разделенных запятой.

14.5. Множественные фоны

Чтобы назначить несколько *разделенных на слои* фоновых изображений, для одного и того же HTML-элемента можно использовать следующий код CSS:

```
001 body { background: url("image2.png"), url("image1.png"); }
```

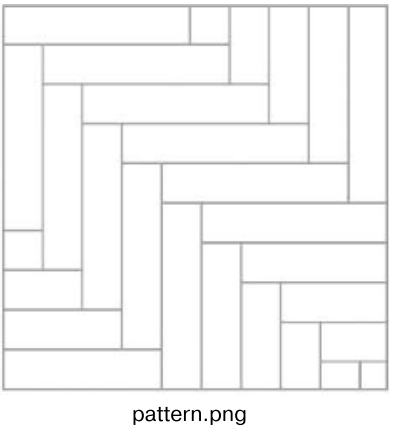
Очень важен порядок, в котором вы присваиваете изображения для свойства `url` фона. Обратите внимание: самое верхнее изображение всегда указывается первым. Вот почему мы начинаем с `image2.png`.



В данном примере мы рассмотрели множественные фоны в теории на субъективном элементе `div` (или аналогичном) в виде квадрата.

Возьмем другой пример.

puppy.png



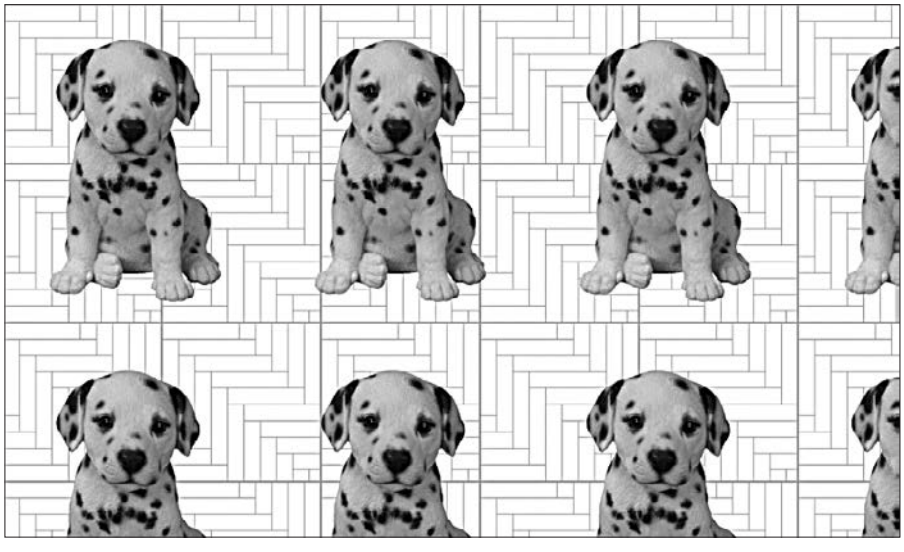
pattern.png

Обратите внимание: изображение `puppy.png` будет первым элементом в списке, разделенном запятыми. Это изображение, которое мы хотим наложить поверх всех других изображений в списке.

Объединяем два изображения:

```
001 body { background: url('puppy.png'), url('pattern.png')
```

Получаем следующий результат.



Существуют и другие свойства фона, которые также принимают списки, разделенные запятыми. Это почти все остальные свойства, связанные с фоном, кроме `background-color`.

Таким же образом можно присвоить иные параметры для каждого отдельного фона с помощью других свойств фона, показанных ниже:

```
001 background
002 background-attachment
003 background-clip
004 background-image
005 background-origin
006 background-position
007 background-repeat
008 background-size
```

Следующее свойство нельзя использовать со списком по очевидным причинам:

```
001 background-color
```

Что бы это значило — обеспечить несколько значений цвета для фона? Всякий раз, когда задается свойство `background-color`, оно

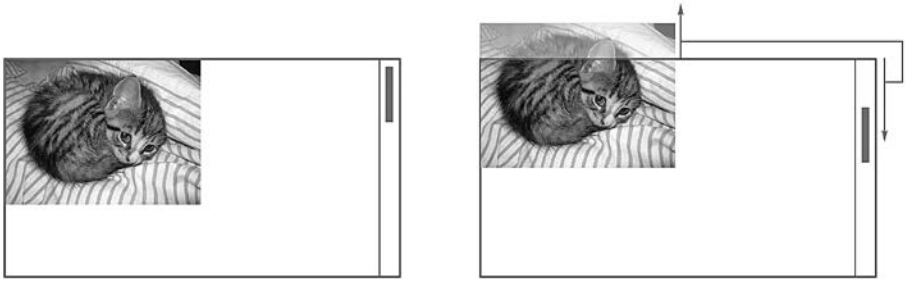
обычно заполняет всю область сплошным цветом. Но множественные фоны требуют прозрачности хотя бы в одном фоне. Таким образом, это свойство не может быть использовано в случае множественных фонов.

Но это еще не все, что можно сказать о фоновых изображениях. Закончим наше обсуждение, рассмотрев другие случаи.

14.6. Свойство background-attachment

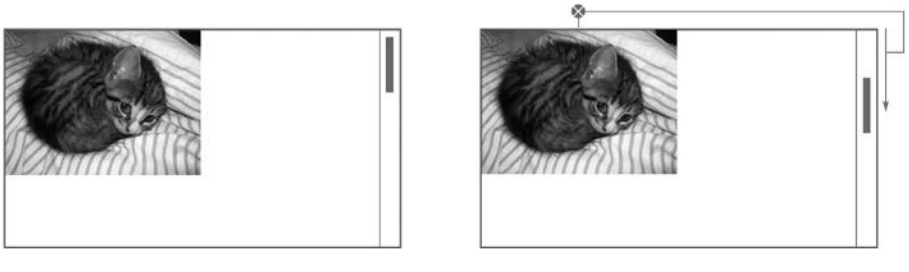
Можно определить поведение фонового изображения относительно полосы прокрутки.

`background-attachment: scroll`:

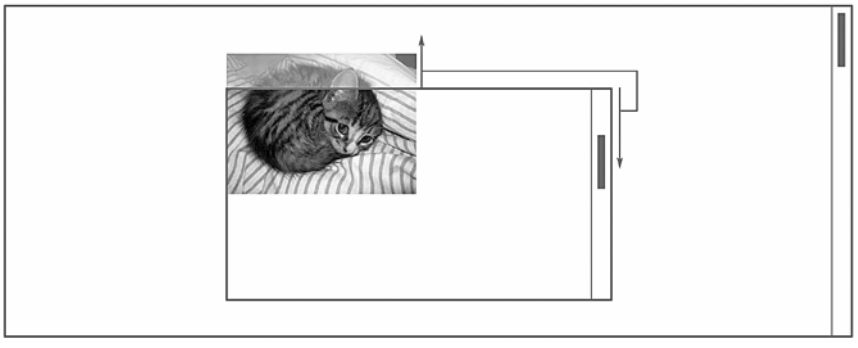


Фиксированные фоны не реагируют на полосу прокрутки.

`background-attachment: fixed`:

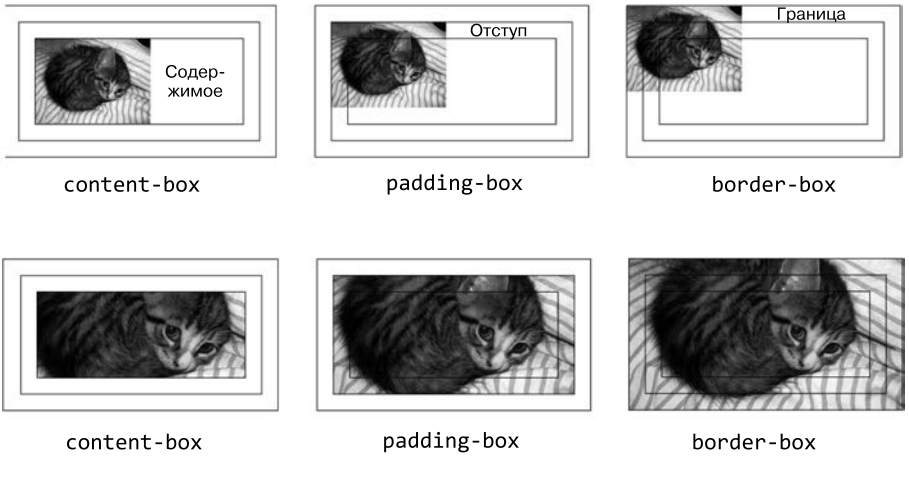


`background-attachment: scroll`:



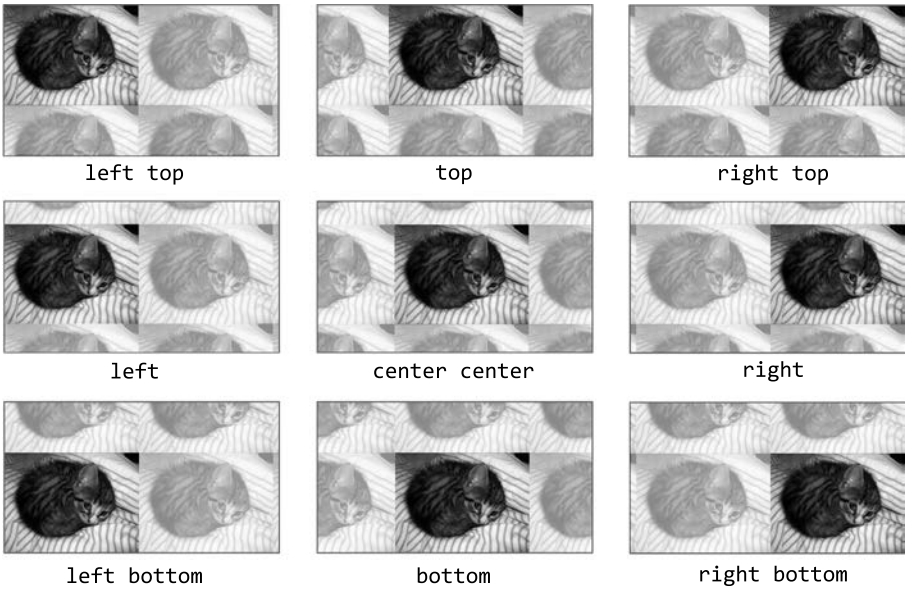
14.7. Свойство background-origin

Данное свойство определяет размер области, которая будет использоваться фоновым изображением, на основе *блочной модели CSS*.

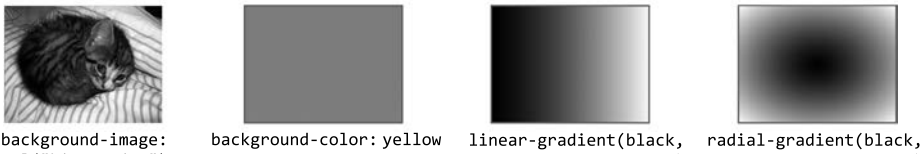


Свойствам `background-position-x` и `background-position-y` присваиваются следующие значения для создания любого из шаблонов

расположения фона: `left top` — `top` — `right top` — `left` — `center center` — `right` — `left bottom` — `bottom` — `right bottom`.



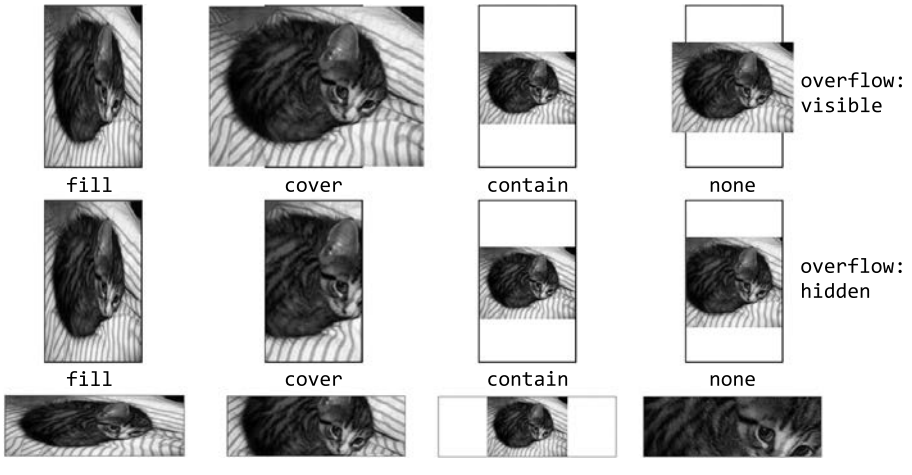
И наконец... в дополнение к изображениям свойство `background` также может определять либо *сплошной цвет*, либо *линейный градиент*, либо *радиальный градиент*.



Примеры других возможных значений, присваиваемых свойству `background`. Обратите внимание: описанию линейных и радиальных градиентов посвящена целая глава этой книги.

15 Свойство object-fit

Некоторые функциональные возможности фонов были заменены немного другим решением для подбора изображений, основанным на свойстве подгонки объекта. Присваивая различные значения, вы можете достичь любого из следующих результатов.



Здесь свойство `object-fit` показывает практически все возможные случаи помещения объекта в родительский контейнер. Обратите внимание: похожее на `background`, свойство `object-fit` работает с нефоновыми изображениями (созданными с использованием элемента `img`), видео и другими «объектами», а не с фоновыми изображениями.

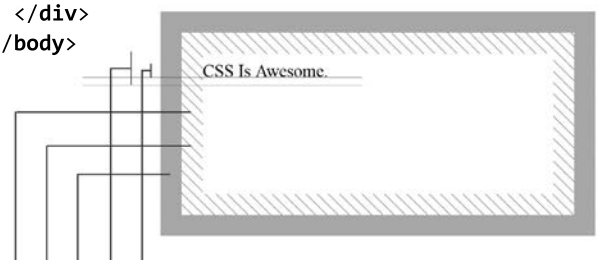
Здесь третья строка, как и вторая, имеет значение `overflow: hidden`, но в нашем примере размеры фактического HTML-элемента были показаны для демонстрации того, что он дает слегка отличающиеся результаты, когда преобладают вертикальные или горизонтальные размеры.

16 Границы

Границы CSS — это гораздо больше, чем кажется на первый взгляд. В частности, вы хотите узнать, как радиус границы (только если указаны значения для осей *X* и *Y*) влияет на другие углы того же элемента. Но прежде, чем двигаться вперед, рассмотрим свойство `border`.

Вы можете легко получить доступ ко всем тем же свойствам CSS через JavaScript. Например, чтобы получить доступ ко всем свойствам CSS, просто возьмите объект с помощью кода `document.getElementById("container")`. Свойства привязаны к свойству объекта `element.style`:

```
<body style = "margin: 30px;">
  <div id = "container">
    <div style = "width: 100%; height: 100%;
         background: white;">CSS Is Awesome.</div>
  </div>
</body>
```

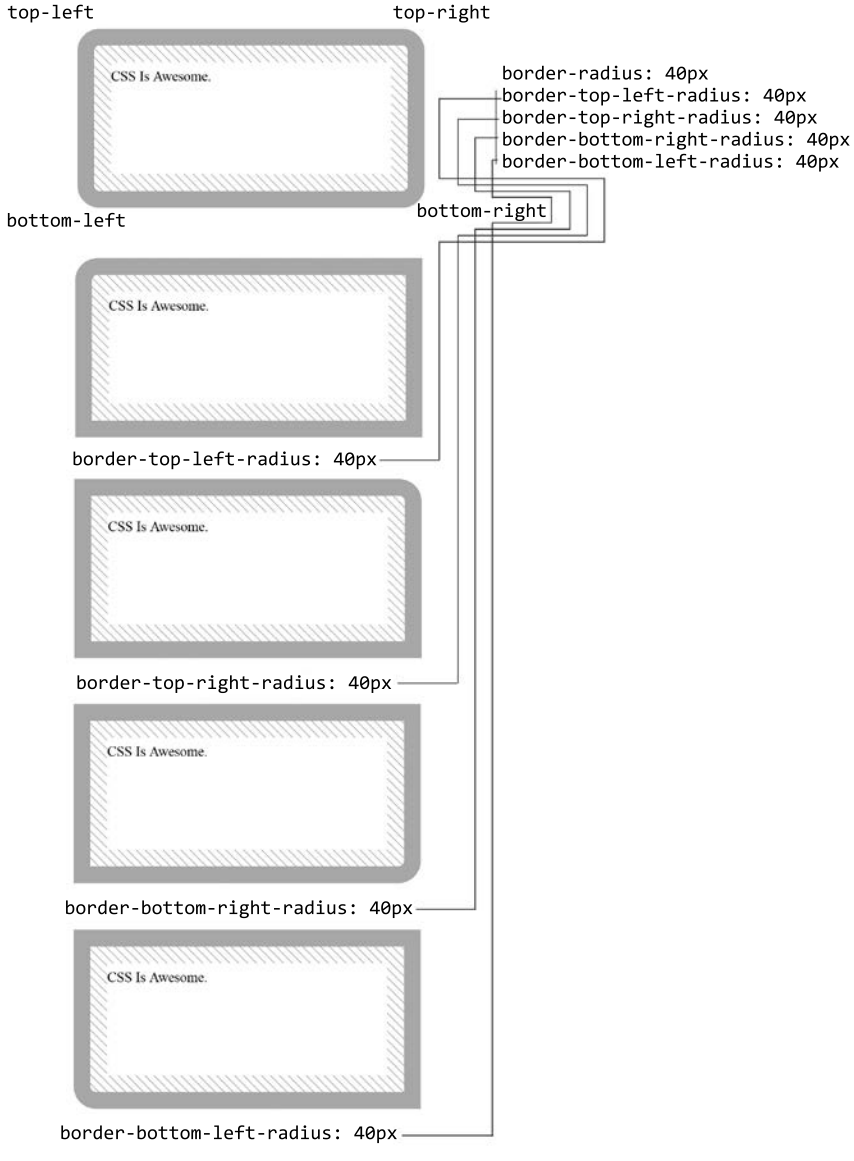


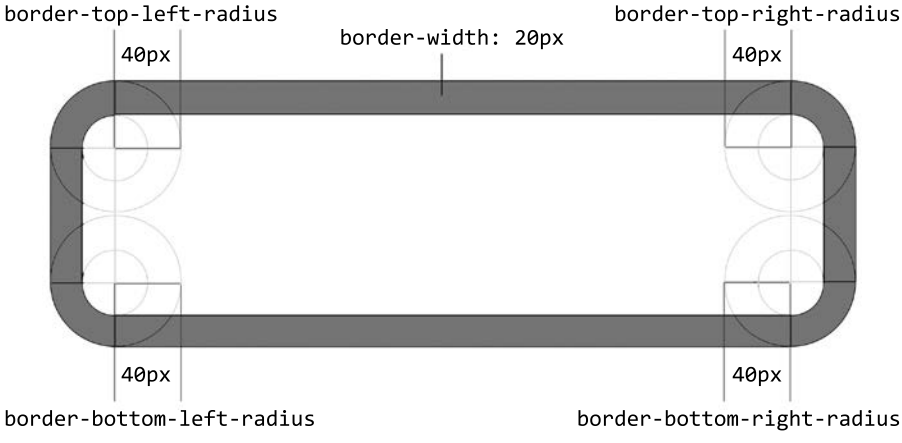

```
var x = document.getElementById("container");
x.style.fontSize = "25px";
x.style.lineHeight = "50px";
x.style.width = "500px";
x.style.height = "200px";
x.style.border = "30px solid silver";
x.style.background = "url(diag.png)";
x.style.padding = "30px";
```

Границы могут быть установлены со всех сторон одновременно благодаря свойству `border`.

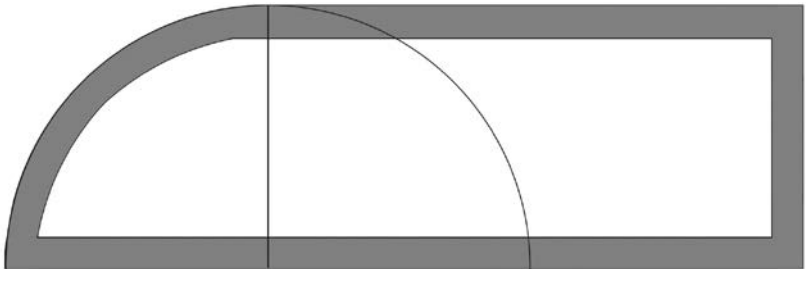
```
001 border: 5px solid gray;
```

Вы также можете установить кривизну в каждом из четырех углов элемента с помощью свойства `border-radius`, указав радиус круга.

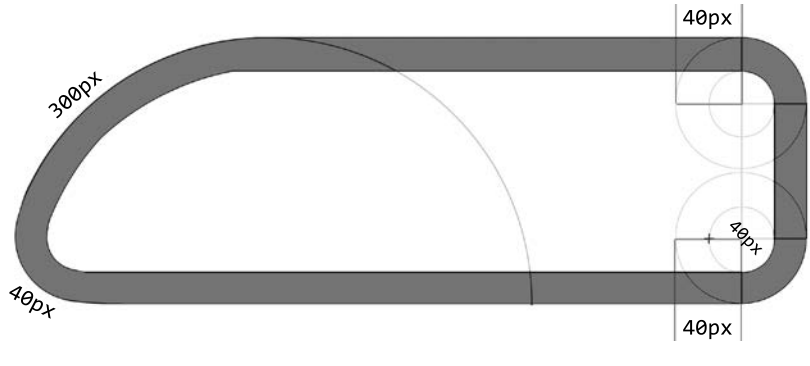




Использование значения, равного или превышающего размер стороны элемента, к которой применяется радиус границы, будет ограничено наибольшим радиусом, подходящим в этой области:



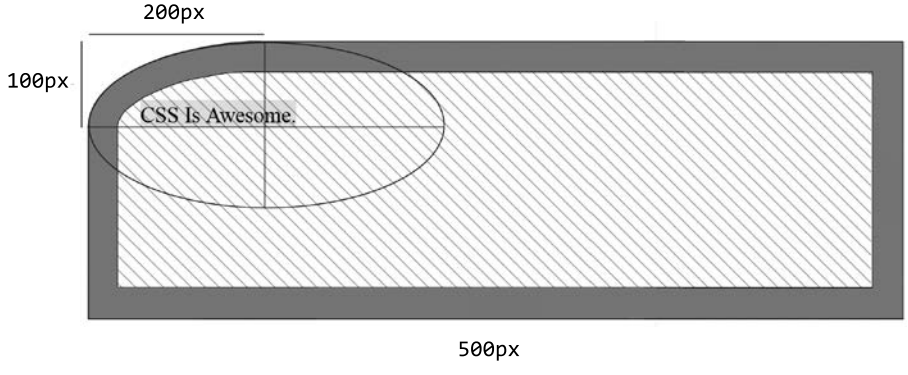
Свойства `border-top-left-radius: 300px`; `border-top-left-radius: 40px`; `border-bottom-left-radius: 40px`; `border-bottom-right-radius: 40px`:



16.1. Эллиптический радиус границы

Даже после долгой работы с CSS я не заметил, что свойство `border-radius` можно использовать для создания эллиптических границ. Но это действительно правда. Результаты эллиптических кривых не всегда легко предсказуемы, как в случае с осевыми значениями радиуса.

Эллиптическая граница создается с помощью двух параметров `border-top-left-radius: 200px 100px`:



Эллиптический радиус задается путем указания двух разделенных пробелом значений для каждой оси в отношении одного и того же угла.

При использовании эллиптического радиуса с чрезвычайно большими значениями кривая одного угла может влиять на кривую соседних углов, особенно если один угол с меньшим значением радиуса. Но хорошо то, что данный уровень неопределенности открывает пространство для творческих экспериментов. Вам просто нужно поиграть с разными значениями, чтобы достичь конкретного эффекта или искомой кривой.

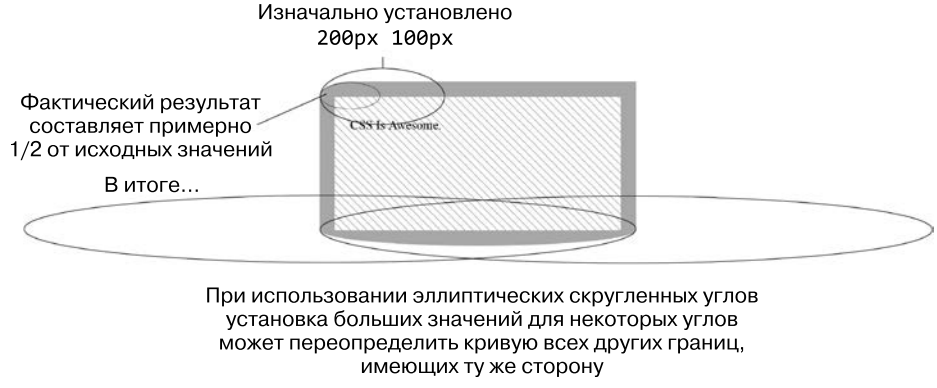
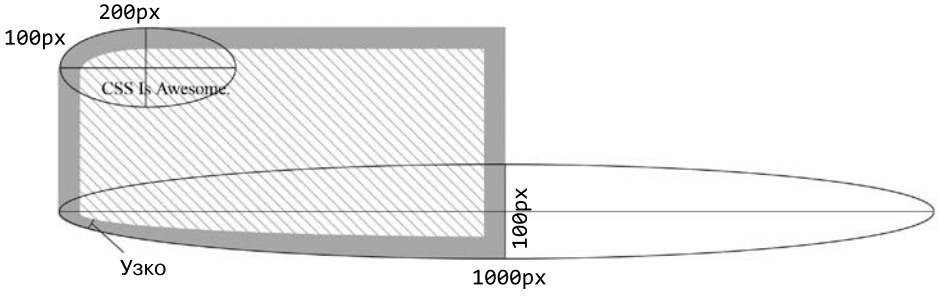
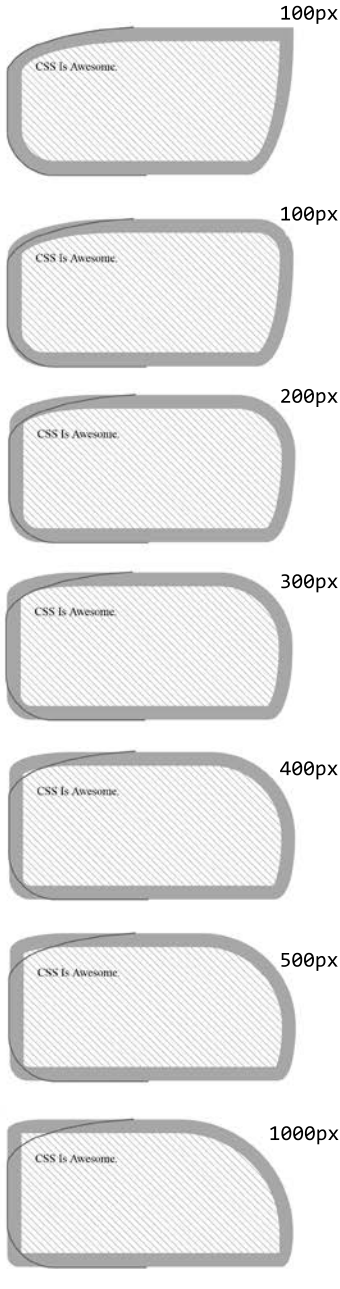


Иллюстрация принципа применения больших значений к эллиптическим дугам:

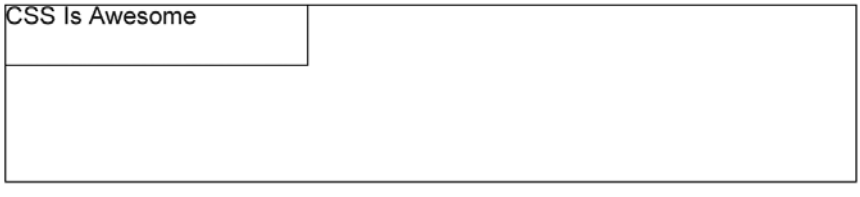


Рассмотрите пример на следующей странице. Мы меняем только значение кривой в верхнем правом углу. Однако обратите внимание: все скругленные углы элемента зависят друг от друга по коду — даже те, значения которых мы не меняем явно.


100px
CSS Is Awesome.
100px
CSS Is Awesome.
200px
CSS Is Awesome.
300px
CSS Is Awesome.
400px
CSS Is Awesome.
500px
CSS Is Awesome.
1000px
CSS Is Awesome.

17 2D-трансформации

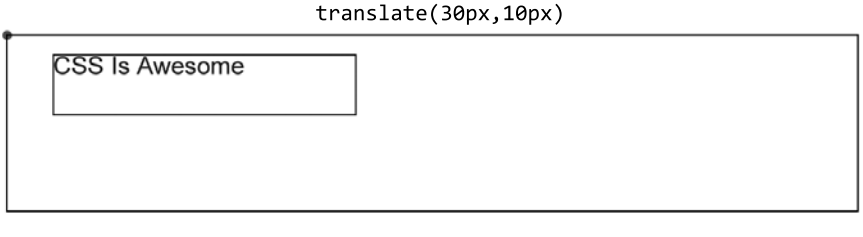
2D-трансформации позволяют перемещать, масштабировать или вращать HTML-элемент.



Это оригинальный образец (родительский и дочерний элементы). Мы будем использовать этот простой пример HTML-элемента для демонстрации 2D-трансформаций.

17.1. Свойство translate

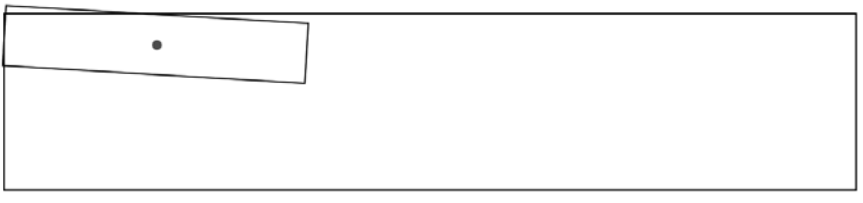
Вместо свойств `top` и `left` мы можем использовать свойство `transform: translate(30px, 10px)` для перемещения элемента по осям *X* и *Y*:



17.2. Свойство rotate

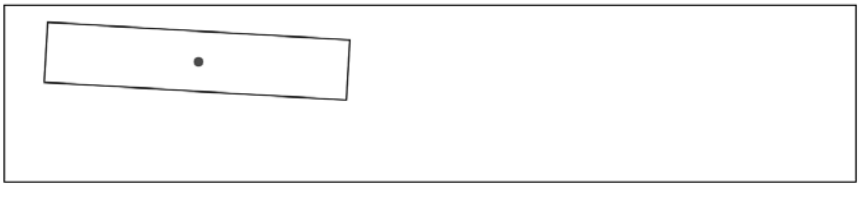
Далее продемонстрировано вращение элемента вокруг его центра с помощью свойства `rotate(`*`угол`*`)`, где *`угол`* — это значение в диапазоне от 0 до 360 градусов с добавлением единиц `deg`.

`rotate(5deg)` будет вращать элемент вокруг своего центра:

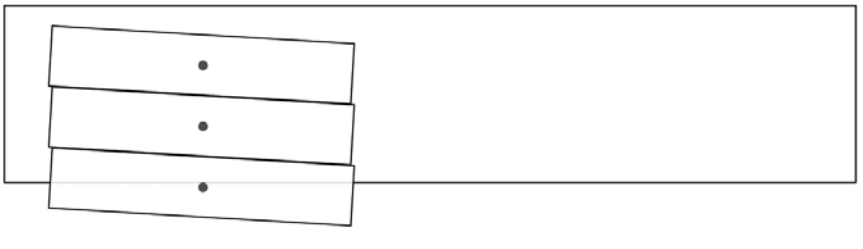


Возможно перемещение и вращение элемента:

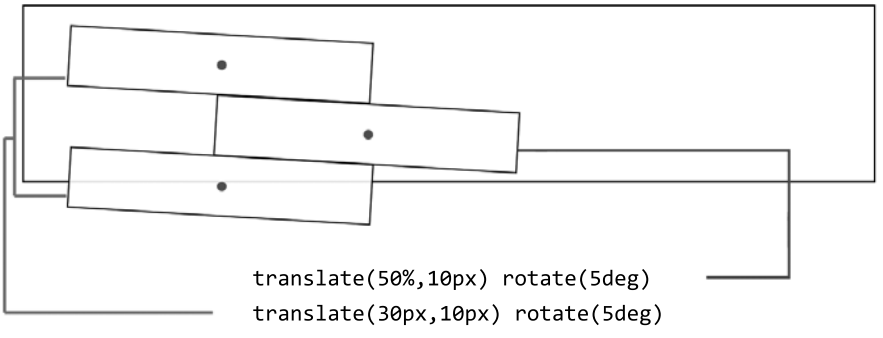
```
translate(30px,10px) rotate(5deg)
```



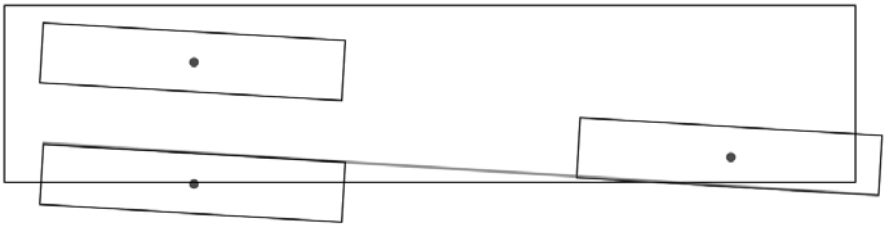
На следующем рисунке три элемента со свойствами `display: block; position: relative;` устанавливаются под одним и тем же углом. Относительное положение всех последовательных элементов сохраняется:



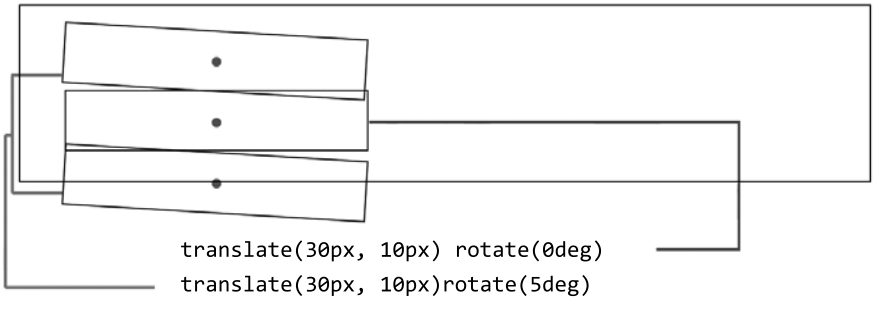
Трансформация перемещения может занять процент от размера элемента. Ниже показан перевод в проценты в зависимости от размеров элемента:



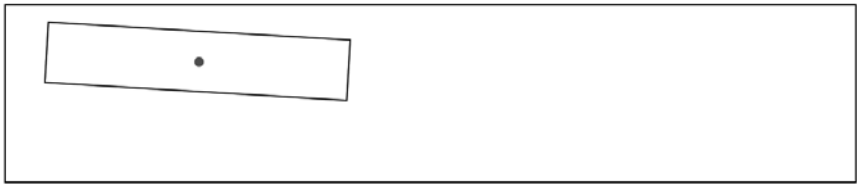
Относительные элементы сохраняют свое положение в документе даже после вращения:



Вращение элемента между другими не влияет на их положение. Края будут перекрываться:



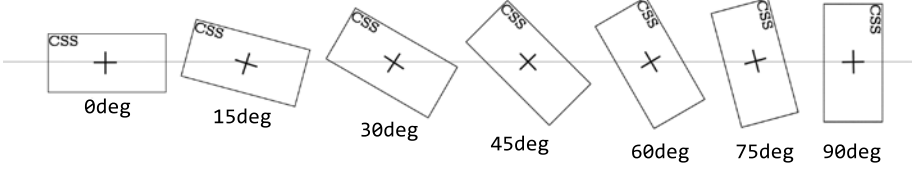
Порядок перемещения и поворота не имеет значения:



`translate(30px, 10px) rotate(5deg)` так же как
`rotate(5deg) translate(30px, 10px)`

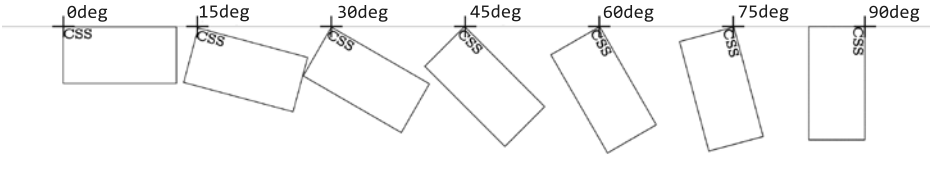
По умолчанию элемент будет вращаться вокруг средней точки:

`transform-origin: 50% 50%` (по умолчанию)

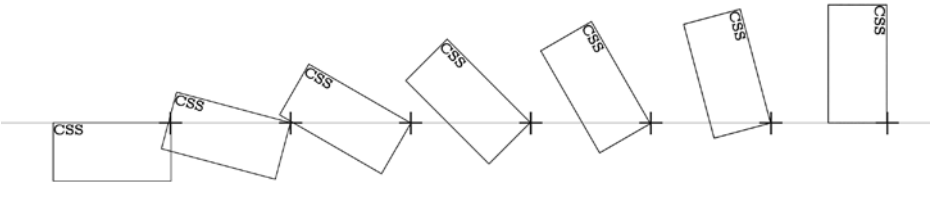


17.3. Свойство transform-origin

Начало вращения подвижного элемента с использованием свойства `transform-origin: 0 0`:

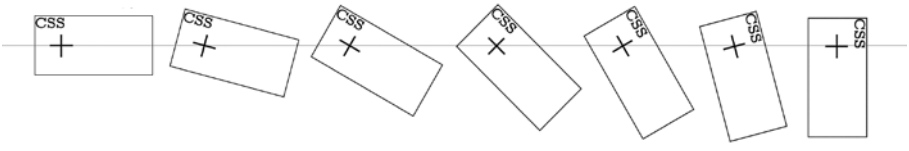


`transform-origin: 100% 0`



Точка вращения не обязательно должна быть в середине или в углах элемента. Она может располагаться где угодно.

```
transform-origin: 25% 50%
```

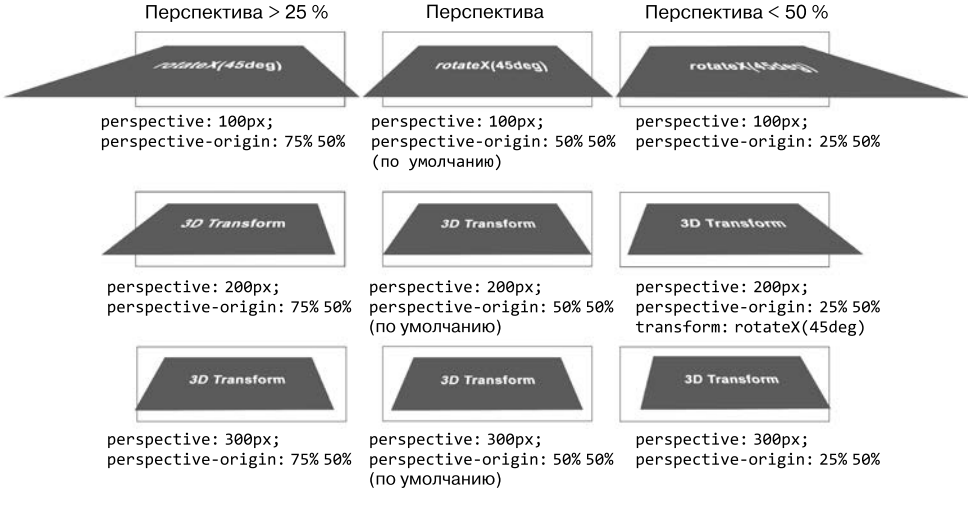


18 3D-трансформации

3D-трансформации могут преобразовать обычные HTML-элементы в трехмерные, добавляя перспективу.

18.1. Свойство rotateX

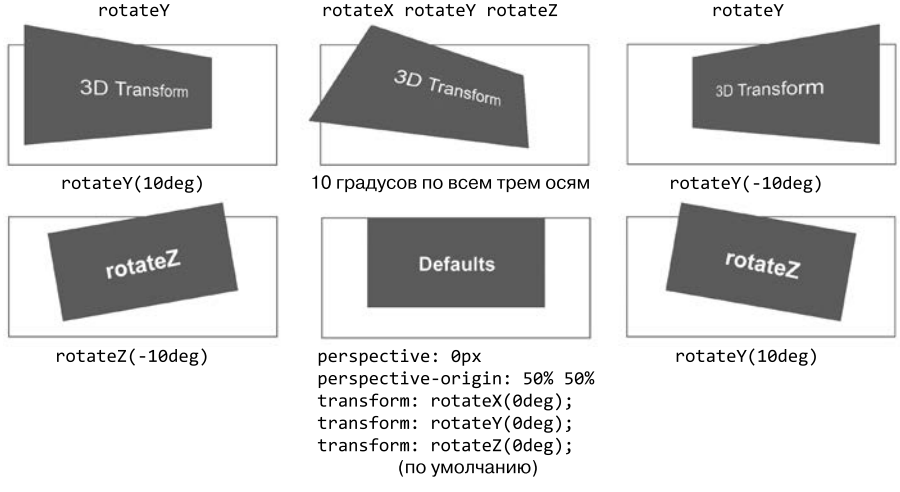
Повернем элемент по оси *X*, используя свойство `transform: rotateX`.



Каждая строка в данном примере показывает, что происходит с HTML-элементом, когда его перспектива изменяется с 100 на 200 пикселов, а затем на 300 сверху вниз с использованием свойства `perspective`. Свойство `perspective-origin` также служит для изображения уклона, созданного при смещении источника.

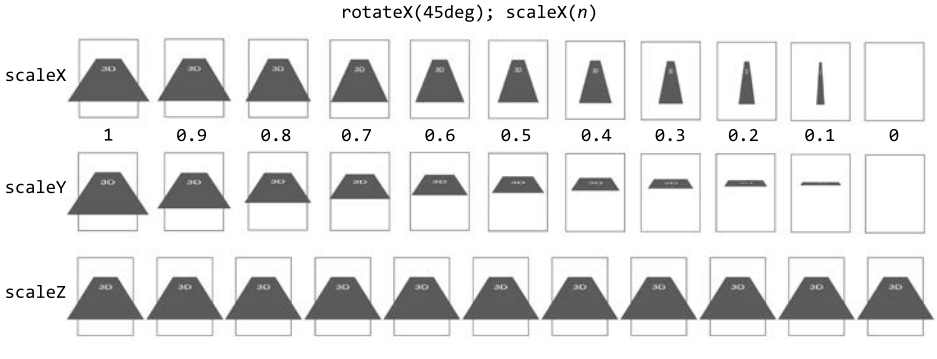
18.2. Свойства rotateY и rotateZ

Вращение элемента по осям *Y* и *Z* дает следующие результаты:



18.3. Свойство scale

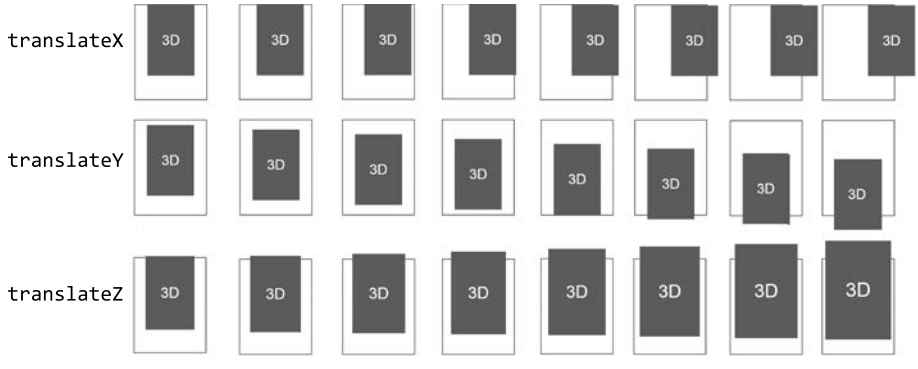
Масштабирование элемента либо уменьшает, либо увеличивает его относительный размер на любой из трех осей.



Как видите, аналогичным образом можно «масштабировать» элемент по любой из трех осей. Масштабирование по оси *Z* не меняет внешний вид элемента, когда перспектива не установлена.

18.4. Свойство translate

Можно перемещать элемент в трех измерениях. Пример ниже показывает, что происходит, когда элемент перемещается по оси *X*, *Y* или *Z*. Обратите внимание: камера обращена вниз по отрицательной оси *Z*. Таким образом, масштабирование элемента по оси *Z* вверх сделает его «ближе» к изображению. Другими словами, его размер будет увеличиваться по мере приближения к камере.



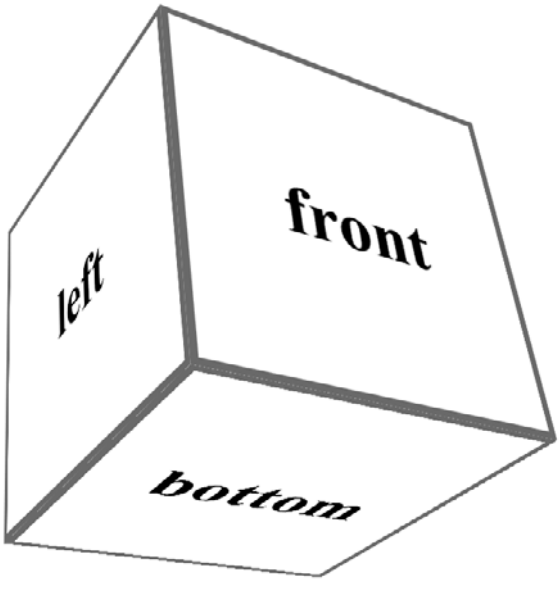
Далее изображена «матрица» 4 × 4. В книге не описано, как работают 3D-матрицы. Но в основном они изменяют перспективу. Часто используются в 3D-видеоиграх для настройки вида камеры, чтобы смотреть на главного героя или «зафиксироваться» на движущемся объекте.

X	*Y*	*Z*	
m_1	m_2	m_3	m_4
m_5	m_6	m_7	m_8
m_9	m_{10}	m_{11}	m_{12}
m_{13}	m_{14}	m_{15}	m_{16}

```
transform: matrix(m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,m16)
```

18.5. Создание 3D-куба

Вспомним, что мы знаем о 3D-трансформациях в CSS, и создадим трехмерный куб из шести HTML-элементов. Каждый из HTML-элементов перемещен на половину своей ширины и повернут на 90 градусов во всех направлениях:

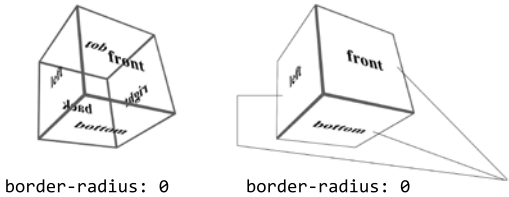


Это просто шесть HTML-элементов, каждый из которых имеет уникальный класс и 3D-трансформации:

```
<div class="view">
    <div class="cube">
    <div class="face front">front</div>
    <div class="face back">back</div>
    <div class="face right">right</div>
    <div class="face left">left</div>
    <div class="face top">top</div>
    <div class="face bottom">bottom</div>
  </div>
</div>
```

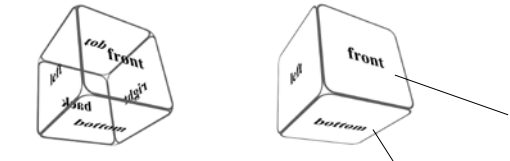
```
.view{
  width: 200px;
  height: 200px;
  perspective: 300px;
}
```

Вращая каждую грань вокруг гипотетического центра куба, мы можем построить наш 3D-объект:



border-radius: 0
backface-visibility:
visible

border-radius: 0
backface-visibility:
hidden



border-radius: 10px
backface-visibility:
visible

border-radius: 10px
backface-visibility:
hidden

```
.cube{
  width: 100%;
  height: 100%;
  position: relative;
  transform-style: preserve-3d;
  transform: translateX(50px)
             translateY(50px);
}
.face{
  position: absolute;
  width: 100px;
  height: 100px;
  background: transparent;
  border: 2px solid red;
  text-align: center;
  line-height: 100px;
}
.front { transform: rotateY(0deg)
                    translateZ(50px); }
.right { transform: rotateY(90deg)
                    translateZ(50px); }
.back  { transform: rotateY(180deg)
                    translateZ(50px); }
.left  { transform: rotateY(-90deg)
                    translateZ(50px); }
.top   { transform: rotateX(90deg)
                    translateZ(50px); }
.bottom{ transform: rotateX(-90deg)
                    translateZ(50px); }
```

Обратите внимание: свойству `backface-visibility` присвоено значение `hidden`, чтобы скрыть элементы, направленные в сторону от камеры. Благодаря этому наш куб кажется твердым.

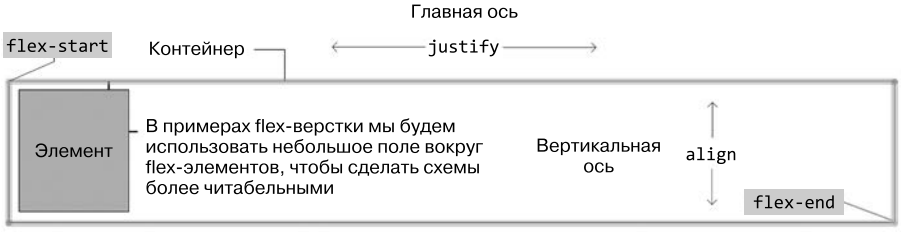
19 Flex-верстка

Flex-верстка определяет способность гибкого элемента растягиваться или сжиматься для заполнения собой доступного пространства.

19.1. Свойство display: flex

В отличие от многих других свойств обычных CSS во flex-верстке у вас есть основной контейнер и вложенные в него элементы.

Некоторые flex-свойства CSS используются лишь для родительского элемента. Остальные — только в отдельных случаях.



Можно представить flex-элемент как родительский контейнер с помощью свойства `display: flex`. Внутри данного контейнера размещаются элементы. Каждый контейнер имеет точки начала и конца `flex-start` и `flex-end`, что и показано на рисунке.

19.2. Главная и перекрестная оси

Хотя список элементов изображен линейно, flex-верстка требует от вас внимания к строкам и столбцам. По этой причине у нее две координатные оси. Горизонтальная ось называется *главной*, а вертикальная — *поперечной*.

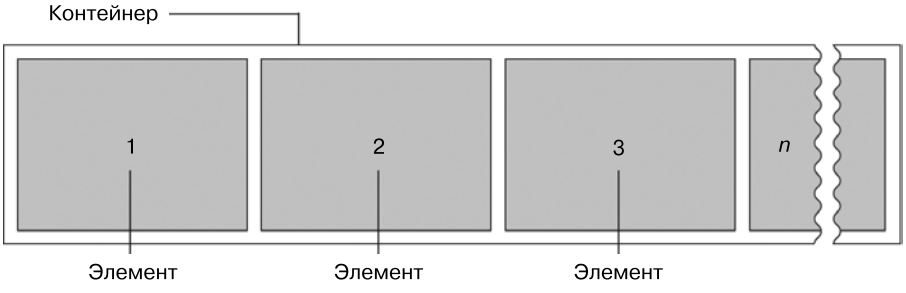
Для управления шириной контента и промежутками между ними, растягивающимися горизонтально по главной оси, вы будете использовать свойство `justify`. Для управления вертикальным поведением элементов служит свойство `align`.

Вот Flex-элементы, равномерно распределенные по главной оси:

Если у вас есть три столбца и шесть элементов, то flex-верстка автоматически создаст вторую строку для размещения остальных элементов:

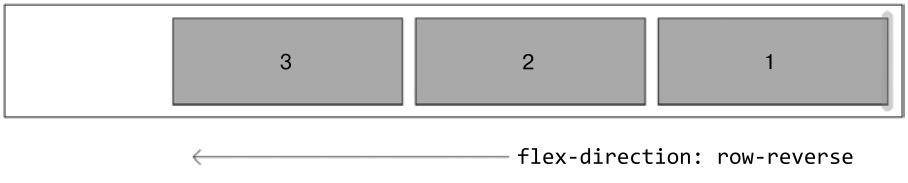
При наличии более шести элементов в списке будет создано еще больше строк. Рассмотрим свойства и значения, чтобы быстро выполнить эту задачу.

То, как строки и столбцы распределяются внутри родительского элемента, определяется свойствами `flex-direction`, `flex-wrap` и некоторыми другими (их мы рассмотрим далее). На следующей схеме у нас есть произвольное количество элементов, расположенных внутри контейнера. По умолчанию элементы растягиваются слева направо. Тем не менее исходную точку можно изменить.



19.3. Свойство flex-direction

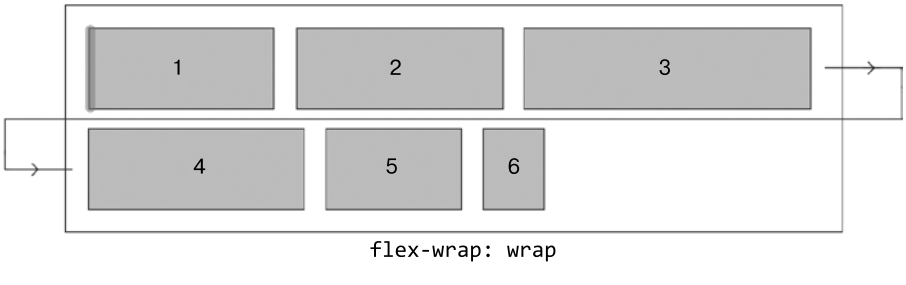
Можно задать направление потока элементов:



Свойство `flex-direction: row-reverse` меняет направление потока списка элементов. По умолчанию используется `row`; это значит, что вы должны двигаться слева направо, как и следовало ожидать!

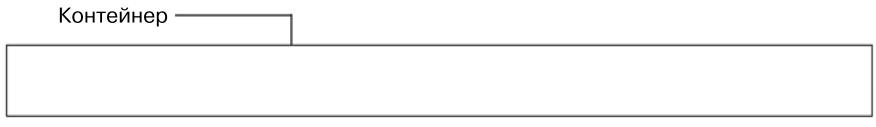
19.4. Свойство flex-wrap

Свойство `flex-wrap: wrap` определяет, как переносятся элементы, когда в родительском контейнере заканчивается свободное место:



19.5. Свойство flex-flow

Свойство `flex-flow` — эта сокращенная форма записи позволяет в одном объявлении указать значения свойств `flex-direction` и `flex-wrap`:

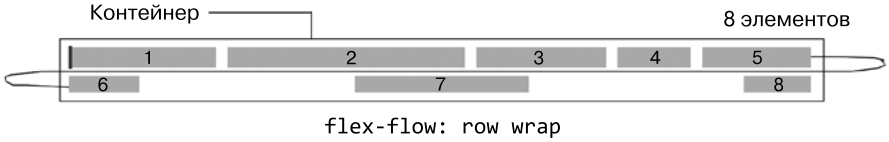


`flex-flow: flex-direction<значение> flex-wrap<значение>`

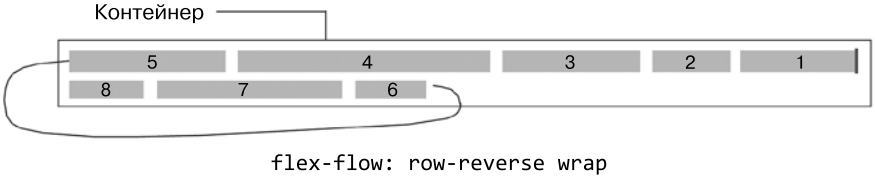
Свойство `flex-flow: row wrap` определяет параметр `flex-direction`, значение которого row, и `flex-wrap`, значение которого `wrap`:

`flex-flow: row wrap`

`flex-flow: row wrap-reverse`

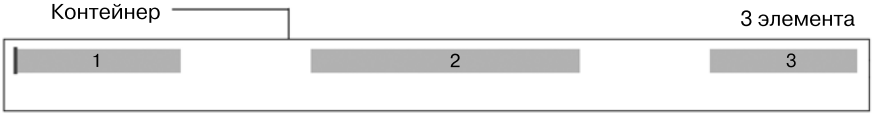


`flex-flow: row wrap`
`justify-content: space-between`

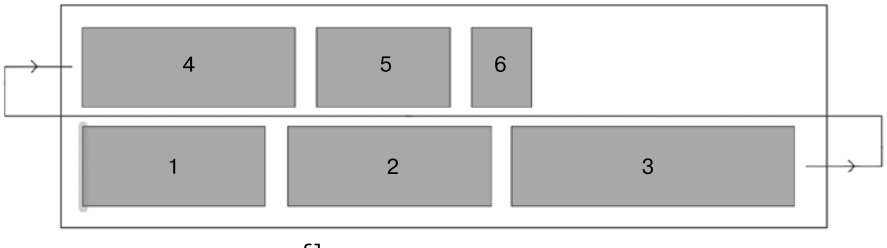


`flex-flow: row-reverse wrap`

flex-flow: row-reverse wrap-reverse

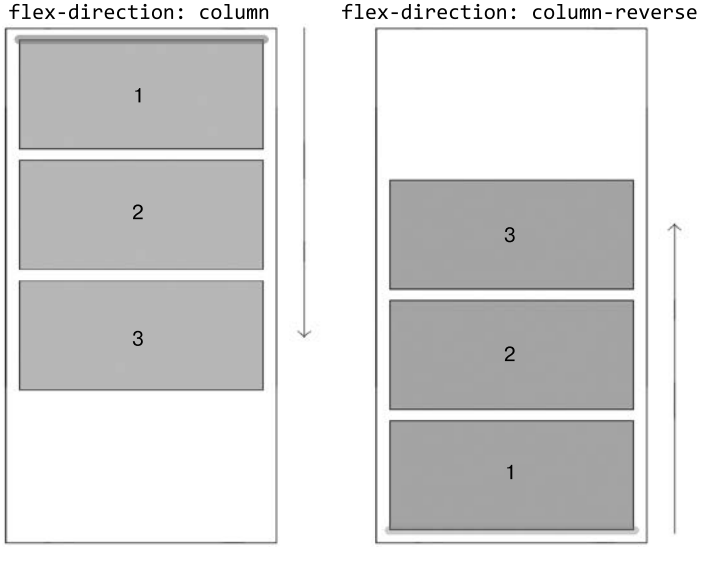


flex-flow: row wrap; justify-content: space-between



flex-wrap: wrap-reverse

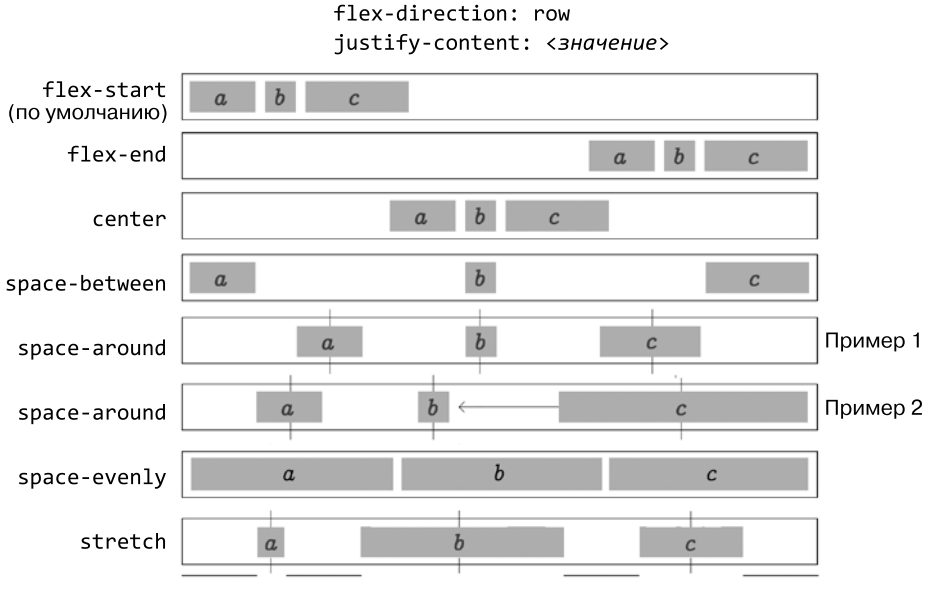
Направление можно изменить, сделав первичной поперечную ось.



Когда мы меняем значение `flex-direction` на `column`, свойство `flex-flow` ведет себя точно так же, как продемонстрировано в предыдущих примерах.

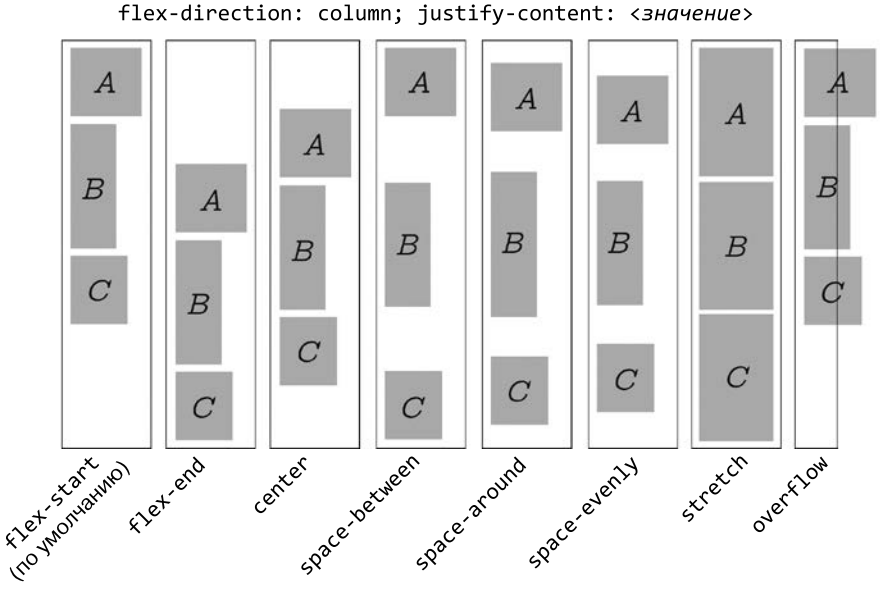
19.6. Свойство justify-content

В следующем примере мы используем только три элемента в строке. Количество элементов не ограничено.

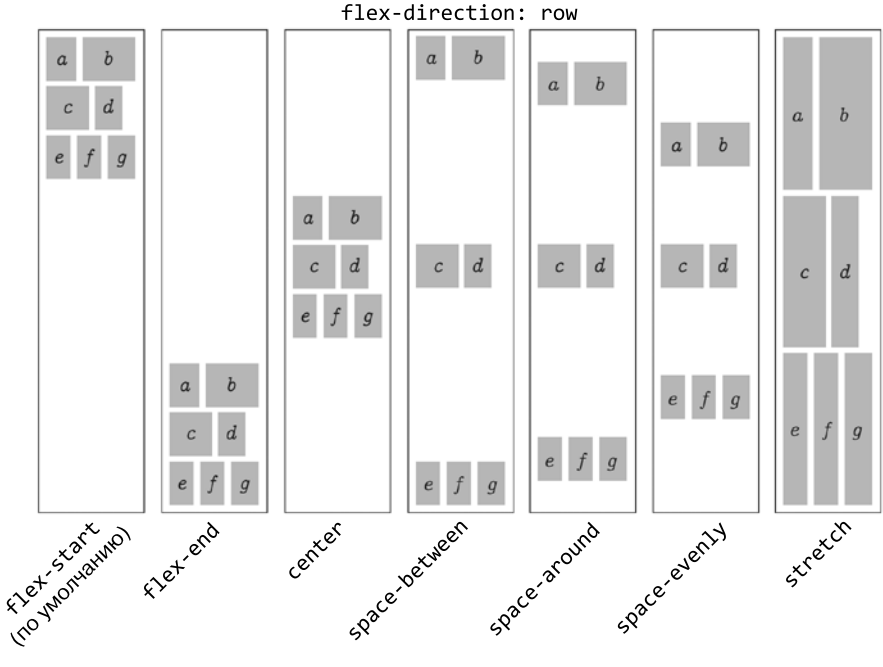


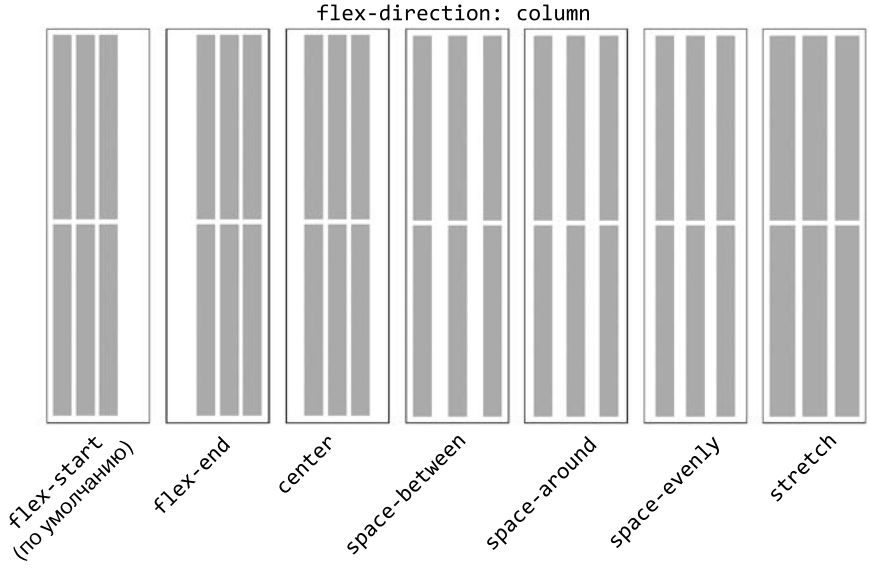
Эти схематичные рисунки демонстрируют поведение элементов лишь в том случае, если для свойства `justify-content` применяется одно из перечисленных значений.

То же свойство `justify-content` используется для выравнивания элементов, когда `flex-direction` присвоено значение `column`:



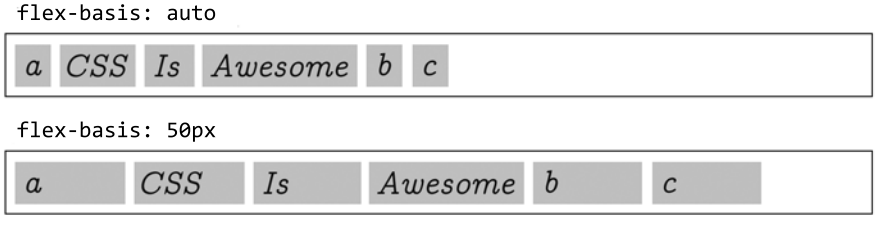
Исходя из документации CSS, это так называемая упаковка flex-строк:





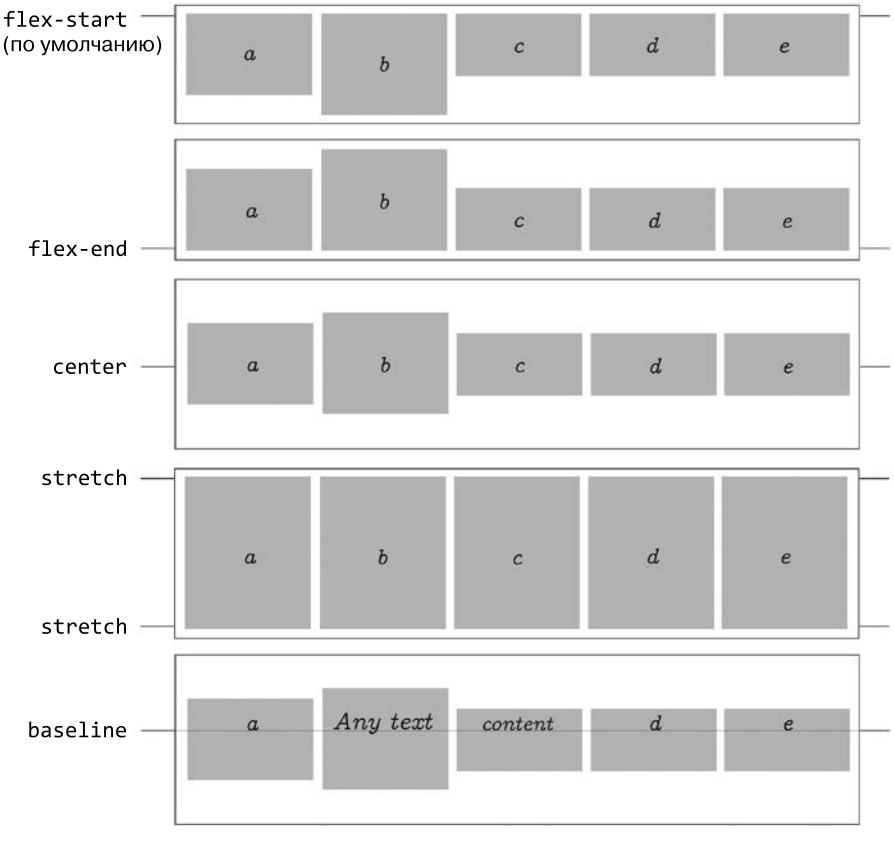
19.7. Свойство flex-basis

Свойство `flex-base` работает аналогично другому свойству CSS: `min-width` — и увеличит размер элемента в зависимости от размера внутреннего содержимого. В противном случае по умолчанию будет использоваться базовое значение.



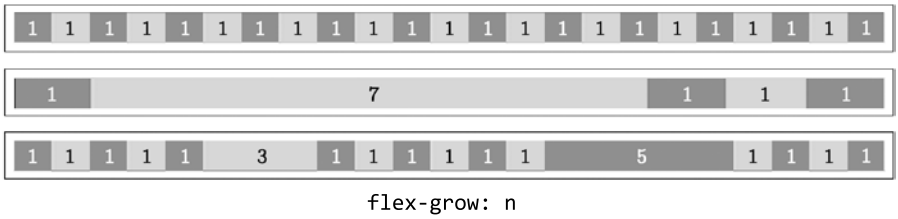
19.8. Свойство align-items

Свойство `align-items` управляет выравниванием элементов по горизонтали относительно родительского контейнера:



19.9. Свойство flex-grow

Свойство `flex-grow` масштабирует элемент относительно суммы размера всех других элементов в той же строке, которые автоматически корректируются в соответствии с указанным значением.

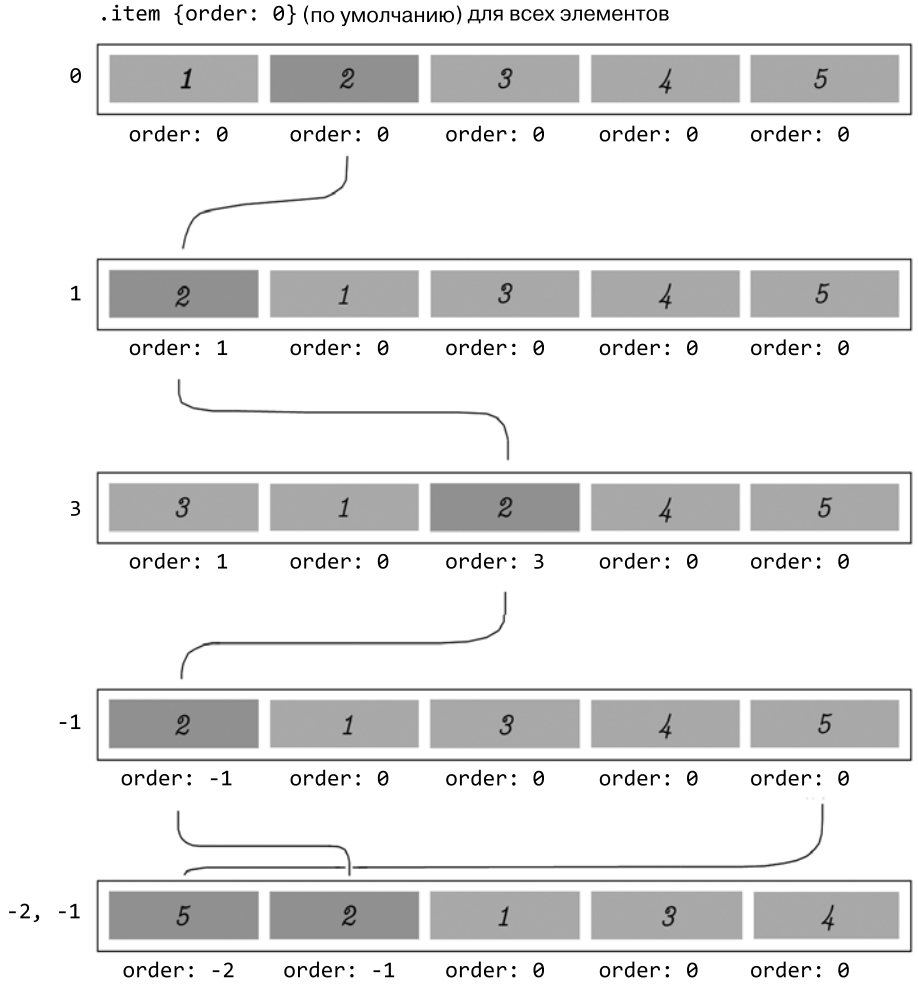


`flex-grow: n`

В каждом примере здесь значение элемента `flex-grow` было установлено как `1`, `7`, а в последнем из них — `3` и `5`.

19.10. Свойство order

С помощью свойства `order` можно изменить порядок элементов:



19.11. Свойство flex-shrink

Свойство `flex-shrink` является противоположностью `flex-grow`. В данном примере значение 7 использовалось, чтобы «сжать» выбранный элемент на промежуток времени, равный 1/7 размера окружающих его элементов, что также будет изменено автоматически:

`flex-shrink: 7` (сжатие в семь раз больше, чем у остальных элементов)

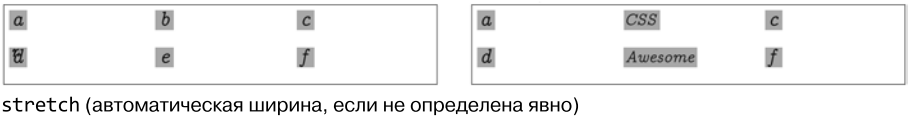
Работая с отдельными элементами, можно применить свойство `flex` в качестве сокращения для свойств `flex-grow`, `flex-shrink` и `flex-based`, используя только одно имя свойства:

```
.item { flex-none | [flex-grow | flex-shrink | flex-basis]}
```

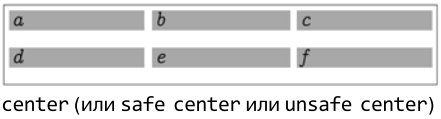
19.12. Свойство justify-items

Свойство `justify-items` аналогично `justify-content` во flex-верстке, только применяется в grid-верстке:

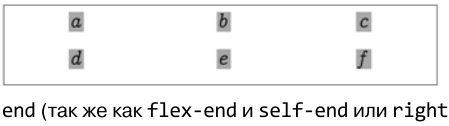
`normal` | `auto` (по умолчанию), так же как `start` или `flex-start` или `self-start` или `left`



`stretch` (автоматическая ширина, если не определена явно)

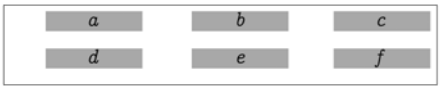


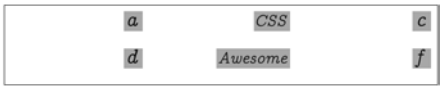
`center` (или `safe center` или `unsafe center`)



`center` (расширяется в зависимости от контента)

`end` (так же как `flex-end` и `self-end` или `right`)





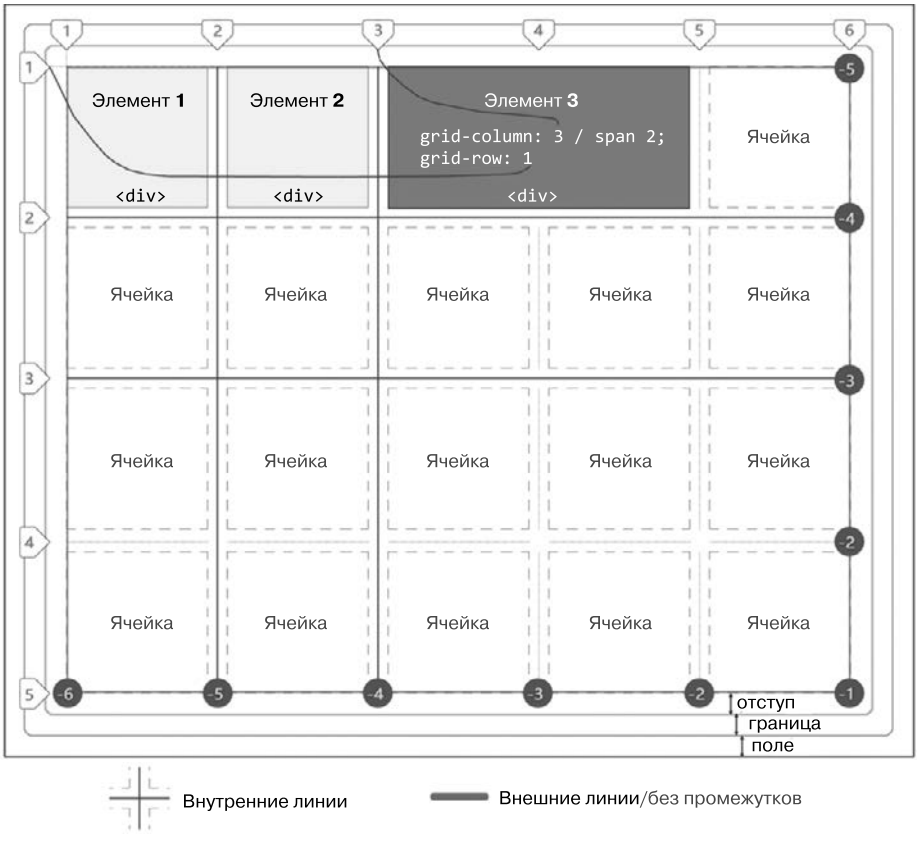
19.13. Интерактивный flex-редактор

Как следует из названия, flex-редактор предоставляет элементы разметки, которые *динамически* реагируют на размеры родительского контейнера. Но это описание не отражает всего объема возможностей редактора. Данный инструмент используется для обучения flex-верстке. Информацию, касающуюся интерактивного редактора, можно найти по ссылке www.csstutorial.org/flex-both.html.

Кроме того, этот редактор применяют разработчики, которые не хотят перенабирать избыточный HTML-код просто для создания новой компоновки, — редактор поможет создать макет и копировать HTML-код одним нажатием кнопки!

20 Grid-верстка: блочная модель

Если бы существовала *блочная модель CSS-сетки*, то она, вероятно, выглядела бы так.



Сетка представляет собой *двумерную систему компоновки*, основанную на разбивке пространства на ячейки. Разработчик должен предоставить достаточно внутренних элементов/ячеек, чтобы определить конкретную область сайта или веб-приложения.

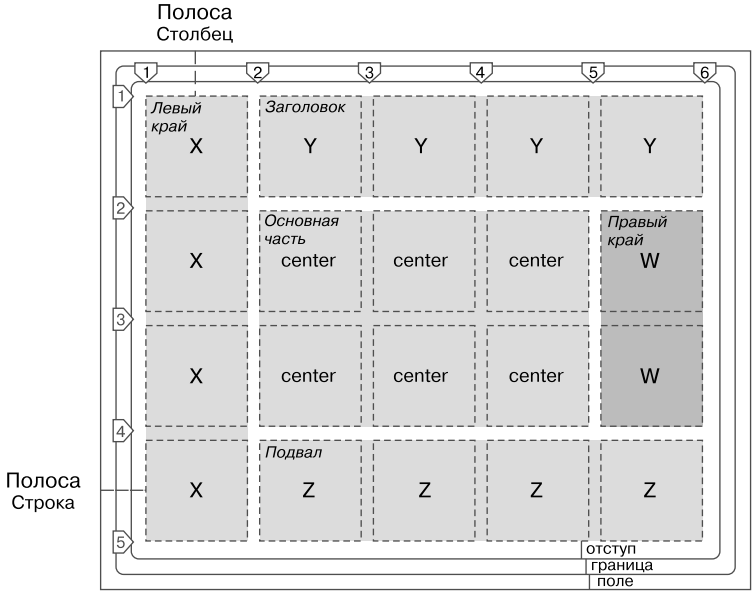
Более темные круги указывают на отрицательную систему координат, которая начинается с `[-1, -1]` в правом нижнем углу.

21 Grid-верстка: шаблоны grid-областей

Области уже обсуждались в предыдущей главе, но каким образом они могут послужить для создания фактического макета?

Имейте в виду: к grid-верстке нужно подходить творчески. Не существует какой-либо одной схемы или формулы.

Самый распространенный метод — это разбиение макета на области с использованием свойства `grid-template-areas` и присвоение имен областей для каждой строки в строковом формате, как показано в примере ниже.



Как видите, нельзя создавать grid-области неправильной формы. Они должны быть квадратными или прямоугольными.

Наш grid-макет может быть создан следующим образом:

```
001 div#grid {
002     display: grid;
003     grid-template-areas:
004
005         'x y y y y'
006         'x center center center w'
007         'x center center center w'
008         'x z z z z';
009 }
```

Соответствующий HTML-код выглядит так:

```
001<div id = "grid">
002     <div style = "grid-area: x">Left</div>
003     <div style = "grid-area: y">Header</div>
004     <div style = "grid-area: z">Footer</div>
005     <div style = "grid-area: w">Right</div>
006     <div style = "grid-area: center">Main</div>
007 </div>
```

Grid-области шаблона весьма уместны при создании первичной внешней рамки для макета. Часто внутренним ячейкам веб-разработчики назначают свойство `display: flex`.

21.1. Grid-верстка и медиазапросы

Медиазапросы

Медиазапросы похожи на оператор `if`. Они начинаются с правила `@media`, а в скобках указывается условие.

Изменение содержимого в зависимости от измерений браузера

Один из наиболее распространенных вариантов использования медиазапросов — выполнение чего-либо на основе текущих измерений браузера. Это значит, что запросы часто применяются в качестве первого варианта выбора для *создания адаптивных макетов*.

Будучи использованными таким образом, *медиазапросы* имитируют событие `onresize` в JavaScript, за исключением того, что вам не нужно самостоятельно создавать какие-либо обратные вызовы событий.

Во время моего собеседования в техасской компании по разработке программного обеспечения в 2017 году руководитель группы озвучил идею, что ученые-компьютерщики (и, возможно, ученые в целом) добиваются прогресса, «заполняя промежутки».

Эта идея запомнилась мне надолго.

Я не знаю, но, может быть, вы, как и я, пытались «заполнить промежутки» в изучении CSS-сетки сразу же после появления новой технологии в JavaScript. Эта идея стала основой для разработки визуального словаря — книги, которую вы сейчас читаете. Именно тогда я решил взять все существующие 415 свойств CSS и визуализировать их, создавая схематичные рисунки.

Хотя в первую очередь я считаю себя программистом JavaScript, мне бы хотелось думать, что я также немного занимаюсь и разработкой графического дизайна. Возможно, я взял идею промежутков из контекста, когда применил ее к grid-разметке в этом уроке.

Немного терпения!

Как уже могут знать профессиональные дизайнеры книг, идея grid-верстки заключается в том, чтобы понять не видимые части дизайна макета, а те, которые не являются таковыми.

Позвольте объяснить!

Верстальщики книг заботятся о полях — по сути, невидимом элементе книжного дизайна. Он может показаться не таким уж важным, но стоит убрать поля (то, о чем читатель осведомлен менее всего) — и читать становится неудобно.

Читатели замечают отсутствие полей, только когда их нет. Поэтому дизайнер (чего угодно) обязательно должен уделять внимание невидимым элементам оформления.

Мы всегда можем просто вставить контент в макет. Возможно ли, что... когда мы проектируем с помощью grid-верстки, все, что мы делаем для создания красивых макетов, — это работаем с улучшенной версией книжных полей? Вполне вероятно.

Промежутки

Конечно, grid-верстка выходит далеко за рамки проектирования полей в книгах, но принцип так называемого невидимого дизайна остается неизменным. Невидимые нами элементы очень важны. В grid-верстке эта концепция рассматривается как промежутки.

В конце концов, grid-верстка — то же самое, что перенести поля книги на следующий уровень.

21.2. Верстка сайта на основе grid-элементов

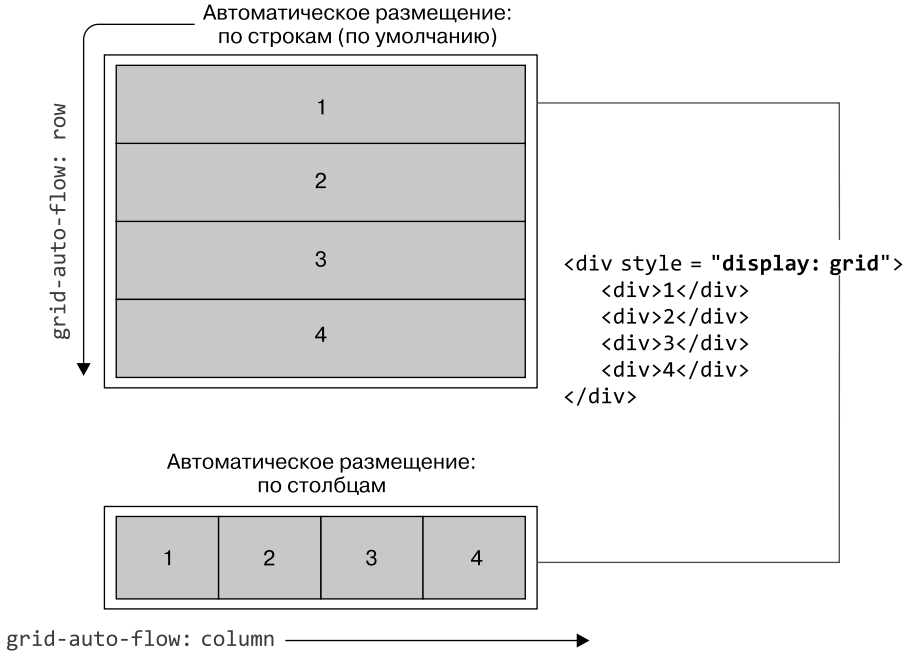
Подобно flex-верстке, свойства при grid-верстке никогда не применяются только к одному элементу. Grid-макет работает как единое целое, состоящее из родителя и содержащихся в нем элементов.

Сначала... нам понадобится контейнер и несколько элементов.

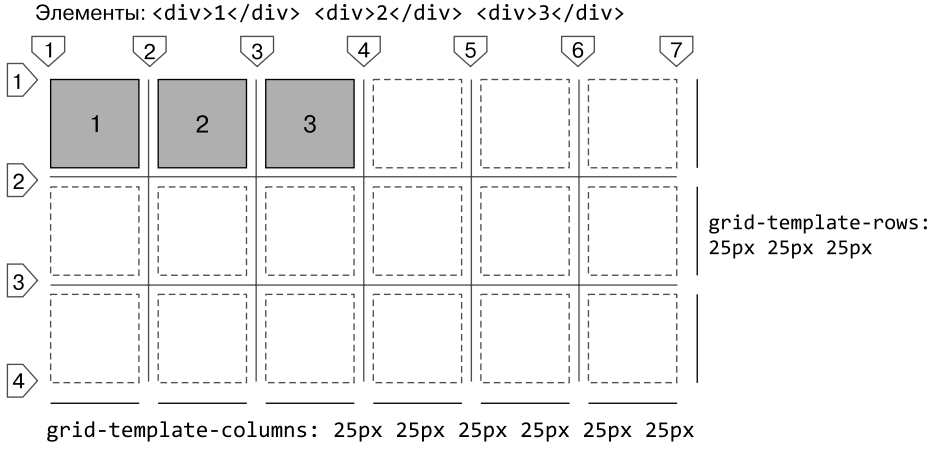
Поток grid-макета может идти в любом направлении. Но по умолчанию используется значение `row`.

Это значит, что если все остальные значения по умолчанию не будут затронуты, то ваши элементы автоматически сформируют одну

строку, где каждый из них наследует свою ширину от элемента grid-контейнера.



Как и flex-макет, grid-разметка может выравнивать элементы в одном из двух направлений: по строкам или столбцам, заданных свойством `grid-auto-flow`.



В процессе grid-верстки создается виртуальная среда, в которой элементам не нужно заполнять всю grid-область. Но чем больше элементов вы добавите, тем больше из них будет доступно для ее заполнения. Grid-верстка делает этот автоматический процесс более изящным.

В процессе grid-верстки шаблоны столбцов и строк служат для выбора того, сколько элементов будет задействовано в вашей сетке вниз и поперек. Вы можете указать их количество, используя свойства `grid-template-lines` и `grid-template-columns` соответственно, как показано на рисунке выше. Это основная конструкция grid-макета.

Момент, который вы сразу заметите в процессе grid-верстки, — определение промежутков. Это свойство отличается от любого другого ранее описанного свойства CSS. Промежутки определяются численно, начиная с верхнего левого угла элемента.

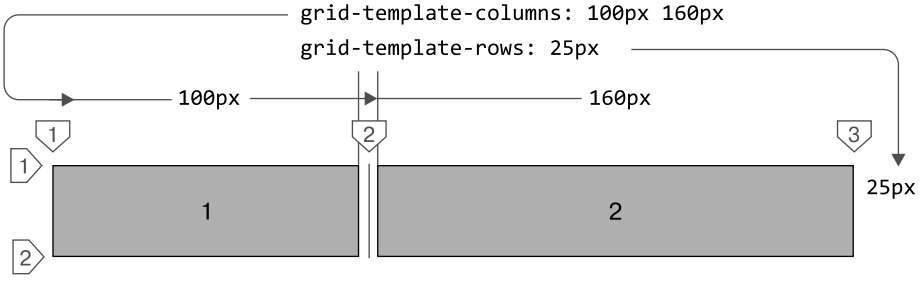
Так, `columns + 1` — промежуток между столбцами, а `rows + 1` — между строками.

Grid-макет не имеет *отступов, границ* или *полей* по умолчанию, и все значения элементов по умолчанию устанавливаются в `content-box`. То есть содержимое дополняется внутри элемента, а не снаружи, как во всех других общих блокирующих элементах.

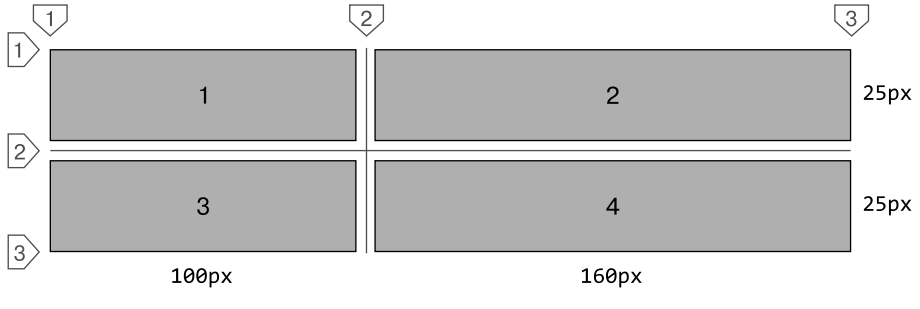
Это одно из лучших свойств grid-верстки. Наконец у нас есть новый инструмент макета, который по умолчанию обрабатывает его блочную модель как `content-box`.

Размер промежутка в grid-макете может быть установлен индивидуально для строки или столбца, если вы используете свойства `grid-row-gap` и `grid-column-gap`. Или ради удобства... можно объединить оба свойства в одно — `grid-gap`.

В следующем примере я создал grid-макет, состоящий из одной строки и двух столбцов. Обратите внимание: клинья здесь определяют горизонтальные и вертикальные промежутки между элементами. На всех будущих схемах отныне эти клинья будут использоваться для показа промежутков. Последние немного отличаются от границ или полей тем, что не заполняются за пределами grid-области.



Чтобы начать знакомство с grid-версткой, рассмотрим первый простой пример (см. с. 147). Здесь у нас есть свойства `grid-template-columns` и `grid-template-rows`, определяющие базовый grid-макет. Они могут принимать несколько значений (разделенные пробелами), которые, в свою очередь, определяют столбцы и строки. В нашем примере мы использовали такие grid-свойства для верстки макета, состоящего из двух столбцов (`100px 160px`) и одной строки (`25px`). Кроме того, промежутки на внешней границе grid-контейнера не добавляют дополнительный отступ, даже когда размер промежутка задан явно. Поэтому их следует рассматривать как определенные справа от края. С другой стороны, расстояние между столбцами и строками — единственное, на что влияет размер промежутка.



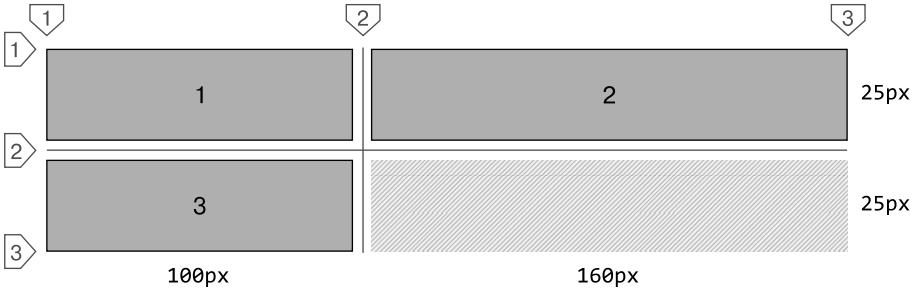
Добавление большего количества элементов в grid-макет, шаблон строки которого не обеспечивает достаточно места для их размещения, автоматически расширит grid-макет. Здесь элементы 3 и 4 были добавлены к предыдущему примеру. Но свойства `grid-template-columns` и `grid-template-row` предоставляют шаблон только для двух элементов максимум.

21.3. Неявные строки и столбцы

В процессе grid-верстки в созданные неявные заполнители автоматически добавляются неявные строки и столбцы, даже если они не были указаны как часть grid-шаблона.

Неявные (я также люблю называть их *автоматическими*) заполнители наследуют их ширину и высоту от существующего шаблона.

Они просто расширяют grid-область при необходимости. Обычно, когда количество элементов неизвестно. Например, когда обратный вызов возвращается из обращения к базе данных, получая несколько изображений из профиля продукта.

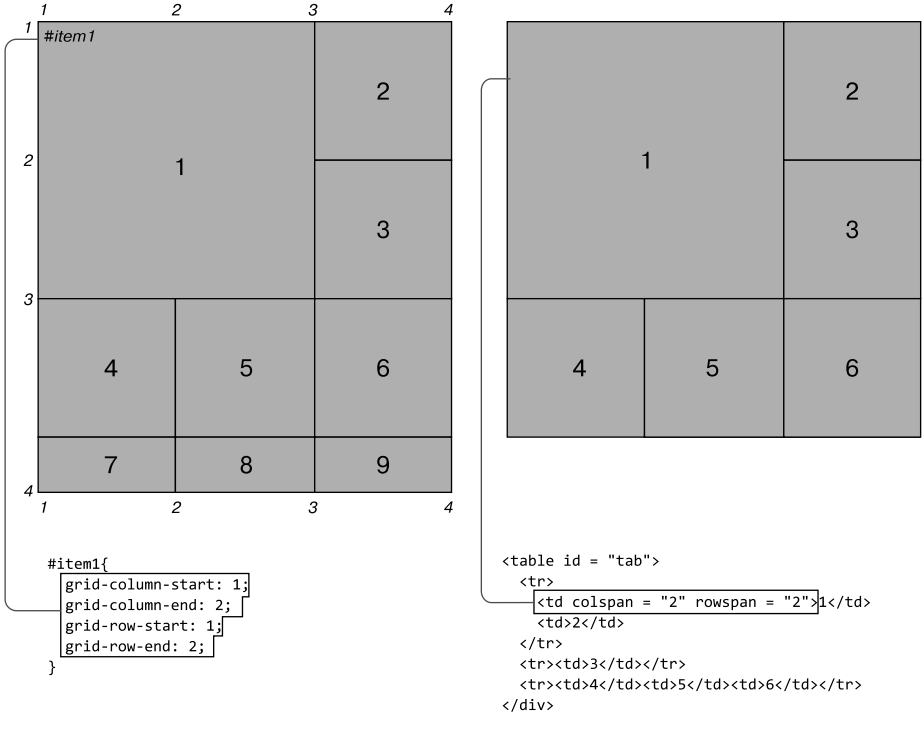


В примере у нас есть неявно добавленный заполнитель для элемента 3. Но, поскольку элемент 4 отсутствует, последний заполнитель не занят, в результате чего grid-макет сбалансирован неравномерно.

Grid-верстка никогда не должна сравниваться с табличной или использоваться подобно ей. Но отметим, что grid-макеты наследуют некоторые моменты из HTML-таблиц. В действительности, при тщательном анализе сходство выражено очень явно.

Посмотрите на следующую схему. С левой стороны вы видите grid-макет. Здесь свойства `grid-column-start`, `grid-column-end`, `grid-row-start-start` и `grid-row-end-end` идентичны табличным свойствам colspan и rowspan. Разница заключается в следующем: в grid-макетах промежутки используются для определения областей пролета. Позже вы также увидите, что для этого есть краткая форма кода. Обратите внимание: здесь элементы 7, 8 и 9 были добавлены неявно, посколь-

ку промежуток, занимаемый элементом 1 в grid-макете, вытолкнул элемент 3 из исходного grid-шаблона. В таблице это было бы невозможно.

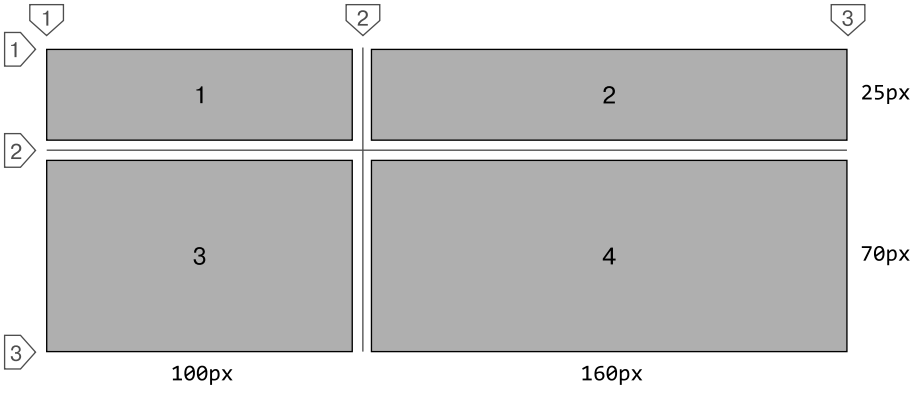


21.4. Свойство grid-auto-rows

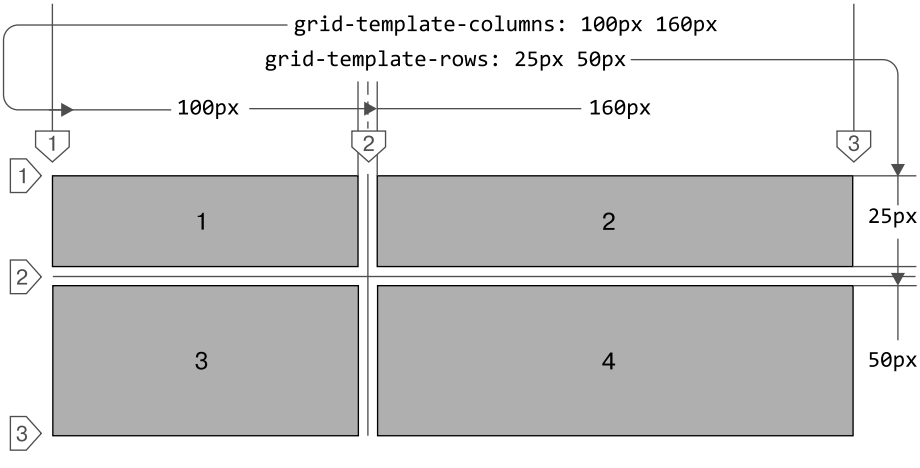
Свойство `grid-auto-row` в grid-макетах позволяет настраивать высоту автоматических (неявно созданных) строк. Да, им можно присвоить другое значение!

Вместо того чтобы наследовать значения от свойств `grid-template-rows`, можно указать конкретную высоту всех неявных строк, которые выходят за пределы ваших стандартных определений.

Неявно указанная высота строки определяется свойством `grid-auto-rows`:



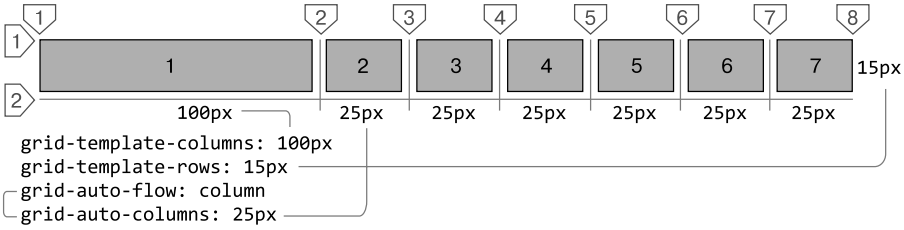
Учтите, что вы все равно можете установить все значения самостоятельно, как показано в следующем примере.



В некотором смысле grid-верстка с применением свойства `grid-auto-flow: column` похожа на *flex-верстку*.

Вы можете сделать свой grid-макет подобным flex, сменив значение по умолчанию свойства `grid-auto-flow: row` на `column`. Обратите внимание: в примере далее мы использовали свойство `grid-auto-columns: 25px` для определения ширины последовательных столбцов. Свойство `grid-auto-columns` похоже на свойство `grid-auto-row`, которое мы рас-

смотрели в одном из предыдущих примеров, за исключением того, что элементы свойства `grid-auto-row` растягиваются по горизонтали.

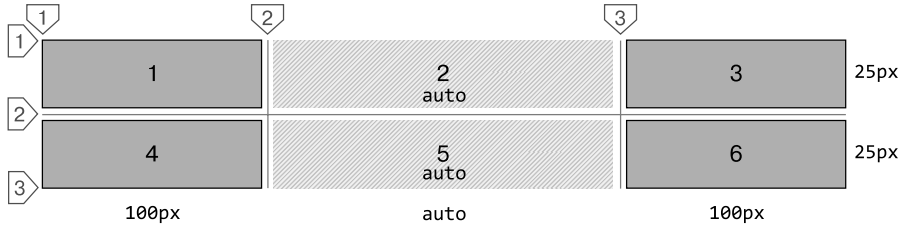


21.5. Ячейки с автоматически изменяемой шириной

Grid-верстка отлично подходит для создания традиционных макетов сайтов с двумя меньшими столбцами с каждой стороны. Существует простой способ сделать это. Просто укажите `auto` в качестве значения одной из ширин в свойстве `grid-template-column`.

`grid-template-columns: 100px auto 100px`:

Ячейки с автоматически изменяемой шириной



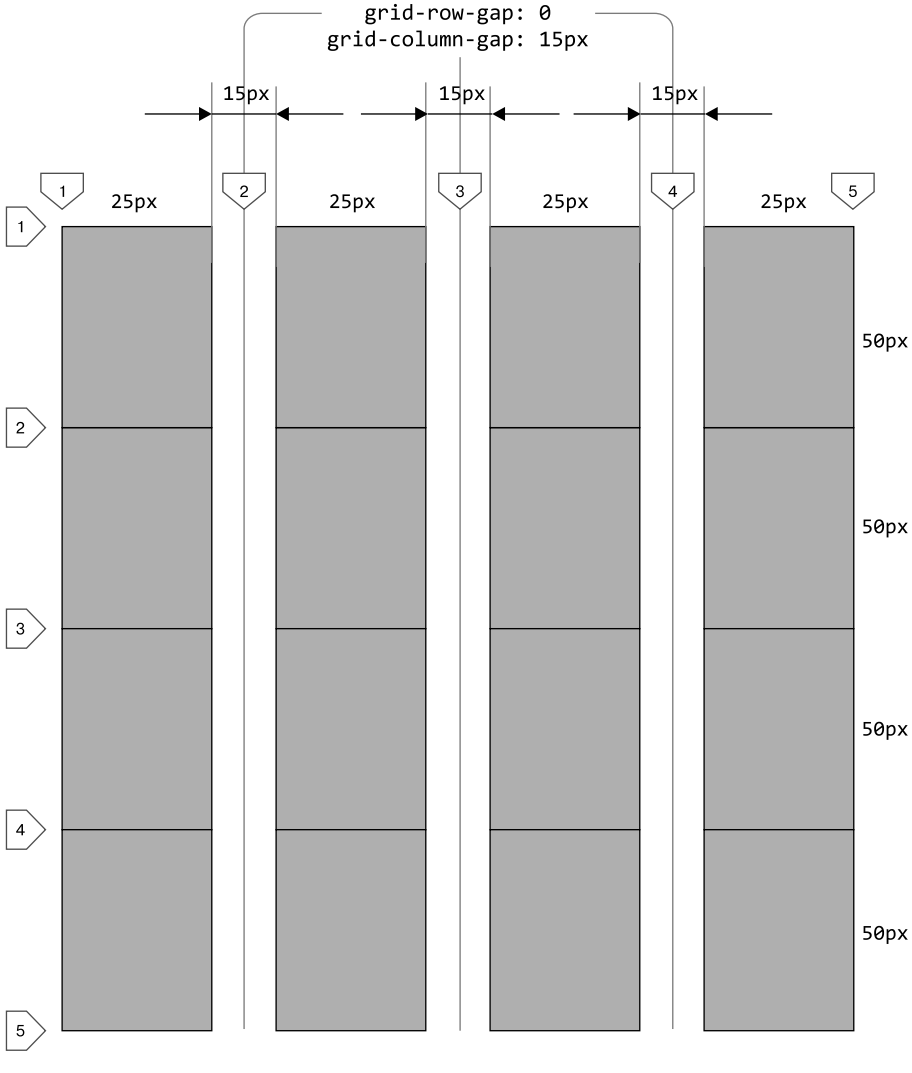
Ваша grid-область будет охватывать всю ширину контейнера или браузера.

Как вы уже поняли, grid-верстка предлагает широкий спектр свойств, которые помогут проявить креативность при создании своего сайта или макета приложения!

21.6. Промежутки

Мы уже говорили о промежутках. В основном только о том, что они *определяют пространство между столбцами и строками*. Но мы не говорили об их изменении.

Набор следующих схематичных рисунков наглядно показывает, как промежутки изменяют внешний вид grid-макета.

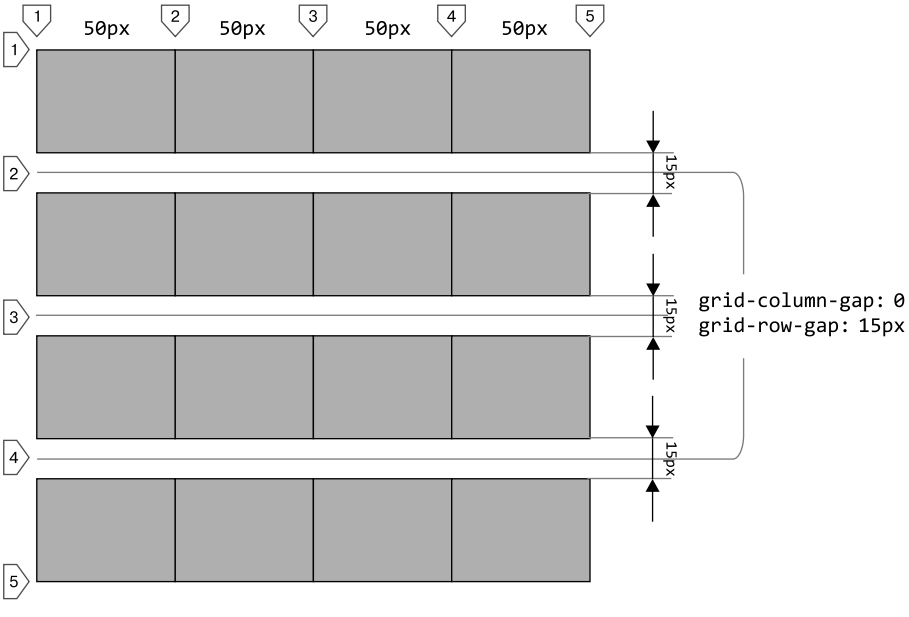


Grid-верстка допускает свойство `grid-column-gap`, которое используется для указания вертикальных промежутков одинакового размера между всеми столбцами в grid-макете.

Я намеренно оставил горизонтальные промежутки сжатыми до значения по умолчанию `0`, поскольку они не рассматриваются в этом примере.

Я уже могу представить себе дизайн в духе сайта Pinterest с несколькими столбцами, используя настройки выше.

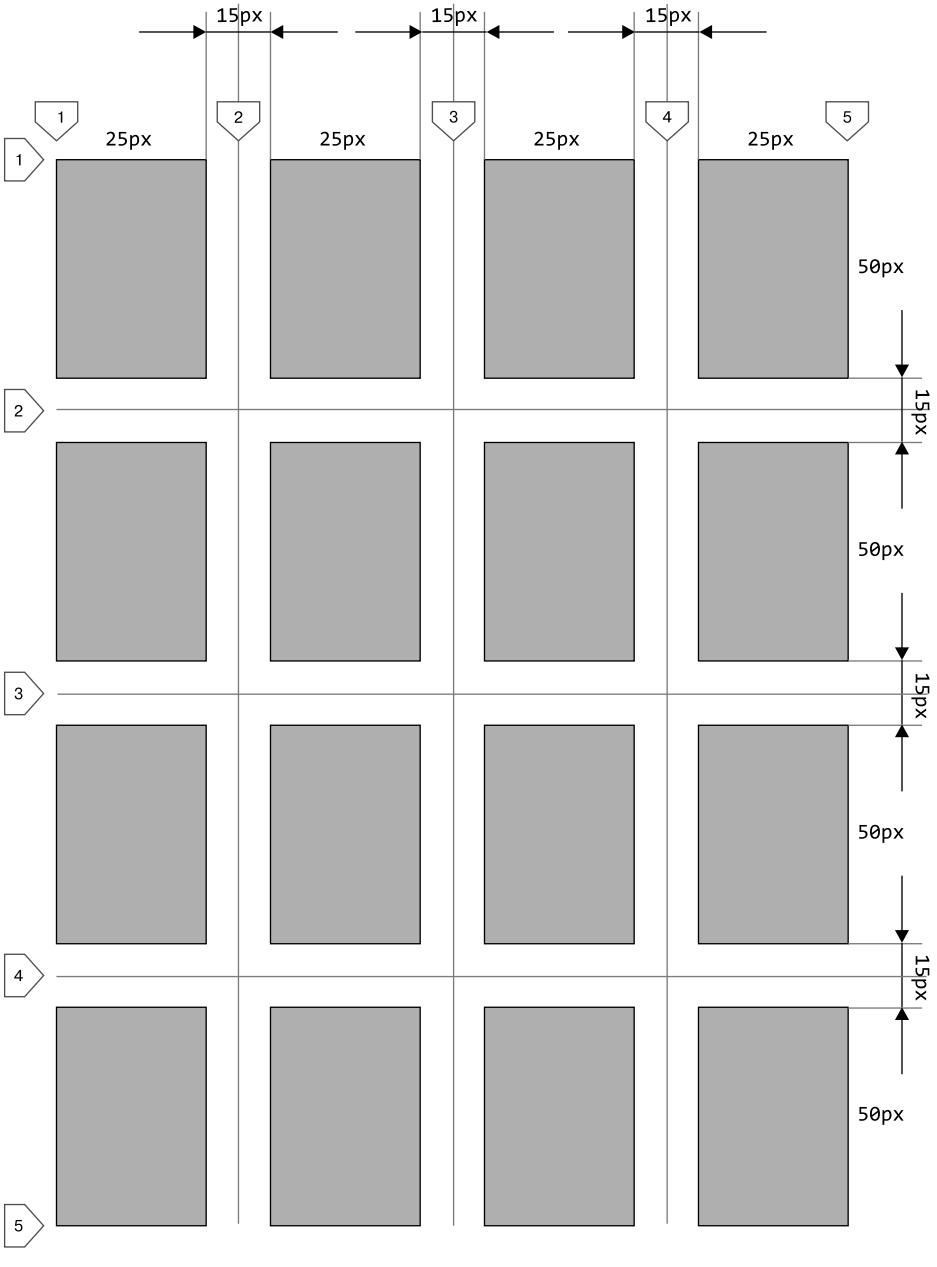
Аналогично, используя свойство `grid-row-gap`, можно установить горизонтальные промежутки для всего grid-макета:



Это то же самое, за исключением горизонтальных промежутков.

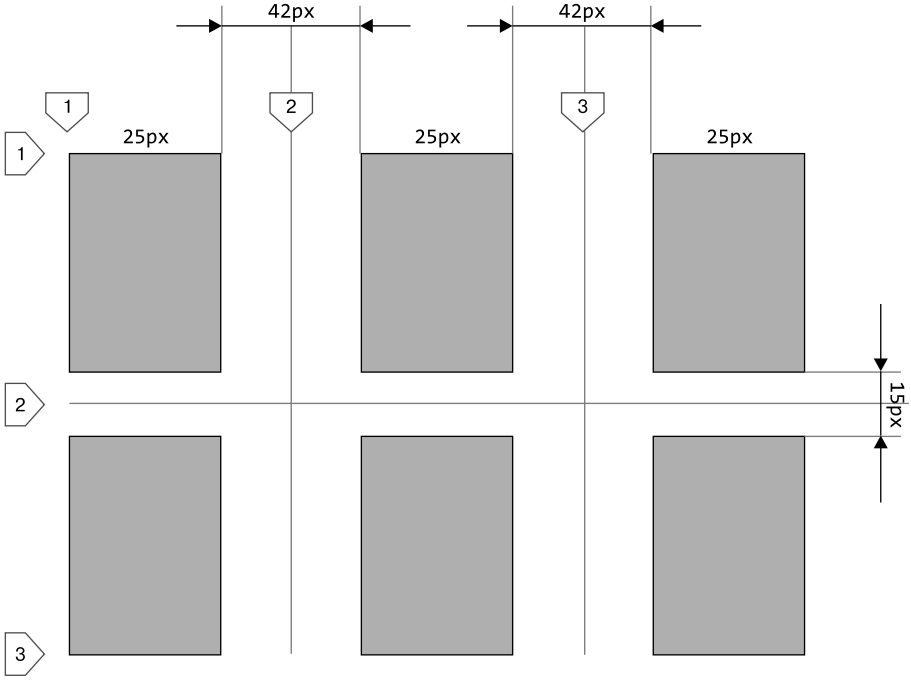
Свойство `grid-gap` позволяет установить промежутки в обоих измерениях одновременно, что показано на следующей схеме.

Можно установить промежутки во всем grid-макете, используя сокращенное свойство `grid-gap`. Однако в таком случае промежутки в обоих измерениях будут равны. В следующем примере это `15px`:


15px
15px
15px
1
2
3
4
5
25px
25px
25px
25px
1
2
3
4
5
50px
15px
50px
15px
50px
15px
50px

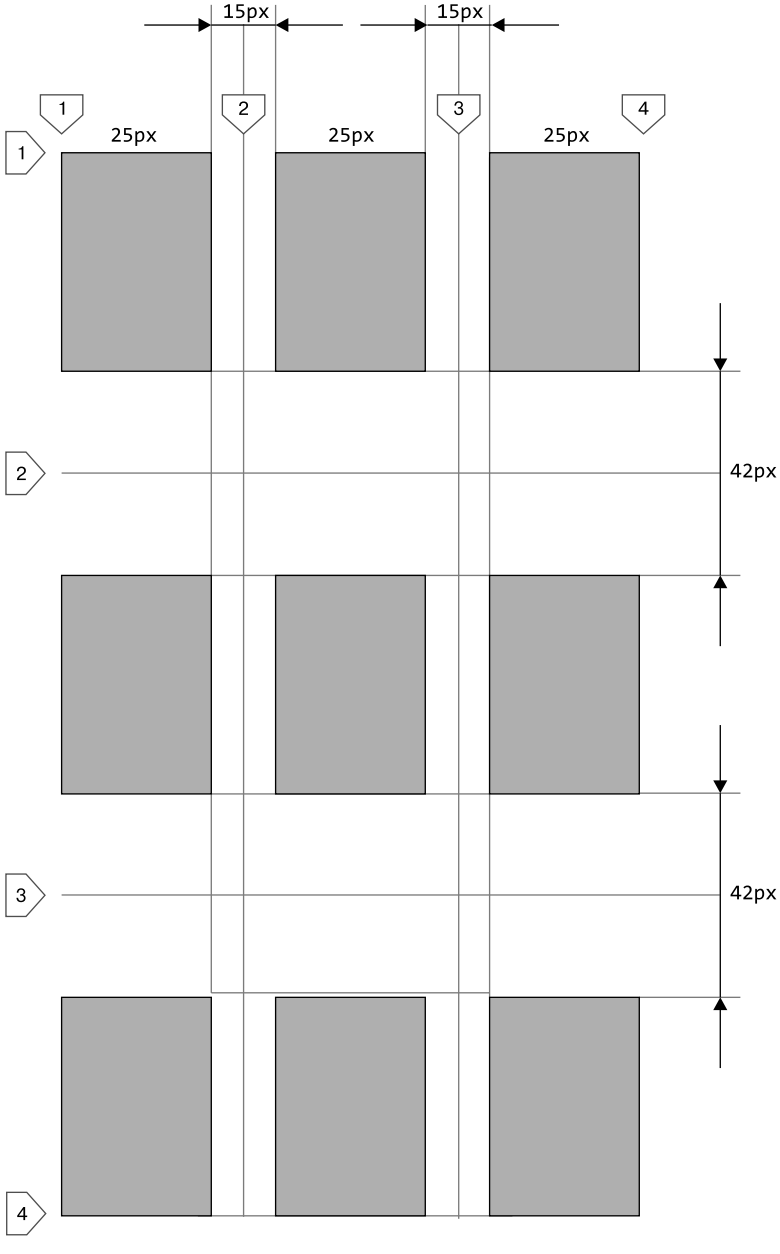
И наконец... вы можете установить промежутки индивидуально для каждого из двух измерений.

Следующие три рисунка были созданы, чтобы продемонстрировать различные варианты grid-верстки, которые могут быть полезны в различных случаях.



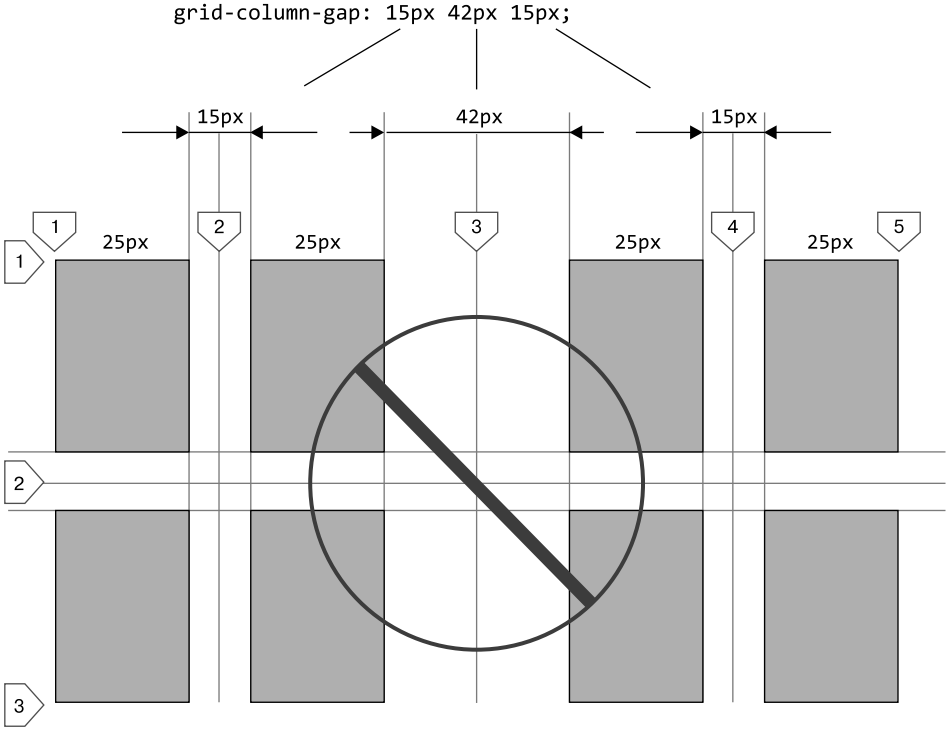
Здесь промежутки устанавливаются индивидуально для каждой строки и столбца, что позволяет варьировать конструкцию последнего. В примере используются широкие промежутки между столбцами. Вероятно, такой вариант позволит создавать галереи изображений для широкоэкранных макетов.

Пример, аналогичный предыдущему (за исключением использования более широких промежутков между строками):



Единственное, что меня разочаровало, — отсутствие поддержки способности создавать различные промежутки в одном измерении.

Думаю, это самое существенное ограничение grid-верстки. И надеюсь, что в дальнейшем оно будет исправлено.



Такой макет нельзя создать с помощью grid-верстки. Изменение размера промежутка в настоящее время (2 июня 2018 года) невозможно при grid-верстке.

21.7. Единицы fr для эффективного определения размера оставшегося пространства

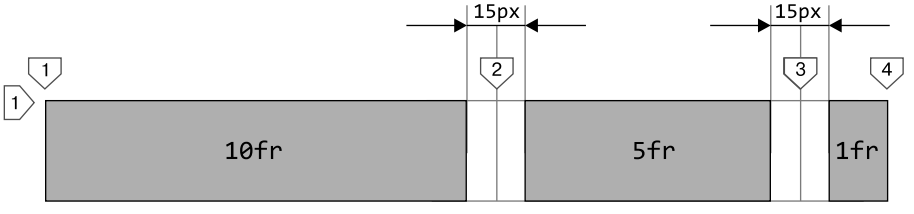
Одним из недавних дополнений к языку CSS стала единица fr. Эти единицы можно использовать не только в grid-макетах. Но в сочетании с техникой верстки они подходят для создания макетов,

адаптируемых под неизвестные разрешения экрана... и все же сохраняют пропорции в процентах.

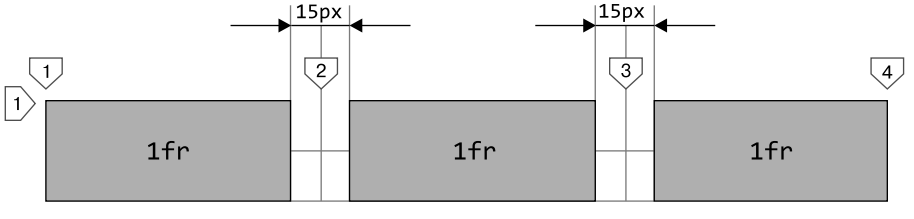
Единица fr аналогична процентным значениям в CSS (25 %, 50 %, 100 %... и т. д.), за исключением того, что представлена дробным значением (0.25, 0.5, 1.0...).

Значение `1fr` не всегда идентично 100 %. Единица fr автоматически делит оставшееся пространство. Проще всего продемонстрировать это на примере следующих схематичных рисунков.

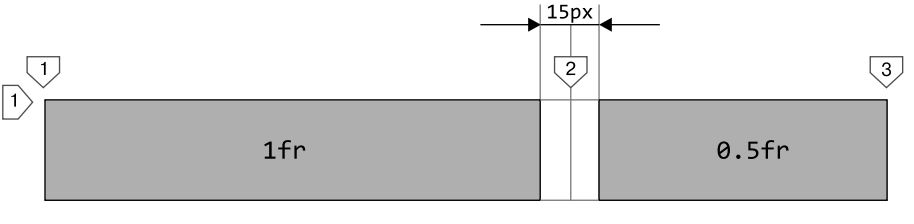
Рассмотрим базовый пример использования единиц fr.



Верстка трех столбцов шириной `1fr` позволяет создать столбцы одинаковой ширины:

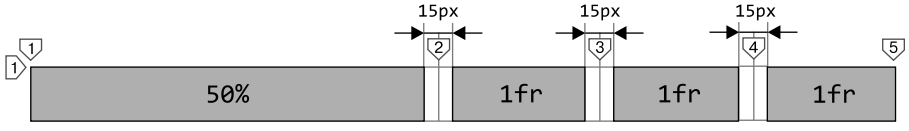


`1fr` будет 10/1 от `10fr` независимо от того, сколько места занимает `10fr`. Это все относительно:



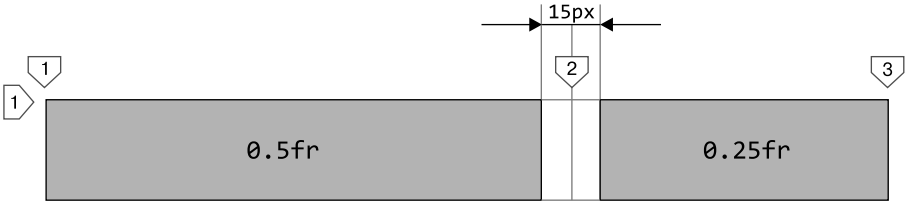
Исходя из `1fr` значение `0.5fr` — это ровно половина `1fr`. Эти значения рассчитываются относительно родительского контейнера.

Можно ли смешать процентные значения с `1fr`? Конечно!



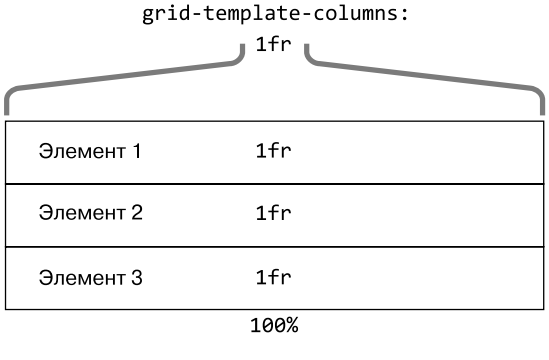
Этот пример демонстрирует смешивание процентных значений с fr. Результаты всегда интуитивно понятны и дают ожидаемый эффект.

Дробные единицы fr относительно самих себя в некоем родительском контейнере:



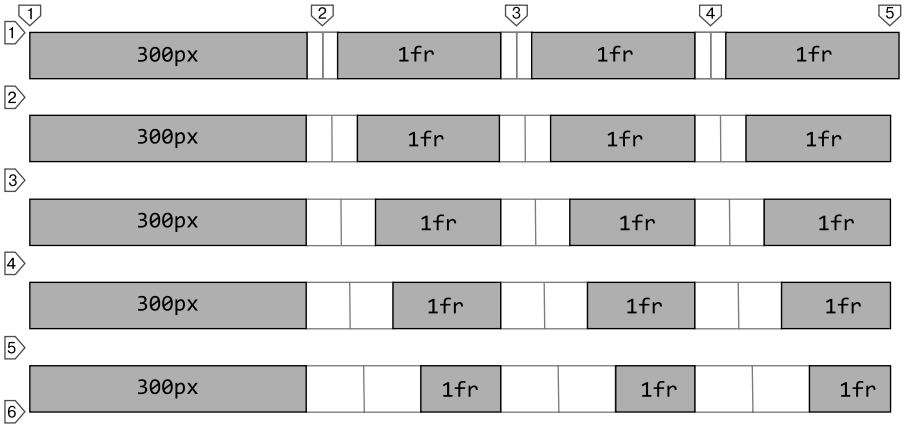
21.8. Работа с единицами fr

Дробные единицы (или *fr*-единицы) делят фрагмент макета на равные и относительные части.

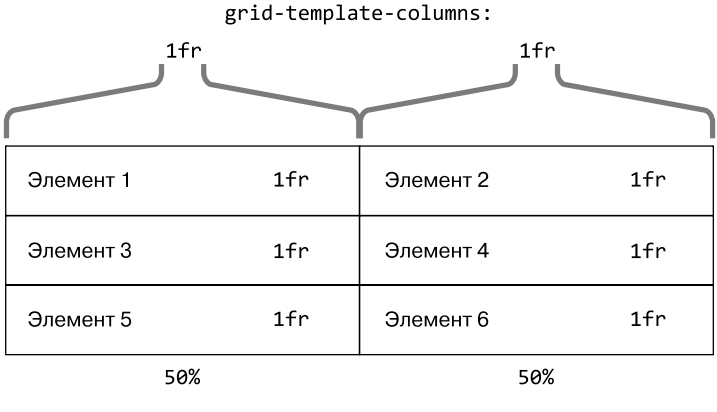


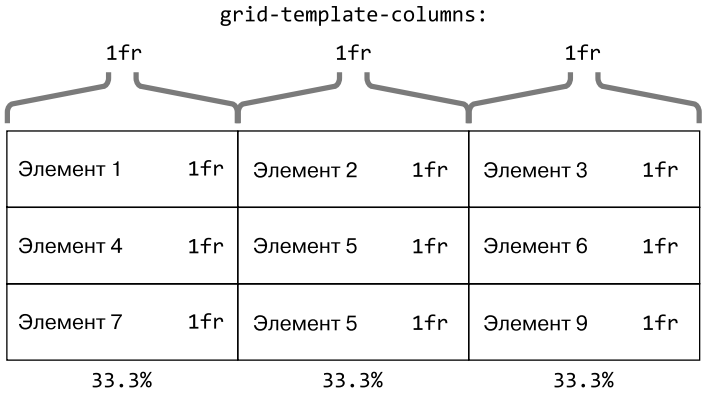
Как мы увидим ниже, `1fr` — это не конкретная единица, подобно px или em, а значение, вычисляемое *относительно* оставшегося пустого пространства.

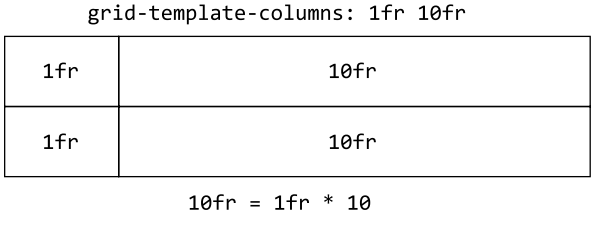
Использование единиц измерения `1fr` и одновременное увеличение промежутков между столбцами приведет к такому результату.

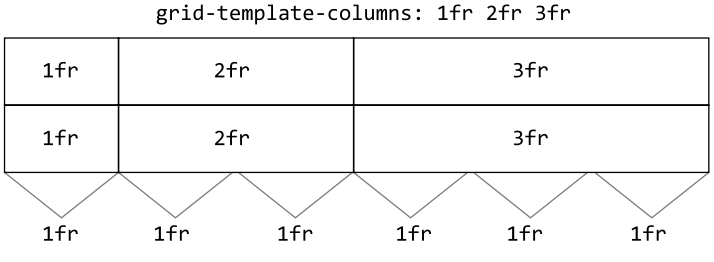


Я рассматриваю данную тему в этом разделе с целью показать: на `1fr` тоже будут влиять промежутки. Выше показаны пять различных grid-макетов, чтобы вы не забывали о промежутках при проектировании со значениями `1fr`.




grid-template-columns:
1fr
1fr
1fr
Элемент 1
1fr
Элемент 2
1fr
Элемент 3
1fr
Элемент 4
1fr
Элемент 5
1fr
Элемент 6
1fr
Элемент 7
1fr
Элемент 5
1fr
Элемент 9
1fr
33.3%
33.3%
33.3%

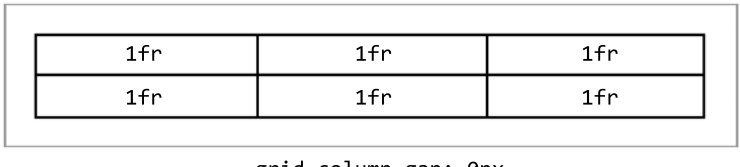

grid-template-columns: 1fr 10fr
1fr
10fr
1fr
10fr
10fr = 1fr * 10


grid-template-columns: 1fr 2fr 3fr
1fr
2fr
3fr
1fr
2fr
3fr
1fr
1fr
1fr
1fr
1fr
1fr

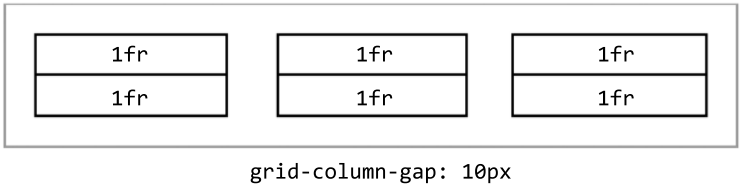
22 Единицы fr и промежутки

Промежутки влияют на значения в fr, поэтому теперь каждый раз при добавлении нового столбца нужно также вычитать размер промежутков из оставшегося пространства в элементе.

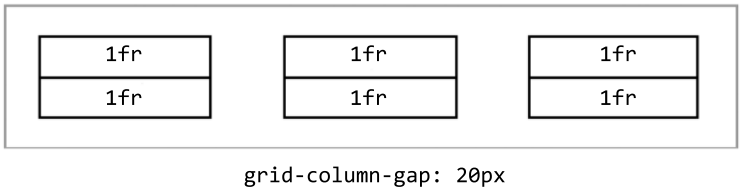
grid-template-column: 1fr 1fr 1fr



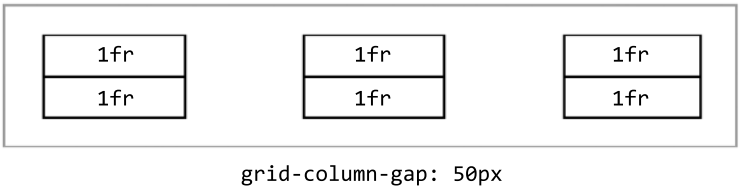
grid-column-gap: 0px



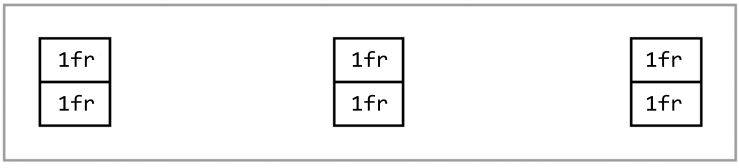
grid-column-gap: 10px



grid-column-gap: 20px

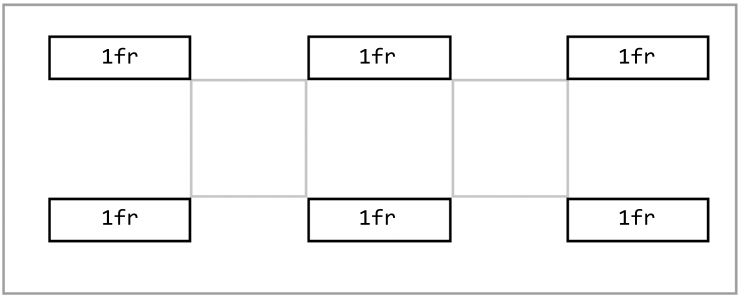


grid-column-gap: 50px

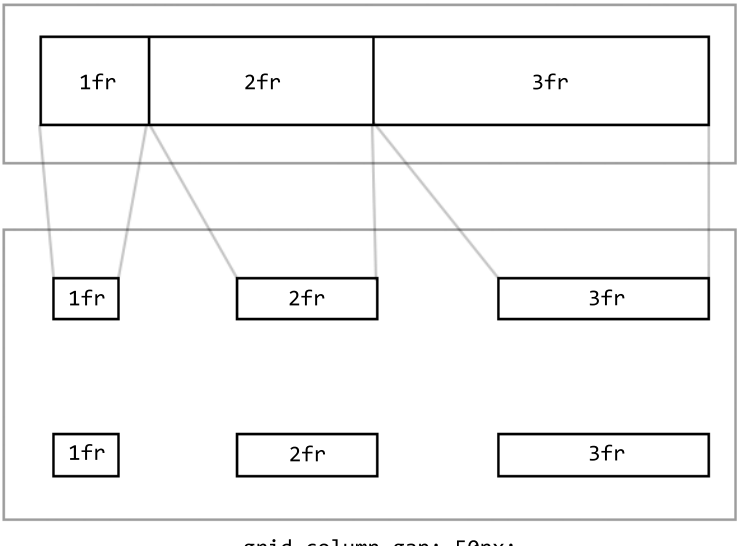


grid-column-gap: 100px

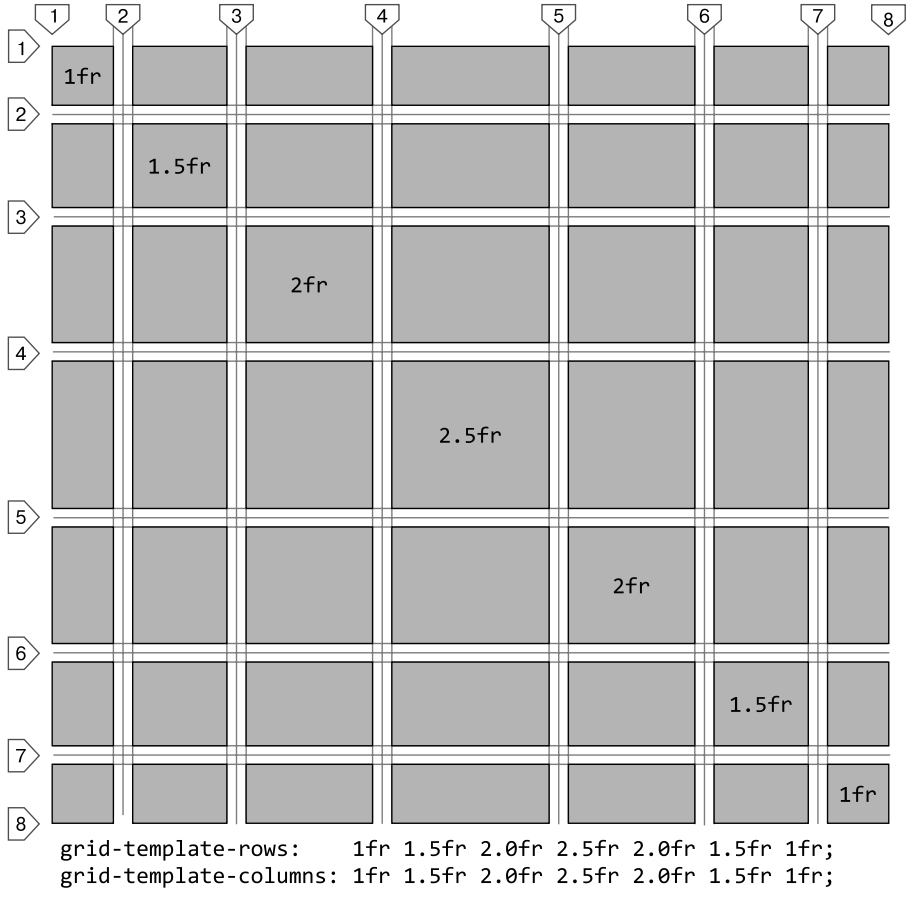
grid-template-column: 1fr 1fr 1fr




```
grid-column-gap: 50px;
grid-row-gap: 50px;
```




```
grid-column-gap: 50px;
grid-row-gap: 50px;
```



Для того чтобы быстрее разобраться в единицах fr, поэкспериментируйте с ними, создавая нечто подобное. Хотя я не знаю, где вам понадобится такая компоновка, но она ясно показывает, как единицы fr могут влиять и на строки, и на столбцы.

22.1. Повторение значений

Grid-верстка допускает использование свойства `repeat`. Оно принимает два значения: сколько раз повторять и что именно. Синтаксис:

```
repeat(раз, … что);
```

Например:


```
grid-template-columns: repeat(3, 60px, 35px)
grid-template-columns: 60px 35px 60px 35px 60px 35px
```

В данном примере мы используем свойство `grid-template-column` с повторением и без него. Обычно целесообразно выбрать более простой вариант.

Здесь свойству `grid-template-columns` присвоены два разных значения для создания одного и того же эффекта. Очевидно, что повторение — более рациональный вариант.

Итак, чтобы избежать избыточности в тех случаях, когда ваш grid-макет должен содержать повторяющиеся значения измерений, используйте свойство `repeat` в качестве средства устранения неисправностей. Свойство `repeat` можно также поместить между другими значениями.


```
grid-template-columns: 50px repeat(3, 15px 30px) 50px
```

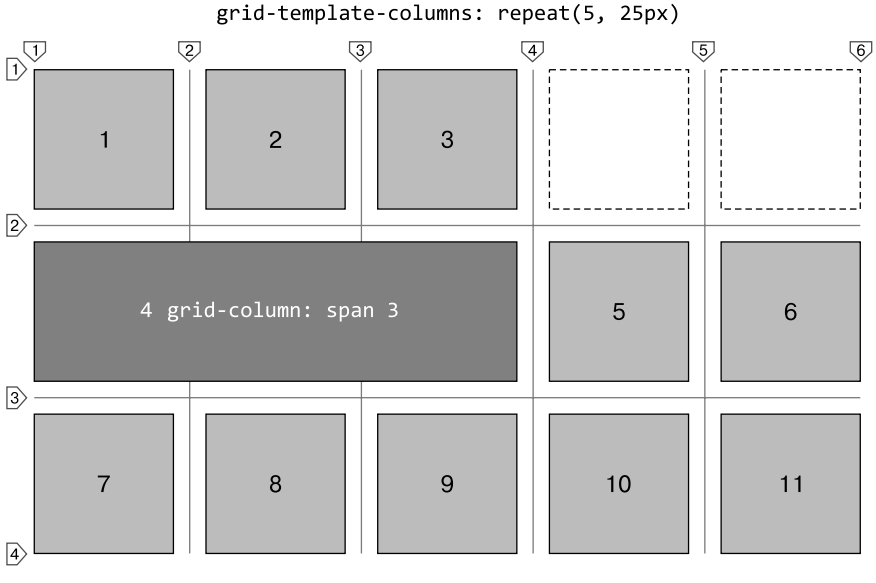
В данном примере мы повторяем раздел из двух столбцов `15px 30px` три раза подряд. Я имею в виду — в столбце. Надеюсь, вы понимаете, о чем я.

22.2. Группировка

В процессе grid-верстки вы позволяете вашим элементам растягиваться на несколько строк или столбцов. Это очень похоже на значения `rowspan` и `colspan` в табличной верстке.

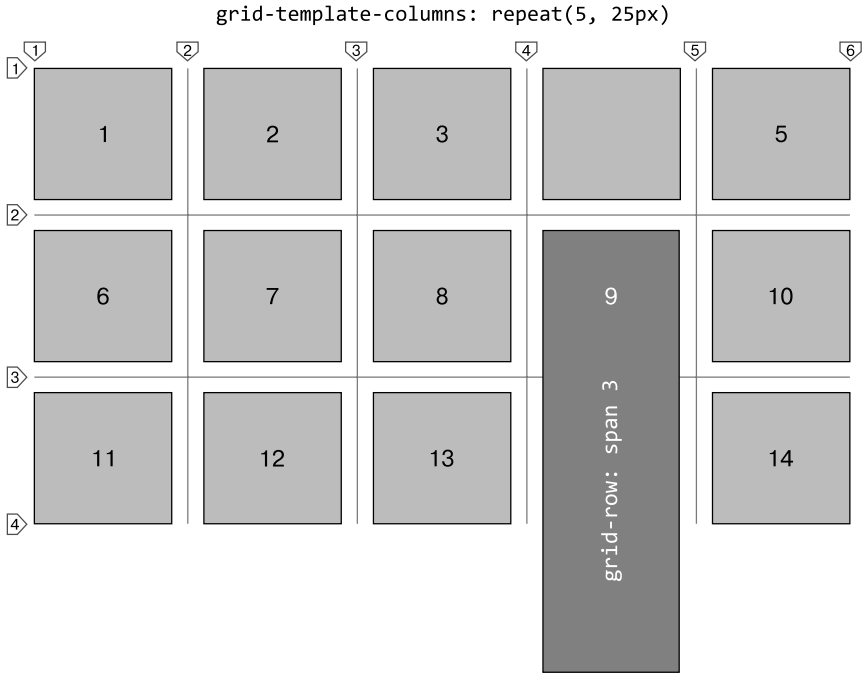
Мы создадим grid-макет, используя свойство `repeat`, чтобы избавиться от лишнего кода. Однако можно обойтись и без него — в любом случае рассмотрим пример именно для данного раздела.

Добавление свойства `grid-column: span 3` к элементу 4 привело к несколько неожиданному эффекту.

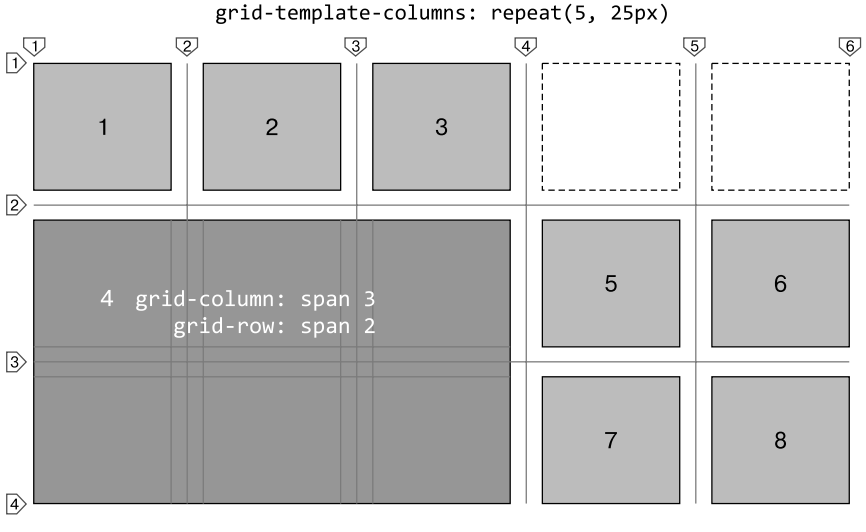


Здесь показано использование свойства `grid-column: span 3` для группировки трех столбцов. Однако grid-макет принимает решение удалить некоторые элементы, так как «составной» элемент не может вписаться в предложенную область. Обратите внимание на пустые квадраты!

Еще в grid-макетах можно группировать несколько строк. И если так получилось, что столбец теперь больше по высоте, чем высота самой grid-области, то данная область адаптируется под это условие.

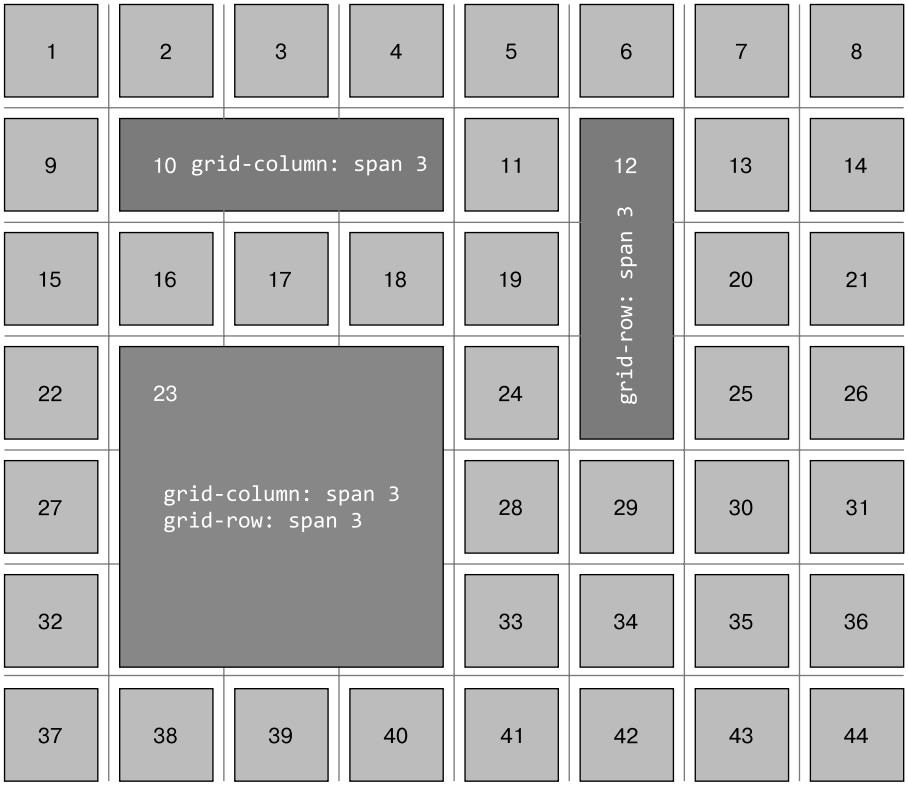


Можно сгруппировать несколько строк и столбцов одновременно. Следующий пример создан с целью быстро продемонстрировать ограничения, хотя в большинстве случаев этого, вероятно, не произойдет.

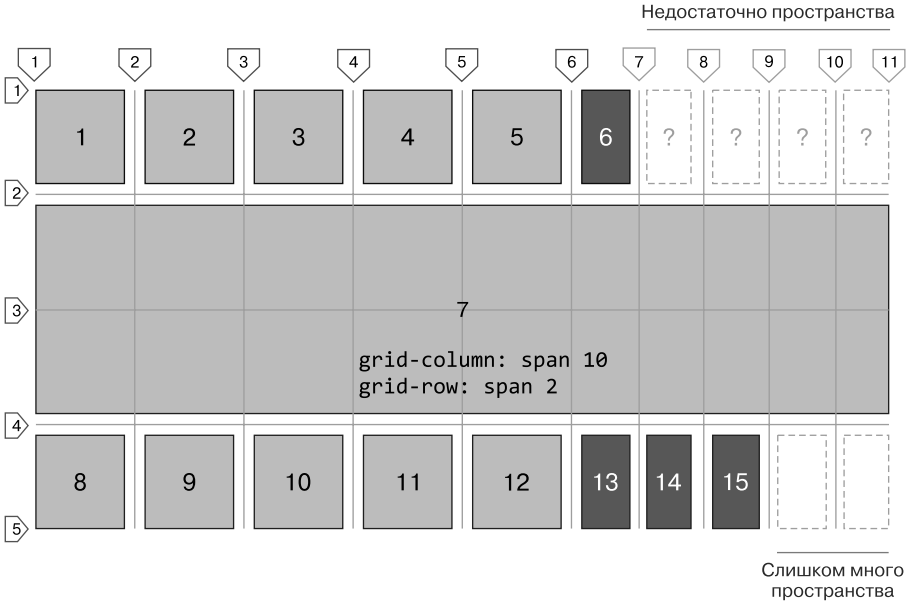


Как видите, grid-макет заполняется пустыми областями:

Обратите внимание на то, как grid-область адаптируется к элементам вокруг промежутков, которые охватывают несколько строк и столбцов. Все элементы по-прежнему остаются в grid-области, но интуитивно оборачиваются вокруг других составных элементов.



Попытавшись разбить макет с большим промежутком, я получил следующий вид, который демонстрирует ключевые ограничения grid-макетов:



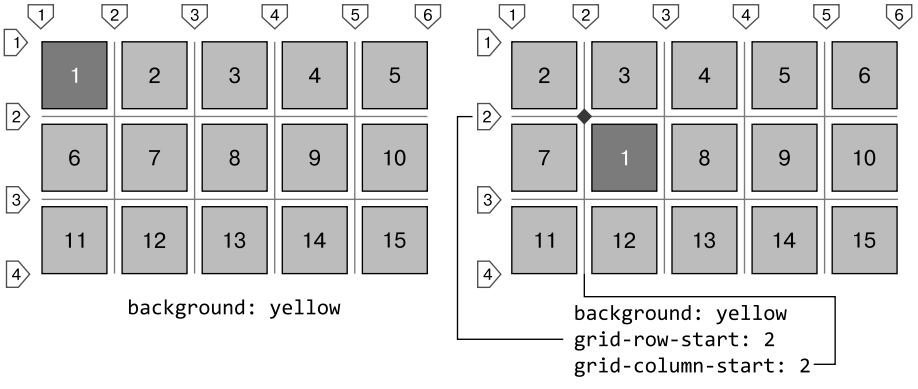
Но это все еще очень похоже на таблицу. Посмотрите мой другой вариант grid-макета, где я покажу интересное сходство.

22.3. Свойства grid-row-start и grid-row-end

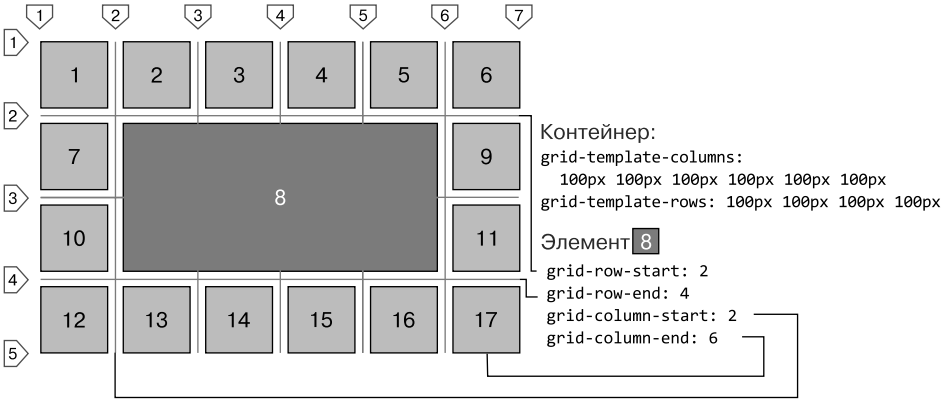
До сих пор мы использовали ключевое слово `span` в grid-макетах для создания многостолбцовых и многострочных элементов, занимающих довольно много пространства. Но grid-верстка допускает другое, более интересное решение данного вопроса.

Свойства `grid-row-start` и `grid-row-end` могут служить для определения начальной и конечной точки элемента в grid-области. Аналогично их эквивалентами для столбцов являются свойства `grid-column-start` и `grid-column-end`. Кроме того, есть два сокращенных свойства: `grid-row: 1/2` и `grid-column: 1/2`. Они работают несколько иначе, чем элементы `span`.

С помощью значений `-start` и `-end` можно физически переместить элемент в другое место в grid-макете. Рассмотрим пример.



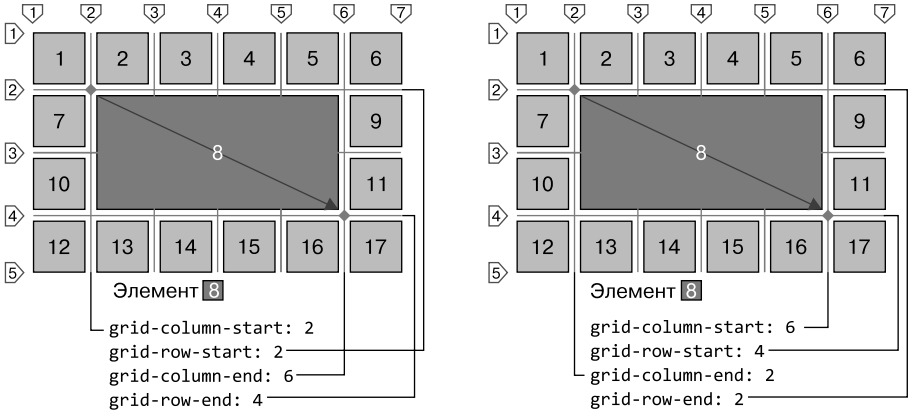
Интересный факт: разработчики grid-верстки решили, что направление группировки будет незначительным. Группировка по-прежнему создается в указанной области независимо от того, указаны ли начальная или конечная точки в обратном порядке.



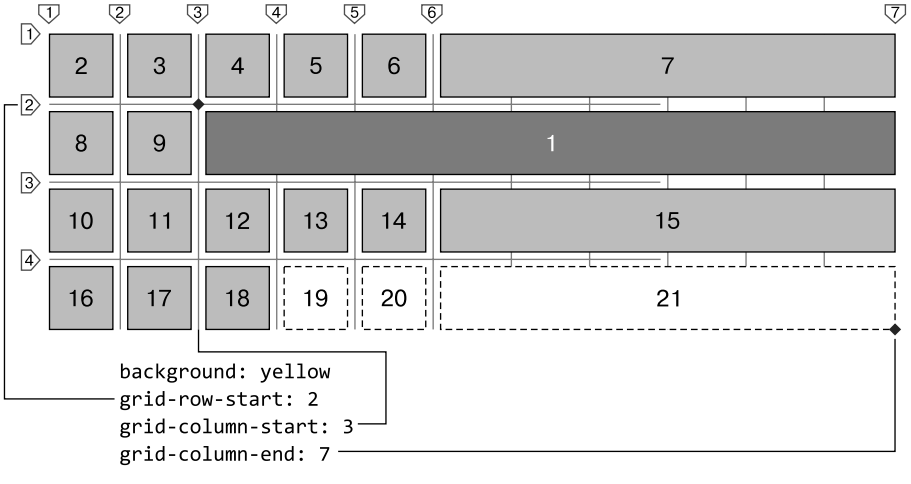
В данном примере мы взяли элемент 8 и (избыточно) указали его местоположение, используя свойства `grid-row-start` и `grid-column-start`. Но обратите внимание, что само по себе это не оказывает важного влияния на элемент 8, поскольку тот уже находится в данном месте grid-макета в любом случае. Однако, делая это, вы можете достичь функциональности, подобной элементам `span`, если также

укажете конечное местоположение с помощью свойств `grid-column-end` и `grid-column-end`.

Указание диапазона элемента независимо от направления начальной/конечной точек дает одинаковые результаты:



Рассмотрим grid-макет 6 × 4. Если вы явно укажете конечную позицию столбца элемента, которая выходит за пределы числа указанных столбцов (≥ 7), то столкнетесь с таким эффектом.

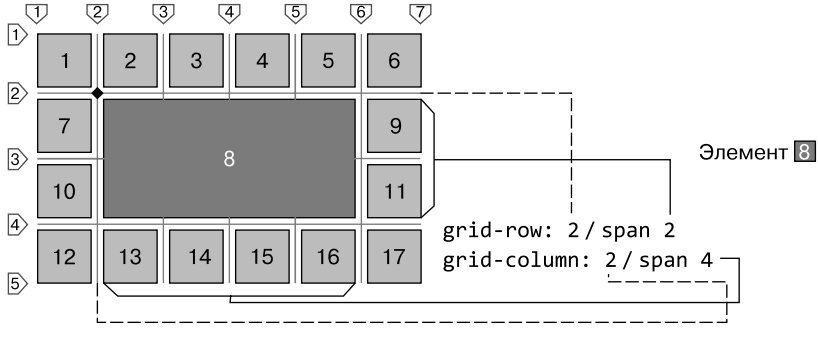


Здесь ширина столбца элемента превышает исходное количество столбцов, указанное в grid-макете.

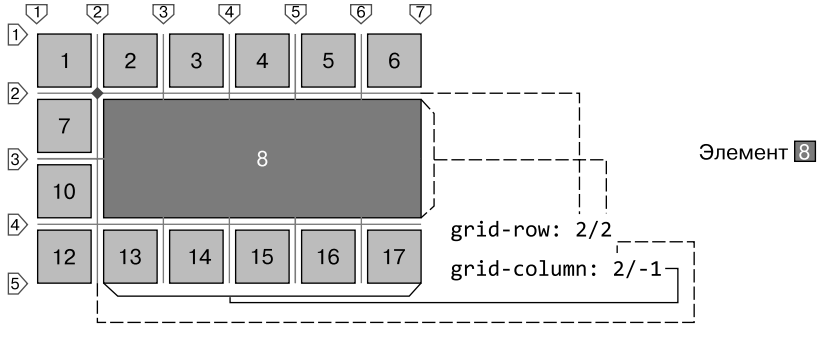
В таком случае grid-макет будет адаптироваться к тому, что вы видите в примере выше. Обычно хорошей идеей является разработка макетов с учетом границ grid-макета.

22.4. Краткая нотация диапазона

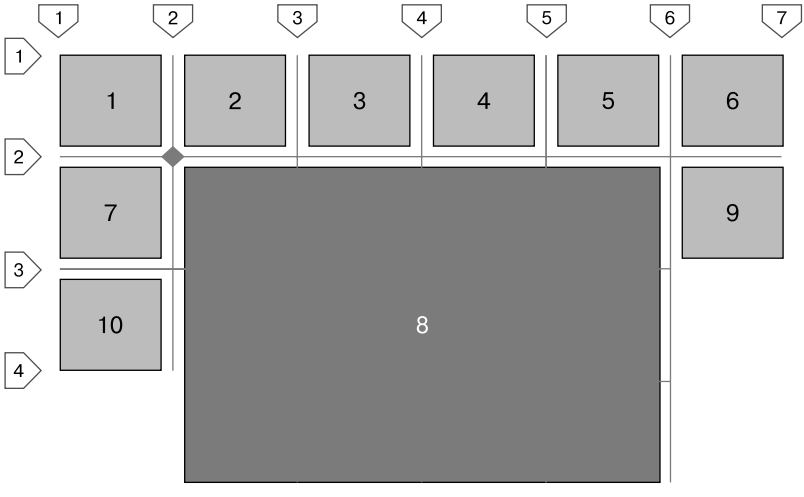
Вы можете применить сокращенные свойства `grid-row` и `grid-column` для достижения вышеописанного эффекта, используя символ `/` для разделения значений. За исключением того, что вместо конечного значения он принимает ширину или высоту диапазона.



Что, если нужно достичь абсолютной максимальной границы grid-макета? Используйте `-1`, чтобы расширить столбец (или строку) до конца grid-макета, когда число столбцов или строк неизвестно. Но помните: любые скрытые элементы (16, 17) удаляются из нижней части grid-макета.



Затем я попытался сделать то же самое со строками, но результаты были более хаотичными, в зависимости от того, какие комбинации значений я предоставил. Я знаю и о других способах использования символа /, но для ясности хотелось бы, чтобы все было просто.



В данном примере используется только десять элементов. Grid-макет, кажется, ненавязчиво меняет размеры.

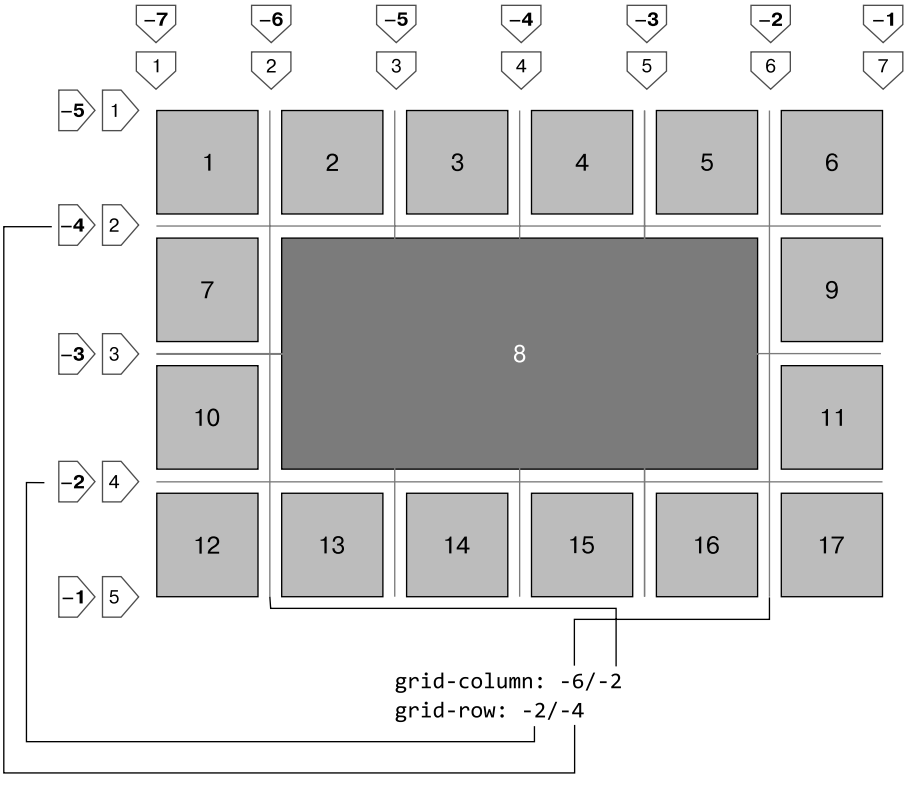
Когда я экспериментировал со строками для достижения того же эффекта, казалось, что значение в свойстве `grid-column: 2/4` нужно было изменить на `2/6`... но только при использовании свойства `grid-row: 2/-1`.

Я был слегка озадачен. Но предполагаю, что мне еще предстоит многое узнать о том, как работают значения, разделенные символом / .

Однако я обнаружил, что манипулирование значениями приводит к результатам, которые невозможно легко документировать с помощью визуальных рисунков.

Ну по крайней мере у нас есть основная идея. Вы можете расширить либо столбец, либо строку до максимума, используя `-1`. Только на практике вы поймете, как одно влияет на другое.

Мы можем немного подробнее рассмотреть схему. Grid-макет имеет вторичную систему координат. И, поскольку не имеет значения, в каком направлении вы используете перекрестные столбцы и переходы между строками, можно задействовать отрицательные значения.



Отрицательные значения, используемые для указания начала и конца столбца и строки, позволяют создать тот же диапазон из предыдущих примеров, поскольку grid-макет не зависит от системы координат. Можно применять как положительные, так и отрицательные числа!

Как видите, система координат grid-макета довольно гибкая.

22.5. Выравнивание контента в grid-элементах

Допустим, вы прошли долгий путь освоения диапазонов элементов grid-макета. Вы уже изучили большое количество неявно созданных строк и столбцов. Теперь вам интересно посмотреть, что еще ждет вас.

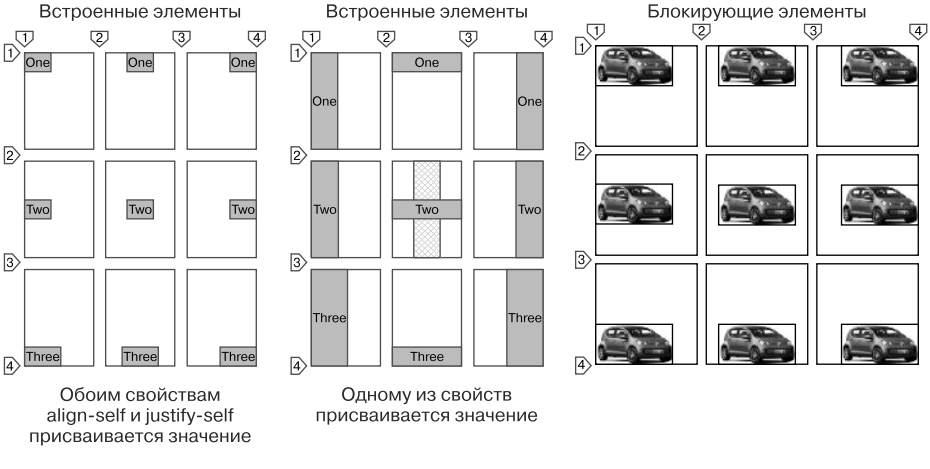
Хорошие новости.

Как веб-разработчик, я давно мечтал о разнонаправленном действии. Я хотел иметь возможность выполнять действия в середине и в любом углу контейнера.

Данная функциональность ограничена только свойствами `align-self` и `justify-self` и не работает ни с одним другим HTML-элементом. Если макет всего сайта построен с использованием grid-верстки, то это решает множество вопросов, связанных с размещением угловых и центральных элементов.

22.6. Свойство align-self

Рассмотрим пример использования свойств `align-self` и `justify-self`:

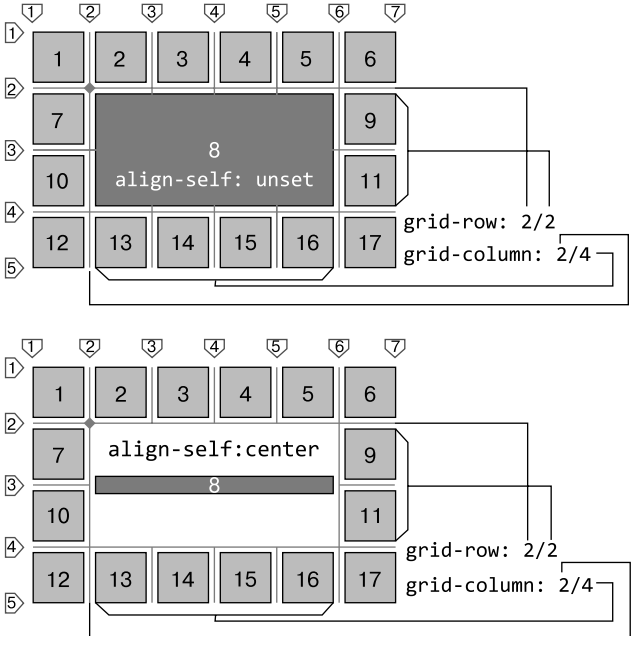


Разница между девятью квадратами — комбинация начальных и конечных значений, предоставленных указанным свойствам для получения любого из результатов, изображенных выше. Я не буду упоминать здесь все эти комбинации, поскольку это интуитивно понятно.

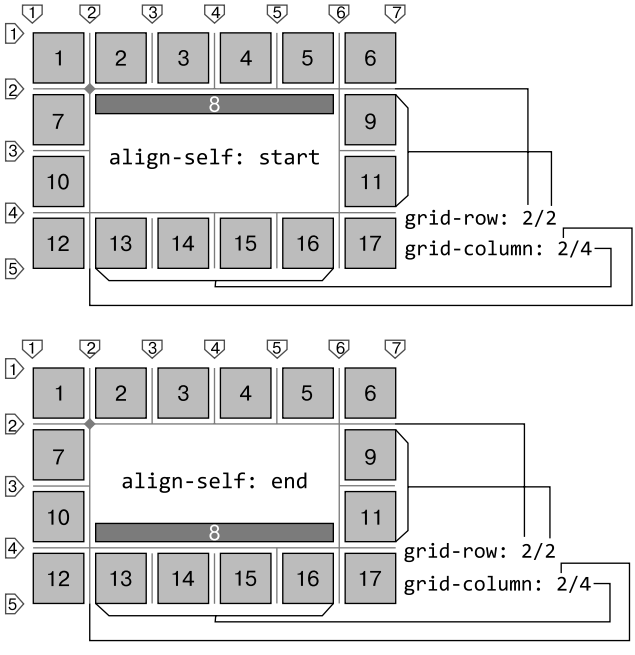
Вертикально: метод `align-self: end` позволяет выровнять содержимое по нижней части элемента. Аналогично `align-self: start` гарантирует, что содержимое будет придерживаться верхней границы.

Горизонтально: `justify-self: start` (или `end`) служит для выравнивания содержимого влево или вправо. В сочетании со свойством `align-self` можно добиться размещения, изображенного на любом из приведенных выше примеров.

В завершение нашего обсуждения рассмотрим такой пример.



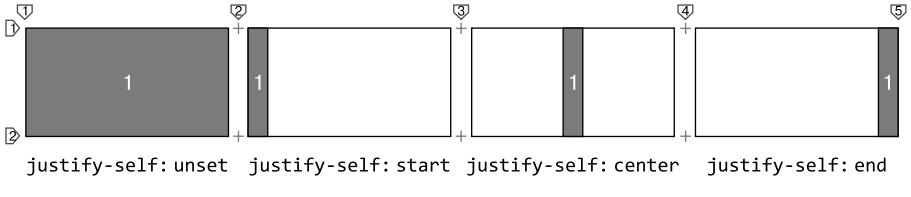
С помощью свойства `align-self` можно выровнять содержимое элемента согласно значениям `start`, `center` и `end`. Обратите внимание: у свойства `align-self` отсутствуют значения `top` и `bottom`.



22.7. Свойство justify-self

Еще одно свойство, которое делает то же самое, но горизонтально, — это `justify-self`.

Посмотрите на пример действия свойства `justify-self` с применением значений `unset`, `start`, `center` и `end`:



Здесь вы можете взаимозаменяемо использовать значения `start`|`left` или `end`|`right`.

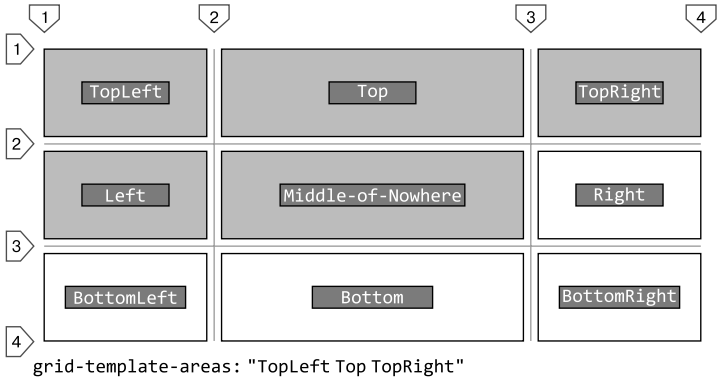
22.8. Шаблоны grid-областей

Шаблоны grid-областей предоставляют ссылку на изолированную часть grid-макета по предопределенному имени. Это имя не может включать пробелы.

Каждый набор имен строк заключен в двойные кавычки. Можно разделить эти наборы имен строк либо переносом на новую строку, либо пробелом, чтобы создать столбцы, как показано в примере ниже.

Хотя в шаблоне присутствует только пять элементов, имена могут занимать места, еще не заполненные элементами.

Рассмотрим пример указания шаблонов grid-областей с помощью свойства `grid-template-area`:



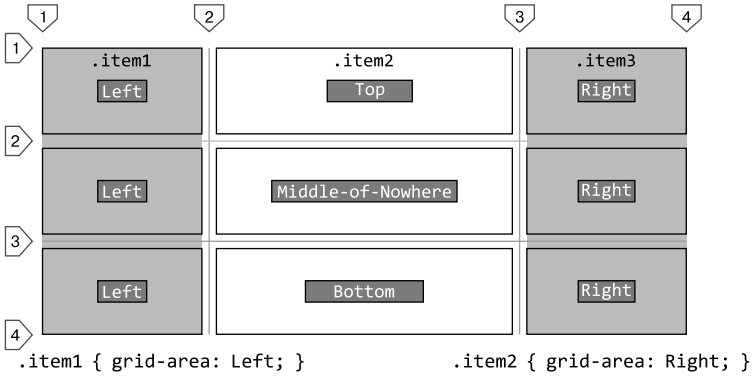

```
grid-template-areas: "TopLeft Top TopRight"
     "Left Middle-of-Nowhere Right" "BottomLeft Bottom BottomRight";
```

Вы можете указать область для любой строки и столбца, если разделяете набор каждой последующей строки пробелом и предоставляете имена для каждой строки с помощью двойных кавычек. В них каждый элемент разделен пробелом. То есть *в именах шаблонов grid-областей не должно быть пробелов*.

Чтобы присвоить имена всем областям в grid-макете, нужно следовать принципу, аналогичному указанию размера строки и столбца. Просто разделите их пробелом или табуляцией. Этот синтаксис позволяет интуитивно называть grid-области.

Но все становится намного удобнее, когда вы начинаете комбинировать области с одинаковыми именами в нескольких контейнерах. В примере ниже я назвал три элемента в левом столбце и три элемента в правом. Шаблоны grid-областей макета автоматически объединяют их, чтобы занимать одно и то же пространство по имени.

Далее показано распределение областей по нескольким ячейкам сетки. Просто присвойте имена столбцам и строкам, и соседние блоки сгруппируются в более крупные области. Просто убедитесь, что они прямоугольные!




```
.item1 { grid-area: Left; }
```

```
.item2 { grid-area: Right; }
```

```
grid-template-areas: "Left Top Right"
     "Left Middle-of-Nowhere Right" "Left Bottom Right";
```

Важно убедиться, что области состоят из элементов, выровненных по большим прямоугольным областям. *Делать блоки, как в тетрисе, здесь не получится*. Области должны быть прямоугольными, в противном случае результат может быть непредсказуемым.

22.9. Наименование линий сетки

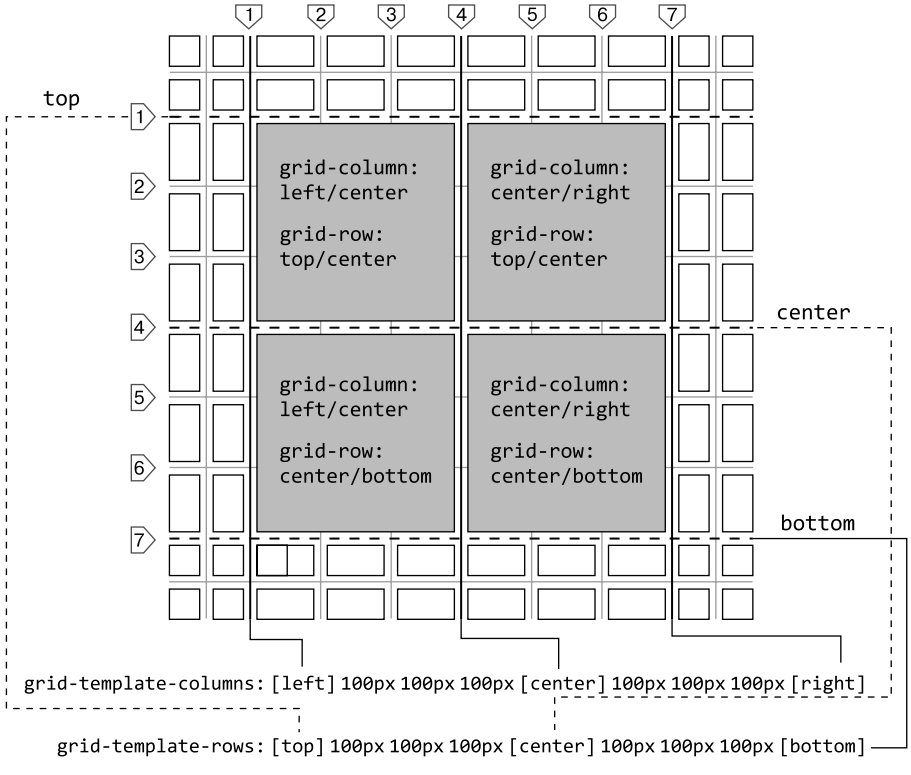
Работа с числами (и отрицательными числами) со временем может стать излишней, особенно при операциях со сложными grid-макетами. Строки grid-макета можно называть по своему усмотрению, используя скобки `[имя]` непосредственно перед значением размера.

Чтобы назвать первую линию grid-макета, можно использовать свойство `grid-template-column: [left] 100px`. Для строк подойдет свойство `grid-template-row: [top] 100px`.

Можно присвоить имена нескольким линиям grid-макета. Скобки `[]` вставляются в интуитивно понятное место в коде. Именно там, где должна появиться линия (промежуток) grid-макета:

```
grid-template-columns:[left] 5px 5px [middle] 5px 5px [right]
```

Теперь можно использовать имена `left`, `middle` и `right` для ссылки на линии grid-макета при создании столбцов и строк, которые должны достичь данной области.



Наименование линий промежутков создает более значимый навык. Целесообразно рассматривать среднюю линию в качестве *центра*

(или *середины*) вместо 4. В книге описано почти все, что касается grid-верстки, с помощью наглядных схематичных рисунков.

Вместо чисел можно называть линии между ячейками grid-макета как значения свойств `grid-template-columns` и `grid-template-row`. Обратите внимание: каждый диапазон на рисунке вышеотносится к именованным линиям, а не к номерам промежутка. Вы можете использовать любое имя.

В заключение... *помните...*

> Музыка не только в нотах, но и в тишине между ними.
>
> *Вольфганг Амадей Моцарт*

Это относится и к grid-верстке, и многому другому!

Почти восемь недель было потрачено на создание рисунков, отражающих практически все, что можно сделать с помощью grid-верстки. Я надеюсь, вы в них разобрались и все поняли.

Конечно, я что-то мог упустить, ведь невозможно описать абсолютно все потенциальные случаи. И я буду рад, если кто-нибудь укажет на это, чтобы улучшить будущие издания еще более полезными и интересными примерами.

23 Анимация

Анимация допустима для любого свойства CSS, физическое *положение*, *размеры*, *угол* или *цвет* которого могут быть изменены. Использовать базовую анимацию на основе ключевых кадров очень легко.

Ключевые кадры CSS-анимации задаются с помощью директивы `@keyframes`. Ключевой кадр — это просто состояние элемента в одной точке на временной шкале анимации.

CSS-анимация будет автоматически интерполировать анимационные ключевые кадры. Нужно лишь указать состояние свойств CSS в *начальной* и *конечной* точках анимации.

Как только все положения *ключевых кадров* настроены *(часто указываются в процентах)*, все, что нужно сделать, — это установить значения по умолчанию для исходного элемента, который вы хотите анимировать.

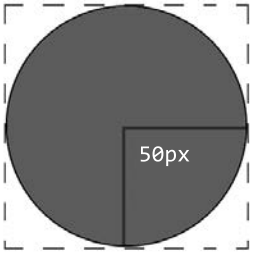
Затем создайте именованную анимацию, используя формат `@keyframes` *`имяАнимации`* `{...}`, в котором будут храниться все ключевые кадры. Поговорим об этом буквально через секунду!

Наконец, создайте специальный класс, который станет определять *продолжительность*, *направление*, *повторяемость* и *динамику* вашей анимации... и связывать его с тем же именем анимации, которое использовалось директивой `@keyframes`.

Превратим *желтый квадрат* в *бирюзовый круг*. Как только класс `.классАнимации` назначен элементу, анимация начнет воспроизводиться. Класс ссылается на `имяАнимации`. Это значение должно совпадать с именем, заданным директивой `@keyframes`. Анимация настроена на 3 с, или 3000 мс. Примечание: динамика добавляет изюминку вашей анимации с помощью кривой, описывающей относительную скорость анимации в определенном месте на временной шкале.

Исходное изображение

```
width: 100px;
height: 100px;
border: 1px solid black;
background: yellow;
```



Полученное изображение

```
border-radius: 50px;
background: teal;
```

```
@keyframes имяАнимации {
    0% { border-radius: 0; background: yellow; }
  100% { border-radius: 50px; background: teal; }
}

.классАнимации {
    animation-name: имяАнимации;
    animation-fill-mode: forwards;
    animation: normal 3000ms ease;
}
```

Далее рассмотрим *сглаживание (плавность)* и все другие свойства CSS-анимаций на основе простого примера.

23.1. Свойство animation

Данное свойство является сокращенной записью восьми свойств анимации, описанных ниже:

- `animation-name` — имя ключевого кадра, заданного директивой `@keyframes`;
- `animation-duration` — продолжительность одного цикла анимации в миллисекундах;
- `animation-timing-function` — описывает плавность воспроизведения анимации между каждой парой ключевых кадров;
- `animation-delay` — добавляет задержку до начала воспроизведения анимации;
- `animation-iteration-count` — устанавливает, сколько раз анимация должна проигрываться;
- `animation-direction` — определяет воспроизведение вперед, назад или в случайной последовательности;
- `animation-fill-mode` — выясняет состояние анимации, когда она не воспроизводится;
- `animation-play-state` — определяет, запущена анимация или приостановлена.

В следующих разделах мы наглядно рассмотрим каждое из перечисленных свойств.

23.2. Свойство animation-name

Буквенно-цифровое имя идентификатора анимации:

```
001 .классАнимации {
002     animation-name: имяАнимации;
003     animation-fill-mode: normal;
004     animation: normal 3000ms ease-in;
005 }
```

Имя анимации должно совпадать с именем, указанным в директиве `@keyframes`:

```
001 @keyframes имяАнимации {
002     0% { }
003   100% { }
004 }
```

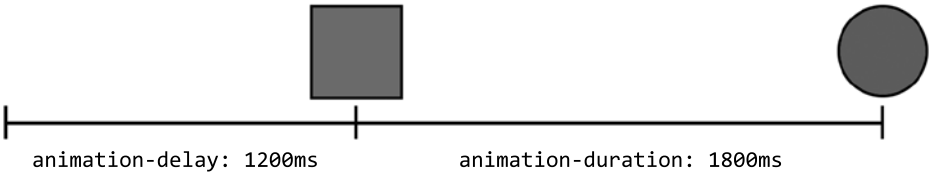
23.3. Свойство animation-duration

Обычно в первую очередь указывается продолжительность одного цикла анимации.

Можно также указать длительность в секундах или миллисекундах, если необходима большая точность. Например, 3000 мс — это 3 с, а 1500 мс — 1,5 с.

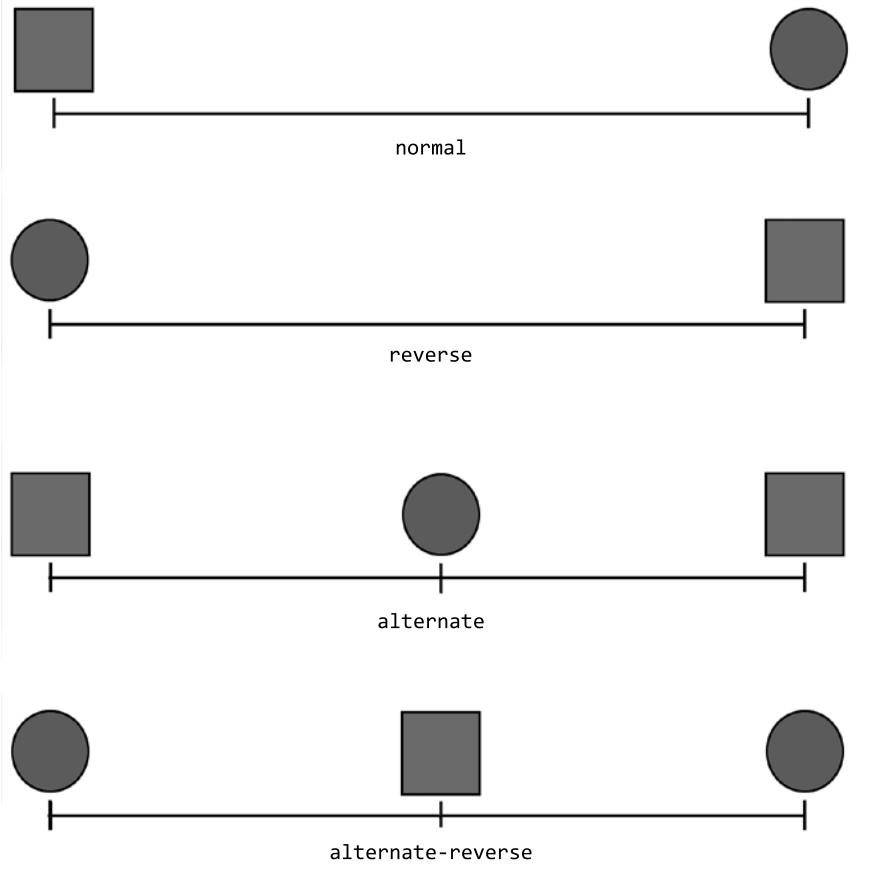
23.4. Свойство animation-delay

Если анимация должна воспроизводиться не сразу, то можно добавить задержку, например установить время задержки в миллисекундах до начала воспроизведения анимации.



23.5. Свойство animation-direction

Можно назначить любое из четырех значений свойству `animation-direction`.



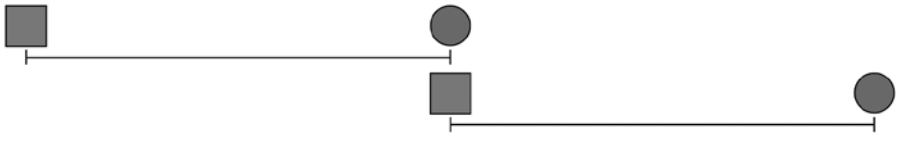
CSS-анимация будет автоматически интерполироваться между кадрами. Интерполированное состояние анимации — это любое состояние между любыми двумя кадрами. По мере того как цвет переходит от желтого к бирюзовому, он будет постепенно меняться в течение периода времени, заданного свойством `animation` (здесь приведена сокращенная нотация).

23.6. Свойство animation-iteration-count

Определяет количество повторений анимации. Рассмотрим примеры.

Воспроизведение анимации один раз (по умолчанию):

Воспроизведение анимации два раза:



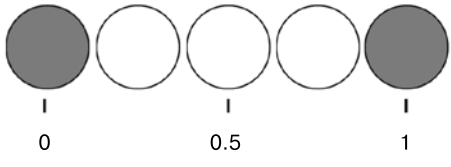
Воспроизведение анимации три раза:

Как видите, очевидная проблема заключается в том, что анимация снова «перепрыгнет» обратно на первый кадр.

Некоторые другие свойства анимации позволяют убедиться, что такой переход не происходит. Вы можете создать цикличную анимацию и настраивать другие свойства в зависимости от конкретной желаемой динамики пользовательского интерфейса. Таким образом, вы можете создавать только «половину» вашей анимации и настраивать свойства для воспроизведения ее вперед или назад, например, при событиях `mouse-in` и `mouse-out`.

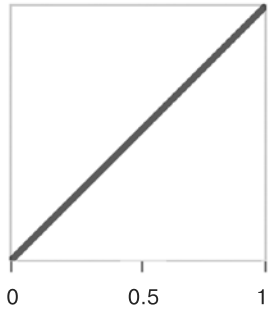
23.7. Свойство animation-timing-function

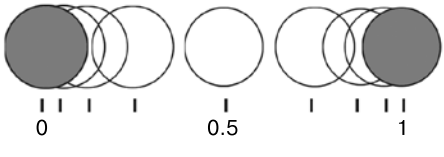
Динамика задается свойством `animation-timing-function`, которое придает вашей анимации индивидуальность. Это делается путем регулировки скорости анимации в любой заданной точке на временной шкале. Важными являются начальная, средняя и конечная точки. Каждый тип замедления определяется функцией *кривой Безье*.




```
animation-timing-function: linear
    cubic-bezier(0, 0, 1, 1);
```

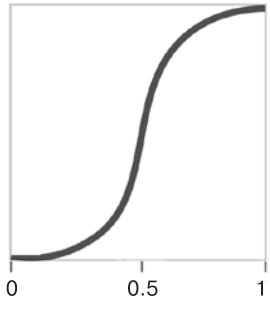
Сохраняет постоянную скорость от начала до конца

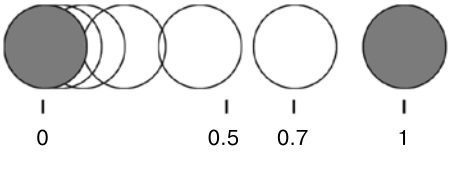





```
animation-timing-function: ease
cubic-bezier(0.25, 0.1, 0.25, 1);
```

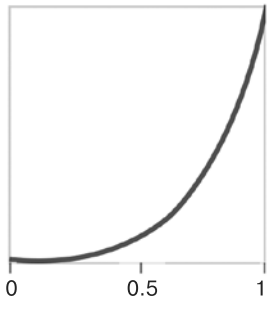
Начинается медленно, ускоряется, замедляется в конце <по умолчанию>

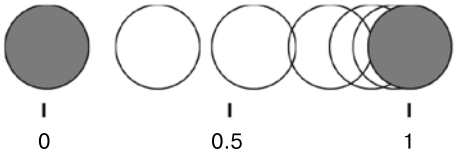





```
animation-timing-function: ease-in
   cubic-bezier(0.42, 0, 1, 1);
```

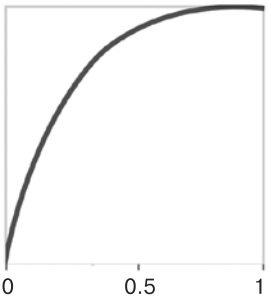
Устанавливает эффект перехода с медленным стартом

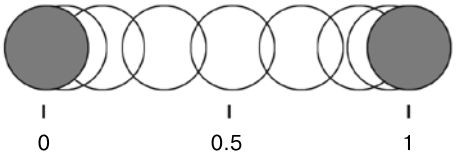





```
animation-timing-function: ease-out
    cubic-bezier(0, 0, 0.58, 1);
```

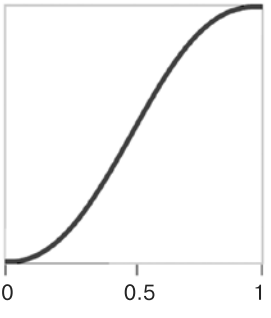
Поначалу быстро и замедляется ближе к концу






```
animation-timing-function: ease-in-out
    cubic-bezier(0.42, 0, 0.58, 1);
```

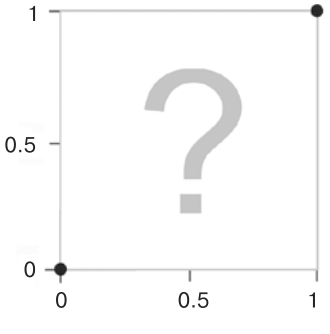
Определяет эффект перехода с медленным началом и концом



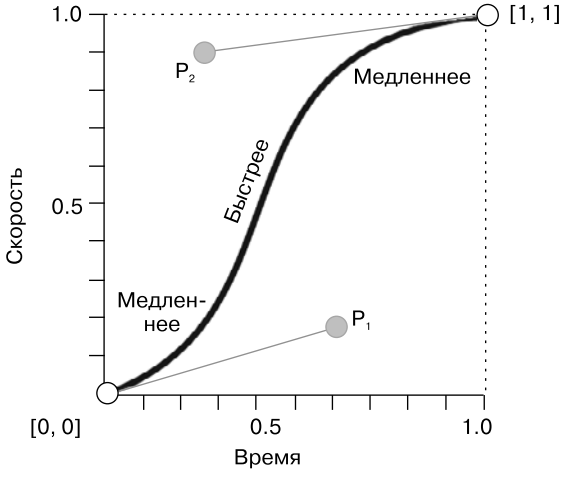
Вы можете создавать собственные кубические кривые Безье:


```
cubic-bezier(P1.x, P1.y, P2.x, P2.y);
```

Определение пользовательского значения в функции кубической кривой Безье



Как же это работает? Две контрольные точки P_1 и P_2 передаются функции `cubic-bezier` в качестве аргументов. Диапазон значений составляет от `0.0` до `1.0`.



Поскольку динамика устанавливается уравнением, вы можете предоставить собственные аргументы, чтобы создать уникальные кривые для достижения конкретного типа скорости, недоступного по предопределенным значениям.

Как показано на рисунках ниже, можно воссоздать стандартный набор значений с помощью функции `cubic-bezier`:

```
.linear {
  animation-timing-function: cubic-bezier(0, 0, 1, 1);
}

.ease {
  animation-timing-function: cubic-bezier(0.25, 0.1, 0.25, 1);
}

.ease-in {
  animation-timing-function: cubic-bezier(0.42, 0, 1, 1);
}

.ease-out {
  animation-timing-function: cubic-bezier(0, 0, 0.58, 1);
}

.ease-in-out {
  animation-timing-function: cubic-bezier(0.42, 0, 0.58, 1);
}
```

Если нужна немного другая кривая для ваших элементов пользовательского интерфейса, то попробуйте поэкспериментировать со значениями, пока не достигнете желаемого эффекта.

23.8. Свойство animation-fill-mode

Если анимация в настоящий момент не воспроизводится, то устанавливается в режим fill mode. Свойство `animation-fill-mode` стилизует анимацию, которая не воспроизводится, выбранным набором свойств, обычно взятых из первого или последнего ключевых кадров.

Допустимые значения:

- `none` — к анимированному элементу не применяются какие-либо стили до или после выполнения анимации;
- `forwards` — стили сохраняются из последнего ключевого кадра (может зависеть от значений свойств `animation-direction` и `animation-iteration-count`);
- `backwards` — стили извлекаются из первого ключевого кадра (может зависеть от значения свойства `animation-direction`), также сохраняют стиль во время состояния `animation-delay`;
- `both` — расширение свойств анимации в обоих направлениях (`forwards` и `backwards`).

23.9. Свойство animation-play-state

Свойство указывает, воспроизводится анимация или приостановлена.

Допустимые значения:

- `paused` — анимация приостановлена;
- `running` — анимация воспроизводится.

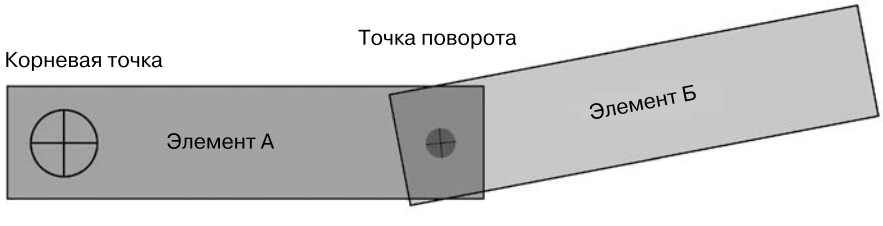
Например, можно приостановить анимацию при наведении указателя:

```
001 div:hover {
002   animation-play-state: paused;
003 }
```

24 Прямая и обратная кинематика

В CSS отсутствует встроенная поддержка обратной кинематики. Но эффект может быть смоделирован с помощью свойств `transform: rotate(`*`градусы`*`)` и `transform-origin`, чтобы указать точку вращения между родительским и дочерним элементами.

Прямая и обратная кинематика — это метод преобразования угла поворота между несколькими объектами, прикрепленными друг к другу в точке поворота. Кинематика часто используется при моделировании физики в видеоиграх. Но можно задействовать тот же принцип для анимации двумерных персонажей.



Корневая точка — это позиция, где основной элемент присоединен либо к другому родительскому элементу, либо к воображаемой статичной точке в пространстве. Если элемент А движется, то должен воздействовать на элемент Б таким образом, как если бы они были прикреплены друг к другу в точке поворота. То есть все виды углов и длин вычисляются с применением тригонометрических формул. Мы можем сделать это, задействуя JavaScript или существующую

библиотеку векторной тригонометрии. Но, к счастью, CSS уже обеспечивает поддержку данных типов динамики элементов с помощью свойства `transform-origin`.

- **Прямая кинематика** — это когда при перемещении **элемента А** отражается и движение **элемента Б** (как цепная реакция), словно они прикреплены друг к другу в общей точке поворота.
- **Обратная кинематика** противоположна прямой: физическое движение **элемента Б** воздействует на **элемент А** при условии, что элемент Б присоединен к некоторой статичной точке или к другому родительскому элементу. Если нет, то два элемента могут плавать в пространстве ☺.

Это очень похоже на подвижные соединения костей анимационного персонажа!

25 Принципы SASS/SCSS

В этой главе мы начнем с простых принципов SCSS и будем опираться на них, двигаясь в сторону расширенных директив. У вас не получится по достоинству оценить возможности SASS, пока вы не создадите свой первый *цикл* `for` в CSS.

Вы даже можете написать собственные функции синуса и косинуса (тригонометрия), используя синтаксис SASS/SCSS. Мы знаем, что они зависят от числа π. Нет проблем! Переменные SASS также могут хранить числа с плавающей точкой: `$PI: 3.14159265359`.

Удивительно, но знание тригонометрии (даже на ее базовом уровне) может оказаться весьма полезным при создании анимированных элементов пользовательского интерфейса. Особенно когда задействованы анимация или вращение.

Это учебное пособие позволит познакомиться с наиболее важными особенностями SASS/SCSS. Итак, начнем!

25.1. Новый синтаксис

SCSS на самом деле не добавляет никаких новых функций в язык CSS, только новый синтаксис, который может сэкономить время, затрачиваемое на написание кода. В данной главе мы рассмотрим новые дополнения к синтаксису языка CSS.

Это не полное учебное пособие по SASS/SCSS. Но, чтобы начать пользоваться преимуществами SASS, вам не нужно знать все. Не-

обходимо изучить только основы. Они будут рассмотрены в следующих разделах данной главы.

В ряде случаев SCSS и SASS будут использоваться взаимозаменяемо, хотя синтаксис немного отличается. Однако прежде всего мы сосредоточимся на SCSS.

Весь код SASS/SCSS компилируется обратно в стандартный CSS, поэтому браузер может реально понимать и отображать результаты (сегодня браузеры не имеют прямой поддержки SASS/SCSS или любого другого препроцессора CSS).

25.2. Необходимые условия

Препроцессоры CSS добавляют новые функции к *синтаксису* языка CSS.

Существует пять препроцессоров CSS: **SASS**, **SCSS**, **Less**, **Stylus** и **PostCSS**.

Данная глава охватывает только SCSS, который подобен SASS. Больше узнать о SASS можно на сайте www.SASS-lang.com.

- SASS (`.SASS`) — Syntactically Awesome Style Sheets.
- SCSS (`.scss`) — SASSy Cascading Style Sheets.

Расширения `.SASS` и `.scss` похожи, но не одинаковы. Можно конвертировать из формата `.SASS` в `.scss` и обратно.

```
001 # Преобразование Sass в SCSS
002 $ sass-convert style.sass style.scss
003
004 # Преобразование SCSS в Sass
005 $ sass-convert style.scss style.sass
```

SASS являлась первой спецификацией для SASSy CSS с расширением файла `.SASS`. Разработка началась в 2006 году. Но позже был разработан альтернативный синтаксис с расширением `.scss`, лучший по мнению некоторых разработчиков.

Независимо от используемого синтаксиса или расширения SASS, в настоящее время Sassy CSS не получило широкой поддержки в браузерах. Но вы можете экспериментировать с любым из пяти препроцессоров на платформе codepen.io. Кроме того, вам необходимо установить препроцессор CSS на вашем веб-сервере.

Эта глава была написана, чтобы помочь вам познакомиться с SCSS. Другие препроцессоры имеют схожие характеристики, но синтаксис может быть другим.

25.3. Расширенные возможности

SASSy CSS в любом своем проявлении является надмножеством языка CSS. Это означает следующее: все, что работает в CSS, все еще будет работать в SASS или SCSS.

25.4. Переменные

SASS/SCSS позволяет работать с переменными. Они отличаются от переменных CSS, которые начинаются с двух дефисов (`--var-color`), тем, что начинаются со знака доллара (`$`).

```
001 $number: 1;
002 $color: #FF0000;
003 $text: "Кусочек строки."
004 $text: "Другая строка." !default;
005 $nothing: null;
```

Вы можете перезаписать имя переменной. Если `!default` добавляется к переопределению переменной и та уже существует, то не назначается повторно. Другими словами, это значит, что конечное значение переменной `$text` из данного примера все равно будет `"Кусочек строки."`.

Второе назначение — `"Другая строка."` — игнорируется, поскольку значение по умолчанию уже существует.

Переменные SASS могут быть назначены любому свойству CSS:

```
001 #container {
002     content: $text;
003 }
```

25.5. Вложенные правила

В стандартном CSS доступ к вложенным элементам осуществляется через пробел.

```
001 /* Стандартный CSS */
002 #A {
003     color: red;
004 }
005
006 #A #B {
007     color: green;
008 }
009
010 #A #B #C p {
011     color: blue;
012 }
```

Приведенный выше код можно выразить с помощью вложенных правил SASSy следующим образом.

```
001 /* Вложенные правила */
002 #A {
003     color: red;
004     #B {
005         color: green;
006         #C p {
007               color: blue;
008         }
009     }
010 }
```

Как видите, данный синтаксис выглядит чище и менее повторяющимся.

Это особенно полезно для управления сложными макетами. Таким образом, выравнивание, в котором вложенные свойства CSS записываются в коде, близко соответствует фактической структуре макета приложения.

Препроцессор все еще компилирует это в стандартный CSS-код (показанный выше), чтобы тот мог фактически отображаться в браузере. Мы просто меняем способ написания CSS.

25.6. Директива &

SASSy CSS допускает символьную директиву `&` (и). Посмотрим, как это работает. В строке 5 символ `&` был использован для указания `&:hover` и преобразован в имя родительского элемента (`a`) после компиляции:

```
001 #P {
002    color: black;
003    a {
004        font-weight: bold;
005        &:hover {
006            color: red;
007        }
008    }
009 }
```

Символ `&` преобразуется в имя родительского элемента и становится `a:hover`:

```
#P { color: black; }
#P a { font-weight: bold; }
#P a:hover { color: red; } // & Компилировано в (родительский элемент)
```

25.7. Примеси

Примесь определяется директивой `@mixin`.

Создадим нашу первую примесь `@mixin`, определяющую поведение flex-элемента по умолчанию.

```
001 @mixin flexible() {
002     display: flex;
003     justify-content: center;
004     align-items: center;
005 }
006
007 .centered-elements {
008     @include flexible();
009     border: 1px solid gray;
010 }
```

Теперь каждый раз, когда вы применяете класс `.centered-elements` к HTML-элементу, последний превращается во flex-элемент. Одним из ключевых преимуществ примесей является то, что их можно использовать вместе с другими свойствами CSS. Здесь я также добавил стиль `border: 1px solid gray` к классу `.centered-elements` в дополнение к примеси.

Вы даже можете передать аргументы `@mixin`, как если бы это была функция, а затем назначить их свойствам CSS. Мы рассмотрим данный процесс в следующем разделе.

25.8. Поддержка разных браузеров

Некоторые экспериментальные функции (например, `-webkit-`*`свойства`*) или Firefox (`-moz-`*`свойства`*) работают только в отдельных браузерах.

Примеси могут быть полезны для определения специфичных для браузера свойств CSS в одном классе. Например, при необходимости повернуть элемент в браузерах на движке Webkit, а также в других можно создать примесь, принимающую аргумент `$degree`.

```
001 @mixin rotate($degree) {
002     -webkit-transform: rotate($degree); // движок WebKit
003     -moz-transform: rotate($degree);    // Firefox
004     -ms-transform: rotate($degree);     // Internet Explorer
005     -o-transform: rotate($degree);      // Opera
006     transform: rotate($degree);         // Стандарт CSS
007 }
```

Теперь все, что нужно сделать, — это с помощью директивы `@include` вложить данную примесь в наше определение класса CSS. Этот поворот сработает во всех браузерах:

```
001 .rotate-element {
002     @include rotate (45deg);
003 }
```

25.9. Арифметические операторы

Подобно стандартному синтаксису CSS, вы можете *складывать*, *вычитать*, *умножать* и *делить* значения, не используя функцию `calc()` из классического синтаксиса CSS.

Но есть несколько неочевидных случаев, приводящих к ошибкам. Далее рассмотрим их.

Сложение

```
001 p {
002     font-size: 10px + 2em; // *ошибка: несовместимые элементы
003     font-size: 10px + 6px; // 16px
004     font-size: 10px + 6;   // 16px
005 }
```

Это сложение значений без использования функции `calc()`. Убедитесь, что оба значения представлены в соответствующем формате.

Вычитание

Оператор вычитания (`-`) работает точно так же, как сложение.

```
001 div {
002     height: 12% - 2%;
003     margin: 4rem - 1;
004 }
```

Умножение

Умножение и деление (последний пример):

```
001 p {
002     width: 10px * 10px;          // *ошибка
003     width: 10px * 10;            // 100px
004     width: 1px * 5 + 5px;        // 10px
005     width: 5 * (5px + 5px);      // 50px
006     width: 5px + (10px / 2) * 3; // 20px
007 }
```

Деление

Деление немного сложнее, поскольку в стандартном CSS-коде символ деления зарезервирован для использования вместе с некоторыми сокращенными свойствами. Согласно SCSS, этот символ совместим со стандартным CSS.

```
001 p { font: 16px / 24px Arial, sans-serif; }
```

В стандартном CSS-коде символ деления используется в сокращенном свойстве `font`. Но данное свойство не применяется для фактического разделения значений. Как же SASS справляется с делением?

Если необходимо разделить два значения, то просто добавьте круглые скобки вокруг операции деления. В противном случае оно будет работать только в сочетании с некоторыми другими операторами или функциями.

```
001 p {
002     top: 16px / 24px     // Выходные значения (классический CSS)
003     top: (16px / 24px)   // Деление (когда добавляются скобки)
004     top: #{$varl} / #{$var2}; // Использование интерполяции,
005                               // выходные данные как CSS
005     top: $varl / $var2;       // Деление
006     top: random(4) / 5;       // Деление (в паре с функцией)
007     top: 2px / 4px + 3px      // Деление (часть арифметики)
008 }
```

Остаток

Оставшаяся часть вычисляет остаток от операции деления. В данном примере посмотрим, как его можно использовать при создании шаблона «полоски зебры» для произвольного набора HTML-элементов.

Начнем с создания примеси `zebra`. Примечание: директивы `@for` и `@if` обсуждаются в следующем разделе.

```
001 @mixin zebra(); {
002     @for $i from 1 through 7 {
003         @if ($i % 2 == 1) {
004             .stripe-#{$i} {
005                 background-color: black;
006                 color: white;
007             }
008         }
009     }
010 }
011
012 * { @include zebra(); }
```

Для этой демонстрации требуется несколько HTML-элементов.

```
001 <div class = "stripe-1">zebra</div>
002 <div class = "stripe-2">zebra</div>
003 <div class = "stripe-3">zebra</div>
004 <div class = "stripe-4">zebra</div>
005 <div class = "stripe-5">zebra</div>
006 <div class = "stripe-6">zebra</div>
007 <div class = "stripe-7">zebra</div>
```

И вот результат в браузере — чередующиеся полосы, созданные примесью `zebra`:

zebra
zebra
zebra
zebra
zebra
zebra
zebra
zebra

Операторы сравнения

Оператор	Пример	Описание
==	x==y	Возвращает значение true (истина) при x, равном y
!=	x!=y	Возвращает значение true (истина), если x и y не равны
>	x>y	Возвращает значение true (истина), если x больше y
<	x<y	Возвращает значение true (истина), если x меньше y
>=	x>=y	Возвращает значение true (истина), если x больше или равно y
<=	x<=y	Возвращает значение true (истина), если x меньше или равно y

Как использовать операторы сравнения на практике? Можно написать примесь, которая выберет размер заполнения, если его значение больше значения внешнего отступа.

```
001 @mixin spacing($padding, $margin) {
002     @if ($padding > $margin) {
003         padding: $padding;
004     } @else {
005         padding: $margin;
006     }
007 }
008
009 .container {
010     @include spacing(10px, 20px);
011 }
```

После компиляции мы получим следующий код.

```
001 .container { padding: 20px; }
```

Логические операторы

Оператор	Пример	Описание
and	x and y	Возвращает значение true (истина), если x и y true
or	x or y	Возвращает значение true (истина), если x или y true
not	x not y	Возвращает значение true (истина), если x не true

Использование логических операторов SASS для создания класса цвета кнопки, который меняет цвет фона в зависимости от его ширины:

```
001 @mixin button-color($height, $width) {
002     @if(($height < $width) and ($width >= 35px)) {
003         background-color: blue;
004     } @else {
005         background-color: green;
006     }
007 }
008
009 .button {
010     @include button-color (20px, 30px)
011 }
```

Строки

В некоторых случаях можно добавить строки в допустимые значения CSS без кавычек.

Комбинирование обычных значений свойств CSS со строками SASS/SCSS:

```
001 p {
002     font: 50px Ari + "al"; // Получается 50px Arial
003 }
```

Следующий пример, с другой стороны, приведет к ошибке компиляции:

```
001 p {
002     font: "50px " + Arial; // Ошибка
003 }
```

Можно добавлять строки без двойных кавычек, если строка не содержит пробелов. Например, следующий пример не будет компилироваться.

```
001 p:after {
002     content: "Строка в кавычках и " + вот такой хвостик.;
003 }
```

Что делать? Строки, содержащие пробелы, должны быть заключены в кавычки:

```
001 p:after {
002     content: "Строка в кавычках и " + "вот такой хвостик.";
003 }
```

Добавляем несколько строк:

```
001 p:after {
002     content: "Очень " + "длинная " + "строка";
003 }
```

Добавляем числа и строки:

```
001 p:after {
002     content: "Длинная " + 1234567 + "строка";
003 }
```

Обратите внимание: свойство `content` работает только с псевдоселекторами `:before` и `:after`. Рекомендуется избегать использования свойства `content` в ваших определениях CSS. Всегда указывайте содержимое между HTML-элементами. Эти рекомендации приведены в книге только в контексте работы со строками в SASS/SCSS.

25.10. Операторы управления потоком

SCSS содержит *функции*`()` и *`@`директивы*. Мы уже создавали функции, когда рассматривали примеси, в которые можно передавать аргументы. У функции обычно есть скобки, добавляемые в конец имени. Директива начинается с символа `@`.

Как и в JavaScript или других языках, SCSS позволяет работать со стандартным набором *операторов управления потоком*.

Функция if()

`if()` — это функция.

Ее синтаксис довольно примитивен. Оператор вернет одно из двух указанных значений согласно условию:

```
001 /* Применение функции if() */
002 if(true, 1px, 2px) => 1px
003 if(false, 1px, 2px) => 2px
```

Директива @if

`@if` — это директива, используемая для ветвления в зависимости от условия.

```
001 /* Применение директивы @if */
002 p {
003   @if 1 + 1 == 2 { border: 1px solid;  }
004   @if 7 < 5      { border: 2px dotted; }
005   @if null       { border: 3px double; }
006 }
```

Оператор `if` в SASSy состоит из:

```
001 p { border: 1px solid; }
```

Пример использования одного оператора `if` или конструкции `if-else`:

```
001 /* Создание переменной $type */
002 $type: river;
003
004 /* Окрашивание в синий, если переменной присвоено
005    значение river */
006 div {
007     @if $type == river {
008         color: blue;
009     }
010 }
011
012 /* Условные цвета по абзацу */
```

```
012 p {
013     @if $type == tree {
014         color: green;
015     } @else if $type == river {
016         color: blue;
017     } @else if $type == dirt {
018         color: brown;
019 }
020 }
```

Проверка наличия родительского элемента

Оператор `&` выбирает родительский элемент, если тот существует. Или в противном случае возвращает значение `null`. Поэтому его можно использовать в сочетании с директивой `@if`.

В следующих примерах посмотрим, как создавать условные стили CSS на основе того, существует родительский элемент или нет.

Если родитель не существует и оператор `&` возвращает `null`, то будет использоваться альтернативный стиль:

```
001 /* Проверка существования родительского элемента */
002 @mixin does-parent-exist {
003     @if & {
004         /* Если существует, применить синий цвет
005            к родительскому элементу */
006         &:hover {
007             color: blue;
008         }
008     } @else {
010         /* Если не существует, применить синий цвет
011            к ссылкам */
012         a {
013             color: blue;
014         }
015     }
016 }
017
018 p {
019     @include does-parent-exist();
020 }
```

Директива @for

Эта директива может использоваться для повторения определений CSS несколько раз подряд.

Цикл `for` для пяти элементов:

```
001 @for $i from 1 through 5 {
002     .definition-#{$i} { width: 10px * $i; }
003 }
```

Данный цикл будет скомпилирован в следующий код CSS:

```
001 .definition-1 { width: 10px; }
002 .definition-2 { width: 20px; }
003 .definition-3 { width: 30px; }
004 .definition-4 { width: 40px; }
005 .definition-5 { width: 50px; }
```

Директива @each

Эта директива может использоваться для перебора списка значений:

```
001 @each $animal in platypus, lion, sheep, dove {
002     .#{$animal}-icon {
003         background-image: url("/images/#{$animal}.png");
004     }
005 }
```

Этот код будет скомпилирован в следующий код CSS:

```
001 .platypus-icon {
002     background-image: url("/images/platypus}.png");
003 }
004 .lion-icon {
005     background-image: url("/images/lion.png");
006 }
007 .sheep-icon {
008     background-image: url("/images/sheep.png");
009 }
010 .dove-icon {
011     background-image: url("/images/dove.png");
012 }
```

Цикл @while

```
001 $index: 5;
002 @while $index > 0 {
003     .element-#{$index} { width: 10px * $index; }
004     $index: $index - 1;
005 }
```

Список из пяти HTML-элементов, созданных циклом `while`:

```
001 .element-5 { width: 50px; }
002 .element-4 { width: 40px; }
003 .element-3 { width: 30px; }
004 .element-2 { width: 20px; }
005 .element-1 { width: 10px; }
```

25.11. Функции SASS

С помощью SASS/SCSS можно определять функции, как и на любом другом языке.

Создадим функцию `three-hundred-px()`, возвращающую значение `300px`:

```
001 @function three-hundred-px() {
002     @return 300px;
003 }
004
005 .name {
006     width: three-hundred-px();
007     border: 1px solid gray;
008     display: block;
009     position: absolute;
010 }
```

```
001 <div class = "name">Hello.</div>
```

Когда класс `.name` применяется к элементу, ему будет присвоено значение ширины `300px`.

Функции SASS могут возвращать любое допустимое значение CSS и назначаться любому свойству CSS. Их даже можно рассчитать на основе переданного аргумента.

```
001 @function double($width) {
002 return $width * 2;
003 }
```

25.12. Тригонометрические функции SASS

Тригонометрические функции `sin()` и `cos()` часто встречаются в качестве составляющей встроенных классов во многих языках, таких как, например, JavaScript.

Я думаю, изучение принципов их работы стоит того, если вы хотите сократить время, затрачиваемое на создание анимации пользовательского интерфейса (например, вращающегося индикатора).

Далее мы рассмотрим пару примеров, которые сводят код к минимуму для создания интересных анимационных эффектов с помощью функции `sin()`. Те же принципы можно расширить и использовать в целях создания интерактивных элементов пользовательского интерфейса.

Применение тригонометрии для создания CSS-анимации — отличный способ уменьшить пропускную способность. Со временем CSS-анимация вытеснила GIF-анимацию (`.gif`) (для загрузки каждой из них может потребоваться дополнительный HTTP-запрос, поскольку GIF-анимации (`.gif`) нельзя поместить в одно изображение).

В SASS вы можете писать собственные тригонометрические функции.

25.13. Пользовательские функции SASS

В данном разделе приведены примеры создания собственных функций в SASS/SCSS. В тригонометрии многие операции основаны на таких функциях. Все они строятся друг на друге. Например, функция `rad()` требует наличия функции `PI()`. Для работы функций `cos()` и `sin()` требуется функция `rad()`.

```
001 @function PI() { @return 3.14159265359; }
```

Написание функций в SASS/SCSS очень похоже на написание функций на JavaScript или аналогичных языках программирования.

```
001 @function pow($number, $exp) {
002   $value: 1;
003   @if $exp > 0 {
004     @for $i from 1 through $exp {
005       $value: $value * $number;
006     }
007   }
008   @else if $exp < 0 {
009     @for $i from 1 through -$exp {
010       $value: $value / $number;
011     }
012   }
013   @return $value;
014 }
```

```
001 @function rad($angle) {
002   $unit: unit($angle);
003   $unitless: Wangle / ($angle *0+1);
004   // Если угол в градусах, то необходимо перевести в радианы
005   @if $unit == deg {
006     $unitless: $unitless / 180 * PI();
007   }
008   @return $unitless;
009 }
```

```
001 @function sin($angle) {
002   $sin: 0;
003   $angle: rad($angle);
004   // Повторить 10 раз
005   @for $i from 0 through 10 {
006     $fact = fact(2 * $i + 1);
007     $pow = pow($angle, (2 * $i + 1)) / $fact;
008     $sin: $sin + pow(-1, $i) * ;
009 }
010 @return $sin;
011 }
```

```
@function cos($angle) {
  $COS: 0;
  $angle: rad($angle);
  // Повторить несколько раз
  @for $i from 0 through 10 {
    $pow = pow($angle, 2 * $i) ;
    $cos: $cos + pow(-1, $i) * $pow / fact (2 * $i) ;
  }
  @return $cos;
}
```

Наконец, чтобы вычислить *тангенс* с помощью функции `tan()`, функции `sin()` и `cos()` обязательны.

```
@function tan($angle) {
  @return sin($angle) / cos($angle);
}
```

Если вам неинтересно писать собственные математические и тригонометрические функции, то можете просто включить библиотеку `compass` (см. следующий пример) и использовать `sin()`, `cos()` и другие тригонометрические функции из списка.

25.14. Анимация генератора

Возьмем все, что мы узнали из этой главы, и создадим анимацию генератора синусоидальных колебаний:

```
@import "compass/css3";

.atom {
    text-align:  center;
    border-radius: 20px;
    height:      40px;
    width:       40px;
    margin:      1px;
    display:     inline-block;
    border:      10px #1893E7 solid;
    /* Применение анимации генератора (определена ниже) */
    animation: oscillate 3s ease-in-out infinite;
    /* Создание 15 классов для каждого из 15 блоков */
    @for $i from 1 through 15 {
```

```
015         &:nth-child(#{$i}) {
016               animation-delay: ( #{sin(.4) * ($i)}s );
017         }
018     }
019 }
020
021 @keyframes oscillate {
022     0% { transform: translateY(0px); }
023     50% { transform: translateY(200px); }
024 }
```

А это HTML-часть, сокращенный вариант примера. Убедитесь, что у вас есть 15 реальных HTML-элементов с классом `class = "atom"`.

```
001 <!-- повторять данный элемент 15 раз //-->
002 <div class = "atom"></div>
```

В результате будет создана следующая анимированная синусоида.

Весь код занял менее 24 строк!

ПРИМЕЧАНИЕ

Если вы используете инфраструктуру Vue, то можете визуализировать все 15 элементов, задействуя этот простой цикл for, с помощью директивы v-for, вместо того чтобы вводить все 15 HTML-элементов вручную.

Настройка для отображения 15 HTML-элементов с помощью всего лишь нескольких строк кода в среде Vue:

```
001 <ul id = "atoms">
002   <li v-for = "atom in atoms">
003     {{ atom.message }}
004   </li>
005 </ul>
```

JavaScript-объект `Vue`, который создает объект `Array` и заполняет его 15 элементами, содержащими значение `0`:

```
001 let atoms = new Vue({
002   el: "#atoms",
003   data: {
004     atoms: new Array(15).fill(0);
005   }
006 });
```

Я включил этот пример сюда только для того, чтобы показать: объединение нескольких фреймворков и библиотек облегчает написание кода и его поддержку в будущем.

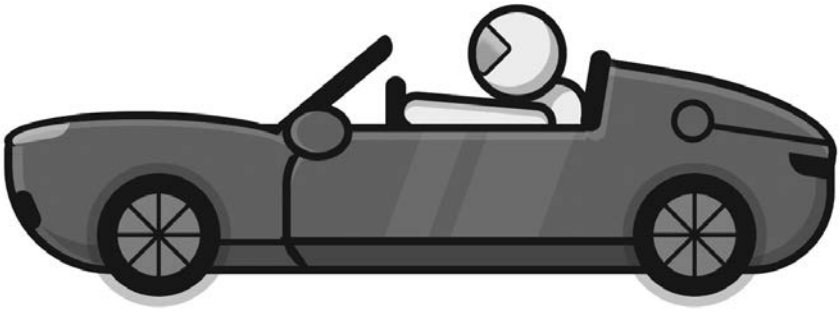
Однако это не значит, что вы должны применять их в каждом отдельном проекте. Иногда проще написать код в обычной форме.

В ходе работы вы столкнетесь с кодом, использующим несколько библиотек.

26 CSS-графика: Tesla

Хотя язык CSS был разработан в первую очередь для оказания помощи в разработке сайтов и макетов веб-приложений, отдельные талантливые разработчики пользовательского интерфейса создают с его помощью невероятные изображения! Некоторые утверждают, что в этом мало практического смысла. Но факт остается фактом... художники создают сложные проекты, используя глубокие знания свойств и значений CSS.

Ниже представлена CSS-модель автомобиля Tesla, разработанная Сашей Тран (`@sa_sha26` в «Твиттере») специально для книги.



```
.front { left: -190px; }
```

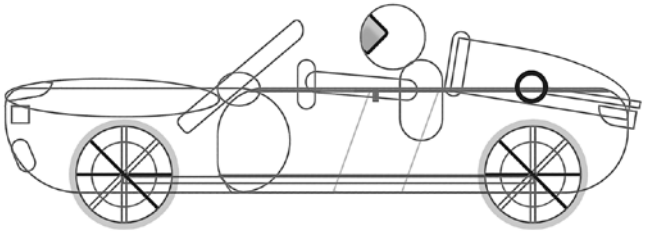
```
.rear { right: -130px; }
```

Далее будет подробно описано, как создавалась каждая отдельная часть автомобиля, какие свойства CSS использовались и т. д.

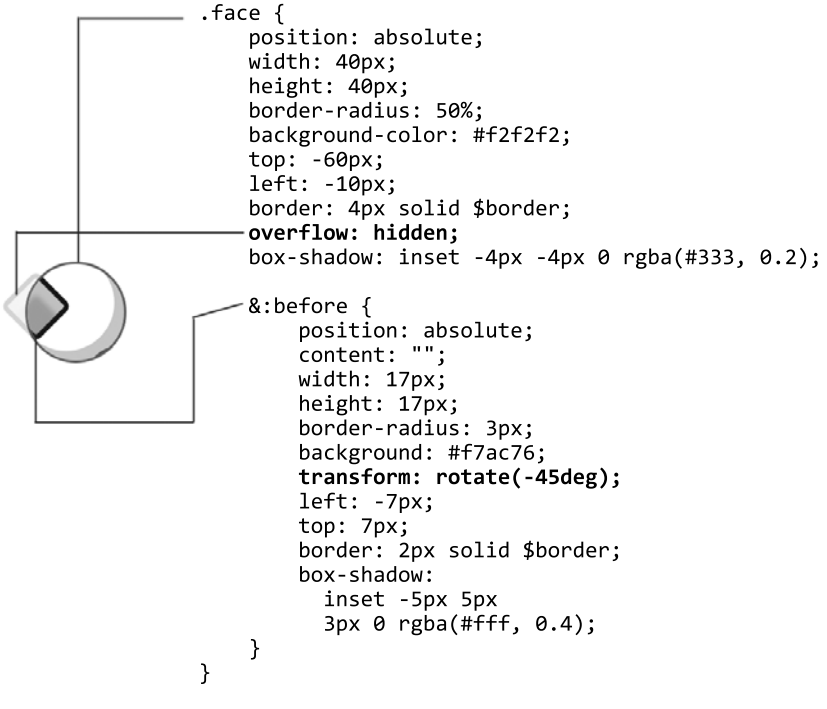
Создание CSS-арта может быть проблемой даже для веб-дизайнеров. Воплотим в жизнь то, что уже узнали из этой книги!

Все зависит от того, насколько творчески вы подходите к свойствам CSS: `hidden`, `transform:rotate`, `box-shadow` и `border-radius`.

Если сделать все фоны прозрачными, то можно ясно видеть композицию Tesla, состоящую из нескольких HTML-элементов `div`:



Далее мы разберем каждый значимый элемент автомобиля, чтобы продемонстрировать, как он был создан.



```
.face {
    position: absolute;
    width: 40px;
    height: 40px;
    border-radius: 50%;
    background-color: #f2f2f2;
    top: -60px;
    left: -10px;
    border: 4px solid $border;
    overflow: hidden;
    box-shadow: inset -4px -4px 0 rgba(#333, 0.2);

    &:before {
        position: absolute;
        content: "";
        width: 17px;
        height: 17px;
        border-radius: 3px;
        background: #f7ac76;
        transform: rotate(-45deg);
        left: -7px;
        top: 7px;
        border: 2px solid $border;
        box-shadow:
          inset -5px 5px
          3px 0 rgba(#fff, 0.4);
    }
}
```

Шлем состоит из круга и оранжевого лицевого щитка, который представляет собой просто вложенный повернутый квадрат с тенью `box-shadow`, отрезаемый по линии радиуса, так как для `.face` установлено значение `overflow: hidden`.

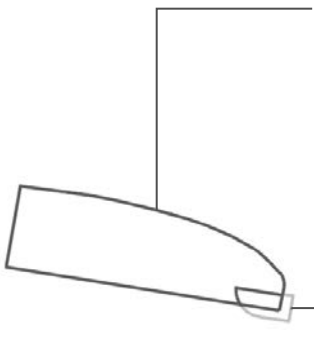
Обратите внимание, как псевдоэлемент `&:before` *вкладывается* внутрь `.face` с помощью `{`*`скобок`*`}`. Это достигается с помощью расширения SASS (Syntactically Awesome Style Sheets). Более подробную информацию можно посмотреть, перейдя по ссылке SASS-lang.com. Кроме того, кратко SASS обсуждается в самом начале книги.

Конечно, вы все еще можете переписать код в стандартном виде CSS, заменив `&:before` и скобки отдельным элементом с собственным *идентификатором* или *классом*.

```
&-bumper-top {
    width: 135px;
    height: 23px;
    position: absolute;
    background-color: $car-body;
    border: 4px solid;
    border-radius: 50%;
    top: -8px;
    left: -235px;
    transform: rotate(1deg);
    border-color: $border transparent transparent $border;
    overflow: hidden;
    z-index: 99;
    box-shadow: inset 0 3px 0 rgba(#fff, 0.17);

    .front-light-bulb {
        position: absolute;
        width: 33px;
        height: 10px;
        background:
            rgba(#fff, 0.5);
        transform:
            rotate(-10deg);
        border-radius: 50px 0;
        left: -4px;
        top: 1px;
    }
}
```

Капот представляет собой длинный овальный элемент, повернутый всего на 1 градус. Как и лицевой щиток шлема, лампочка скрывается внутри родительского элемента с помощью свойства `overflow: hidden`. Сокрытие потока — то, что помогает создавать более сложные неправильные формы, точно описывающие реальные объекты.

Важность свойства `overflow: hidden` в создании стилей CSS невозможно переоценить. Подсветка использует абсолютно ту же технику, что и в предыдущих двух примерах. Задняя часть автомобиля представляет собой повернутый прямоугольник с одним из закругленных углов. Здесь вы просто должны следовать своему внутреннему чутью для создания форм, соответствующих вашим предпочтениям и стилю.



```
&-rear-top {
    position: absolute;
    width: 113px;
    height: 33px;
    background-color: $car-body;
    top: -25px;
    left: 50px;
    border-radius: 0 70% 0 0;
    transform: rotate(9.2deg);
    border: 4px solid;
    border-color: $border transparent transparent;
    z-index: $hand;
    box-shadow: inset 0 4px 0 rgba(#ff0, 0.17);

    .back-light {
        position: absolute;
        width: 23px;
        height: 10px;
        background-color: $border;
        top: 27px;
        left: 94px;
        z-index: 0;
        border-radius: 0px 0 0 50px;
    }
}
```

Основание автомобиля, которое тянется к его задней части, представляет собой большой прямоугольный `div`-элемент с закругленными углами и внутренней тенью `box-shadow`.

```
&-fender {
    position: absolute;
    top: -2px;
    left: -100px;
    width: 260px;
    height: 65px;
    border-radius: 30px 20px 40px 20px;
    background-color: #ce4038;
    border: 4px solid;
    border-color: $border;
    z-index: $car-rear;
    overflow: hidden;
    box-shadow: inset 0 4px 0 rgba(#fff, 0.17),
      inset -5px -4px 0 rgba(#333, 0.2);
```

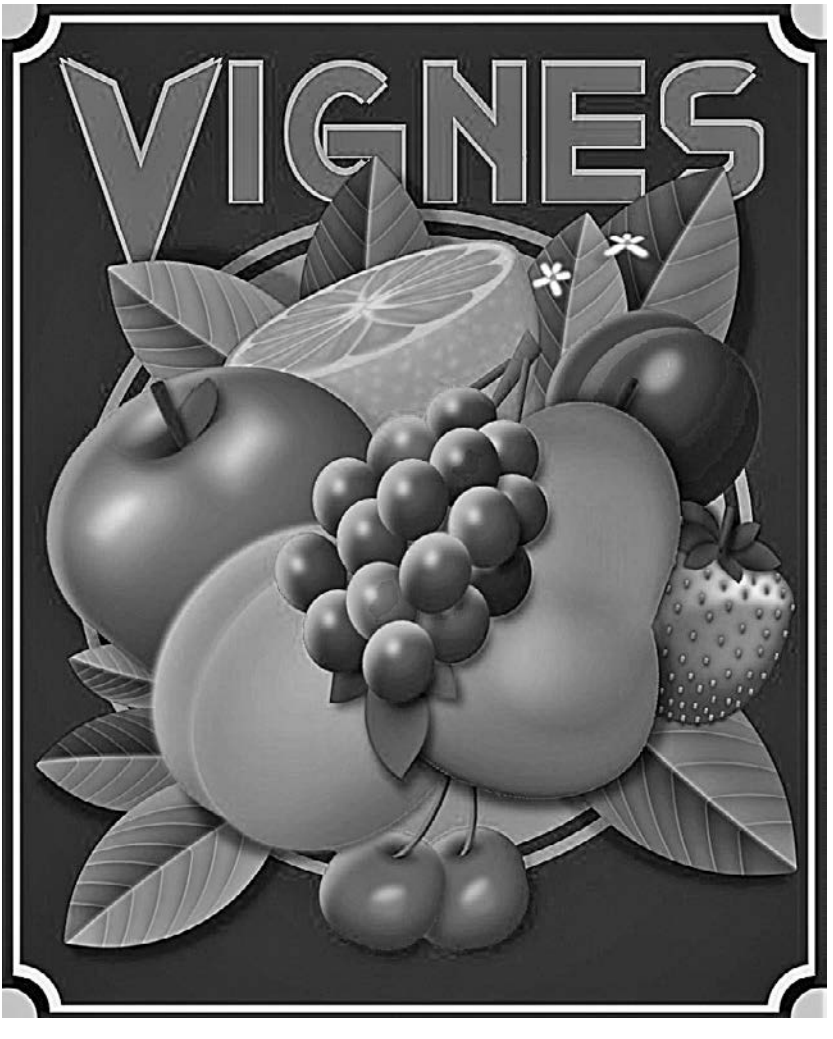
```
    &-tire {
        .front, .rear {
        width: 60px;
        height: 60px;
        background: $border;
        position: absolute;
        border-radius: 50%;
        top: 22px;
        z-index: $tire;
        display: flex;
        justify-content: center;
        align-items: center;

        &:before {
            position: absolute;
            width: 60px;
            height: 60px;
            content: "";
            border: 5px solid #333;
            opacity: 0.2;
            border-radius: 50%;
        }
    }
}
```

Здесь символ `&` означает ключевое слово `this` (концептуально похожее на объект `this` в JavaScript), то есть... элемент ссылается сам на себя. Как уже было показано в одной из глав, псевдоселекторы `:before` (а также `:after`) фактически содержатся в одном и том же HTML-элементе. Их можно применять для создания дополнительных фигур без необходимости добавления элементов.

И вот машина готова! Я говорил только о ключевых свойствах CSS, часто используемых для создания CSS-графики. Чтобы избежать избыточности, некоторые из них, наиболее очевидные, были пропущены. Например, предполагается, что вы уже знаете, как применять свойства `top`, `left`, `width` и `height`.

Для просмотра оригинального кода CSS проекта на сайте codepen.io, пожалуйста, перейдите по ссылке codepen.io/sashatran/pen/gvVWKJ.

Этот графический объект был создан с помощью CSS:



Еще одно изображение, созданное Дианой Смит (Diana A Smith) с помощью CSS. Вы можете просмотреть его в своем браузере, перейдя по ссылке diana-adrianne.com/purecss-zigario/.

Послесловие

Редко бывает так, что книгу целиком пишет один человек. Хотя все рисунки были созданы мной, эта книга не появилась бы без участия других талантливых разработчиков, графических дизайнеров и редакторов. Их имена перечислены ниже. Я благодарен, что у меня есть команда соавторов и волонтеров, которые после нескольких редакций коллективно помогли сделать книгу такой, какая она есть сегодня.

Благодарности

Я хотел бы поблагодарить следующих людей.

Сашу Тран (Sasha Tran), *разработчика пользовательских интерфейсов*, за проектирование автомобиля Tesla с помощью CSS и полный исходный код CSS. Если вам нравится ее творчество, то можете посетить сайт sashatran.com либо ее аккаунт на сайте Codepen.io по адресу codepen.io/sashatran/ или в «Твиттере» @sa_sha26.

Катю Сорок (Katya Sorok), *редактора*, за многочисленные указания, как улучшить текст и иллюстрации. Ее учетная запись в «Твиттере» — @KSorok.

Фабио Ди Корлето (Fabio Di Corleto), *графического дизайнера*, за предоставление оригинальной концепции автомобиля Tesla в пространстве. Если вы ищете талантливого графического дизайнера, то можете связаться с ним по адресу fabiodicorleto@gmail.com или с помощью его профилей в социальных сетях Instagram и Dribbble по имени пользователя fabiodicorleto.

Наконец, я хотел бы выразить огромную благодарность…

Диане А. Смит (Diana A. Smith), *UI-дизайнеру*, за создание невероятных дизайнерских работ с помощью CSS. Вам обязательно стоит

посетить ее сайт diana-adrianne.com. Таких оригинальных дизайнов, созданных с помощью CSS, в Интернете вы больше не найдете.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Грег Сидельников

Наглядный CSS

Перевел с английского *С. Черников*

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*А. Питиримов*
Ведущий редактор	*Н. Гринчик*
Литературный редактор	*Н. Хлебина*
Художественный редактор	*В. Мостипан*
Корректоры	*Е. Павлович, Е. Рафалюк-Бузовская*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 05.2021. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214,
тел./факс: 208 80 01.

Подписано в печать 14.05.21. Формат 70×100/16. Бумага офсетная. Усл. п. л. 18,060. Тираж 500. Заказ 0000.

Отпечатано в полном соответствии с качеством предоставленных материалов в ООО «Фотоэксперт».
109316, г. Москва, Волгоградский проспект, д. 42, корп. 5, эт. 1, пом. I, ком. 6.3-23Н.