



Динамические элементы

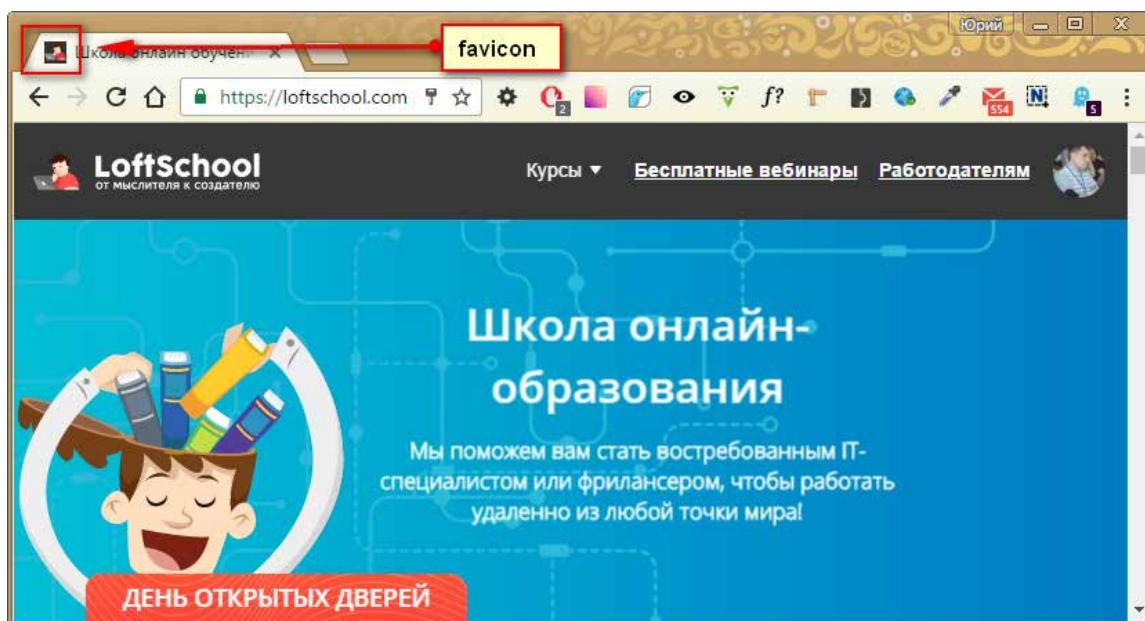
Содержание

1. Создание и установка favicon	3-5
2. CSS-спрайты	6-8
3. Центрирование	9-16
• По горизонтали	10
• По вертикали	11
• И по горизонтали, и по вертикали	15
4. Создание треугольников с помощью стилей	17-21
5. Список использованных источников	22-23
6. Border-radius	24-26
7. Градиент	27-29
8. Анимация	30-35
9. Анимации с помощью библиотеки Animate.css	16-37
11. Вендорные префиксы	38-39

1

Создание и установка favicon

Для каждого ресурса существует маленькое изображение, что-то вроде мини-логотипа для сайта.



Favicon – это файл небольшого размера (**16×16 пикселей**), имеющий расширение **.ico** (это расширение используется операционными системами для хранения иконок). **Favicon.ico** есть практически у каждого сайта, причем неважно, на каком он хостинге расположен или какой движок/тему использует. Онлайн-генераторы и галереи favicon позволяют сделать иконку за пару минут из любого изображения, нарисовать ее с нуля или же скачать готовую.

Есть несколько путей создания **favicon.ico**:

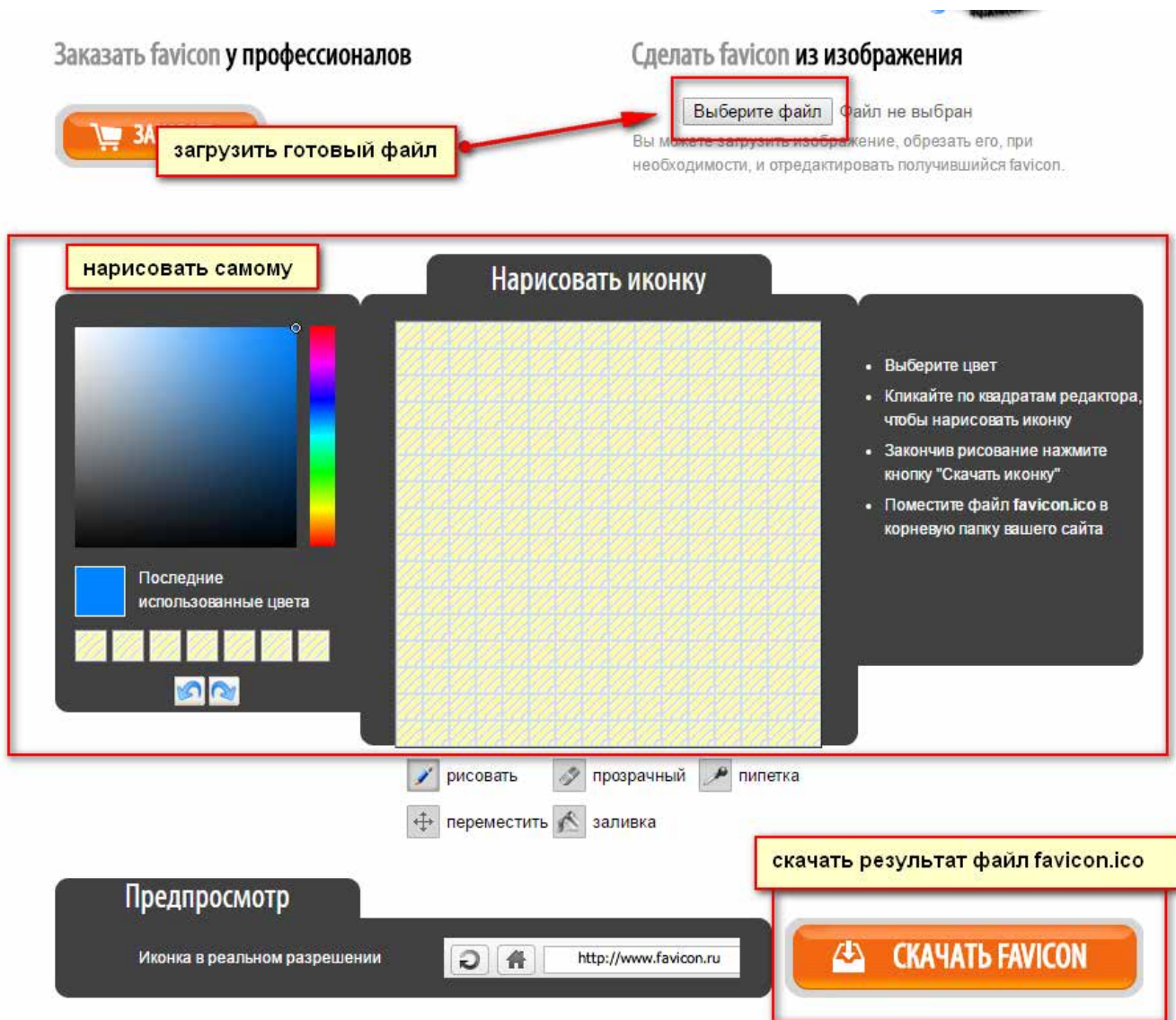
1. Скачать готовую иконку из галерей favicon.
2. Сделать фавикон с нуля с помощью готового файла или онлайн-генератора.

Коллекции и галереи Фавикон:

- <http://www.thefavicongallery.com>
- <http://www.iconj.com>
- <http://www.favicon.cc>
- <http://favicon-generator.org>

Преимущество создания favicon с нуля заключается в том, что получившаяся иконка будет уникальна, а уникальность очень важна! Поэтому лучше не полениться и создать собственный фавикон.

Самый популярный генератор в рунете <http://favicon.ru>



Использовать его можно двумя способами: просто положить в корневую папку сайта или прописать на странице внутри тега **<head>**

```
<link rel="icon" type="image/x-icon" href="http://путь_к_фавикон/favicon.ico">
```

2

CSS-спрайты

CSS-спрайт – способ объединить много изображений в одно, чтобы:

- Сократить количество обращений к серверу.
- Загрузить несколько изображений сразу, включая те, которые понадобятся в будущем.
- Если у изображений сходная палитра, то объединённое изображение будет меньше по размеру, чем совокупность исходных картинок.

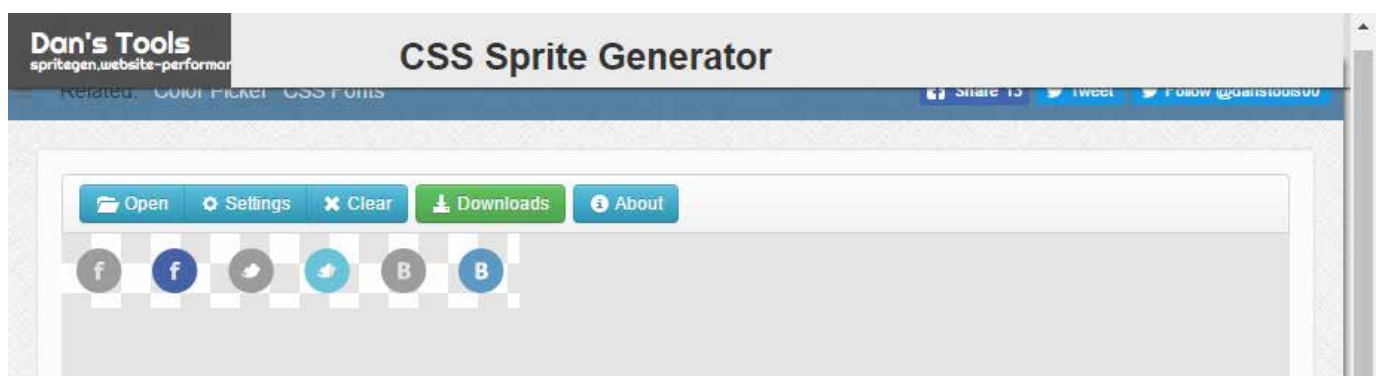
Первый шаг к объединению изображений в «спрайт» – показывать их через **background**, а не через тег **IMG**.

Если фоновое изображение нужно повторять по горизонтали или вертикали, то спрайты тоже подойдут – изображения в них нужно располагать в этом случае так, чтобы при повторении не были видны соседи, т.е., соответственно, вертикально или горизонтально, но не «решёткой».

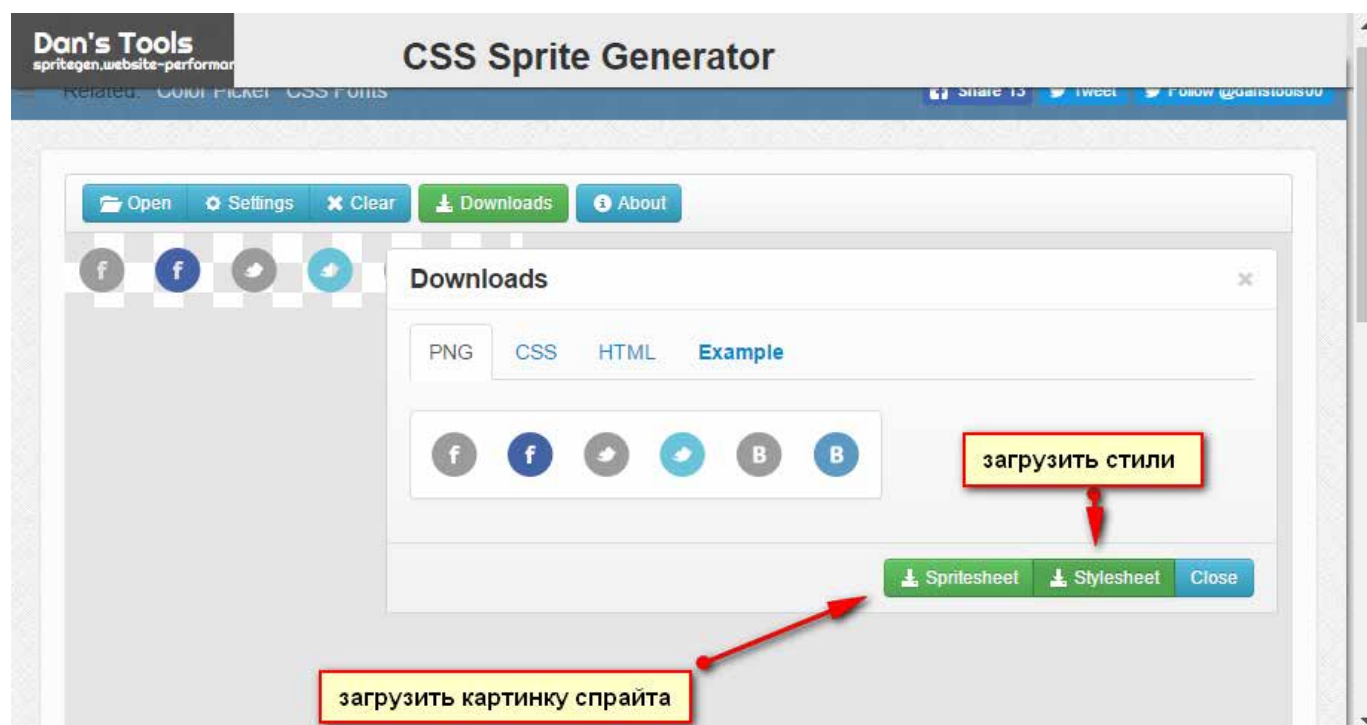
Чтобы иконки при наведении меняли своё изображение. Один спрайт будет содержать все состояния иконок, а переключение внешнего вида – осуществляться при помощи сдвига **background-position**.

Для генерации будем использовать популярный генератор <http://spritegen.website-performance.org/>

Загрузим иконки из нашего макета в генератор



Если теперь нажать кнопку **download** мы сможем скачать сам спрайт в формате **png** и файл стилей **css**.



Там же, на вкладках, для понимания функционирования работы спрайта, находятся готовые примеры кода, которые можно посмотреть, чтобы понять принцип устройства спрайтов.

3

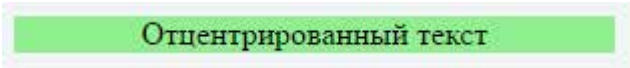
Центрирование

Очень важно знать и понимать основные принципы центрирования в верстке. Итак начнем. Если нужно отцентрировать элемент...

По горизонтали

Строчные элементы внутри родительского блочного можно центрировать так:

```
.inner {  
  width: 300px;  
  background-color: lightgreen;  
  text-align: center;  
}
```



Отцентрированный текст

Это сработает для элементов с типом отображения **inline**, **inline-block**, **inline-table**, **inline-flex** и т.д.

Блочный элемент можно центрировать, указав для **margin-left** и **margin-right** значение **auto** (а также прописав для него конкретный **width**, иначе он займёт всю ширину родительского контейнера и не будет нуждаться в центрировании). Для этого часто используют сокращённую запись:

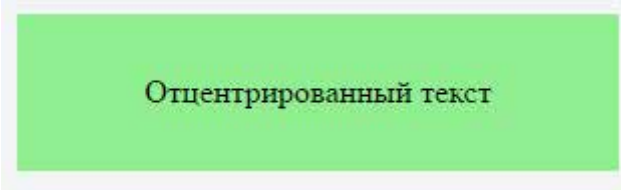
```
.center-me {  
  margin: 0 auto;  
}
```

Способ работает независимо от ширины центрируемого блочного элемента и его родителя.

По вертикали

Иногда строчные/текстовые элементы могут выглядеть отцентрированными по вертикали только потому, что у них одинаковые верхнее и нижнее поля (**padding**).

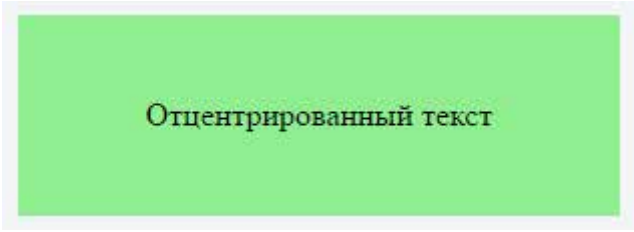
```
.inner {  
  width: 300px;  
  background-color: lightgreen;  
  text-align: center;  
  padding-top: 30px;  
  padding-bottom: 30px;  
}
```



Отцентрированный текст

Если использование полей по какой-либо причине невозможно, а вы хотите отцентрировать текст, который точно не будет переноситься в следующую строку, можно использовать хитрость с установкой **line-height**, равной высоте элемента, это отцентрирует текст:

```
.inner {  
  width: 300px;  
  background-color: lightgreen;  
  text-align: center;  
  height: 100px;  
  line-height: 100px;  
  white-space: nowrap;  
}
```



Отцентрированный текст

Центрирование нескольких строчек текста также можно сделать с помощью установки одинаковых верхнего и нижнего полей, однако, если этот вариант вам не подходит, можно сделать элемент, в который помещён текст, ячейкой таблицы — буквально или же просто её имитацией на CSS.

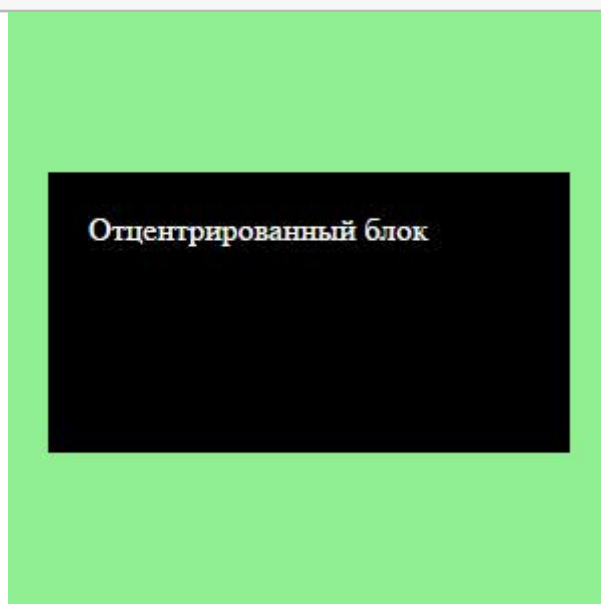
За центрирование в этом случае отвечает свойство **vertical-align**, несмотря на то, что обычно он просто выравнивает по горизонтали элементы в ряду.

Если вам известна высота блочного элемента, вертикальное центрирование можно сделать так:

```
<div class="outer">
  <div class="inner">
    Отцентрированный блок
  </div>
</div>
```

```
.outer {
  background: lightgreen;
  height: 300px;
  margin: 20px;
  width: 300px;
  position: relative;
}
```

```
.inner {  
  position: absolute;  
  top: 50%;  
  left: 20px;  
  right: 20px;  
  height: 100px;  
  margin-top: -70px;  
  background: black;  
  color: white;  
  padding: 20px;  
}
```



Если вам неизвестна высота блочного элемента, то его можно центрировать, подняв вверх на половину его высоты после того, как сдвинули его наполовину вниз:

```
.outer {  
  background: lightgreen;  
  height: 300px;  
  margin: 20px;
```

```
width: 300px;
position: relative;
}

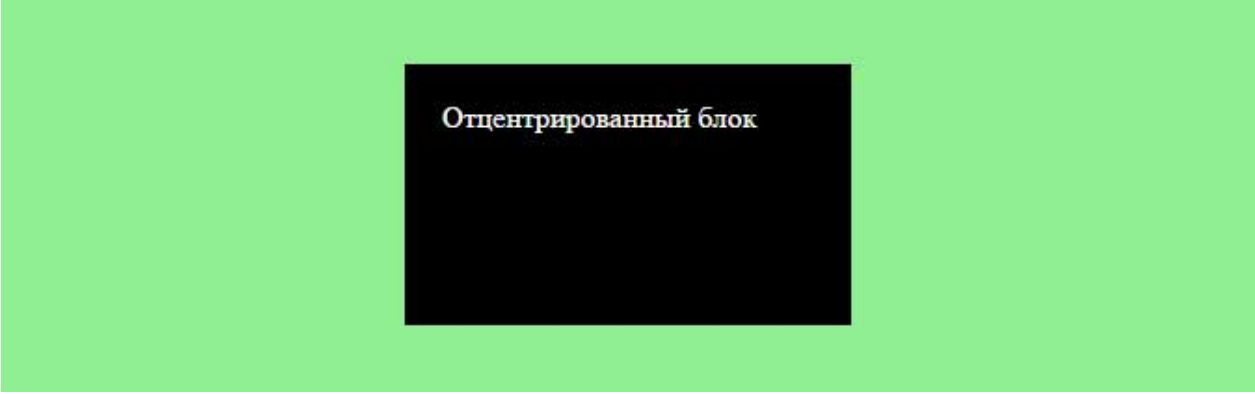
.inner {
  position: absolute;
  top: 50%;
  left: 20px;
  right: 20px;
  background: black;
  color: white;
  padding: 20px;
  transform: translateY(-50%);
}
```



И по горизонтали, и по вертикали

Если у элемента фиксированная высота и ширина, то использование отступов с отрицательными значениями, равными половине высоты и ширины элемента после того, как для него было прописано абсолютное позиционирование в точке **50%/ 50%**, отцентрирует элемент внутри родителя. Кроме того, способ хорошо поддерживается браузерами:

```
.outer {  
  background: lightgreen;  
  height: 200px;  
  width: 60%;  
  margin: 0 auto;  
  padding: 20px;  
  position: relative;  
}  
  
.inner {  
  position: absolute;  
  background: black;  
  color: white;  
  width: 200px;  
  height: 100px;  
  margin: -70px 0 0 -120px;  
  top: 50%;  
  left: 50%;  
  padding: 20px;  
}
```



Отцентрированный блок

Если вам не известны ширина или высота, для центрирования можно использовать свойство **transform** и отрицательное значение **translate** по **50%** в обоих направлениях (оно вычисляется от текущей ширины/высоты элемента):

```
.outer {  
  background: lightgreen;  
  height: 200px;  
  width: 60%;  
  margin: 0 auto;  
  padding: 20px;  
  position: relative;  
}  
  
.inner {  
  position: absolute;  
  background: black;  
  color: white;  
  width: 50%;  
  transform: translate(-50%, -50%);  
  top: 50%;  
  left: 50%;  
  padding: 20px;  
}
```



Отцентрированный блок

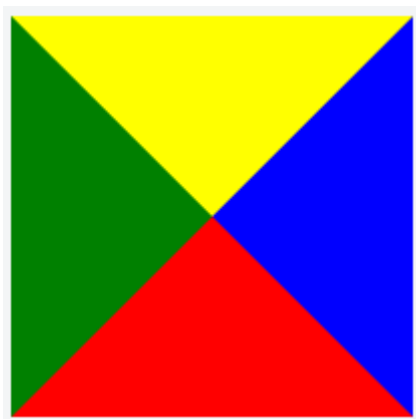
4

Создание треугольников с помощью стилей

На сайтах треугольники применяются сплошь и рядом, как часть дизайна элементов, например, они служат указателем на какой-то объект, направляя внимание читателя в нужное место. Также треугольники выполняют декоративные функции, делая блоки, где они применяются, более изящными и современными.

Хотя границы, создаваемые через свойство **border**, напрямую не имеют отношения к треугольникам, именно border используется для этого наиболее часто. Если задать нулевую ширину и высоту элемента, а также установить достаточно толстую границу, то мы увидим набор из четырёх треугольников. Для наглядности границы на всех сторонах установлены разного цвета.

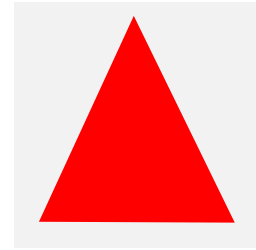
```
<div id="triangle-up"></div>
#triangle-up {
width: 0;
height: 0;
border-left: 100px solid green;
border-right: 100px solid blue;
border-top: 100px solid yellow;
border-bottom: 100px solid red;
}
```



Оставляя только нужную границу, а остальные делая прозрачными, мы получим треугольник нужного цвета.

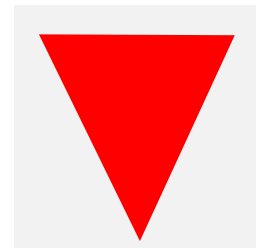
Треугольник вверх

```
#triangle-up {  
  width: 0;  
  height: 0;  
  border-left: 50px solid transparent;  
  border-right: 50px solid transparent;  
  border-bottom: 100px solid red;  
}
```



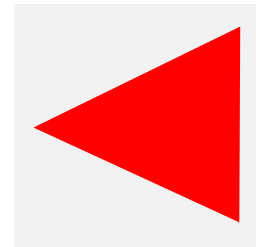
Треугольник вниз

```
#triangle-down {  
  width: 0;  
  height: 0;  
  border-left: 50px solid transparent;  
  border-right: 50px solid transparent;  
  border-top: 100px solid red;  
}
```



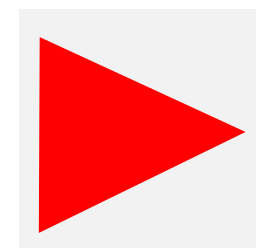
Треугольник влево

```
#triangle-left {  
  width: 0;  
  height: 0;  
  border-top: 50px solid transparent;  
  border-right: 100px solid red;  
  border-bottom: 50px solid transparent;  
}
```



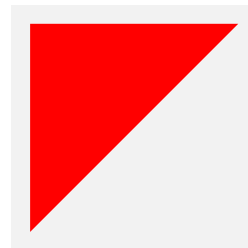
Треугольник вправо

```
#triangle-right {  
  width: 0;  
  height: 0;  
  border-top: 50px solid transparent;  
  border-left: 100px solid red;  
  border-bottom: 50px solid transparent;  
}
```



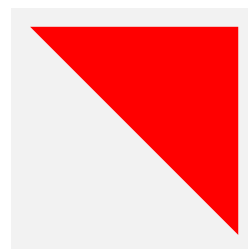
Левый верхний уголок

```
#triangle-topleft {  
  width: 0;  
  height: 0;  
  border-top: 100px solid red;  
  border-right: 100px solid transparent;  
}
```



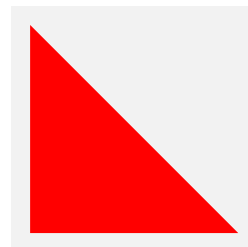
Правый верхний уголок

```
#triangle-topright {  
  width: 0;  
  height: 0;  
  border-top: 100px solid red;  
  border-left: 100px solid transparent;  
}
```



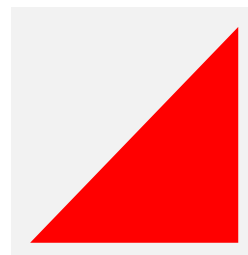
Левый нижний уголок

```
#triangle-bottomleft {  
  width: 0;  
  height: 0;  
  border-bottom: 100px solid red;  
  border-right: 100px solid transparent;  
}
```



Правый нижний уголок

```
#triangle-bottomright {  
  width: 0;  
  height: 0;  
  border-bottom: 100px solid red;  
  border-left: 100px solid transparent;  
}
```

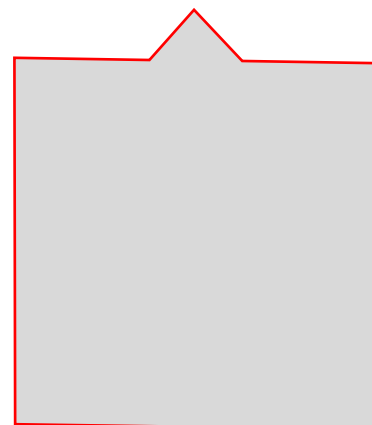


С помощью **border** мы получаем сплошные цветные треугольники, для создания рамки приходится идти на хитрость и накладывать один элемент поверх другого с небольшим смещением. Опять же здесь нам помогут псевдоэлементы **:before** и **:after**

```
<div class="triangle"></div>
.triangle {
  position: relative;
  width: 300px;
  height: 200px;
  background-color: #ccc;
  margin: 20px;
  border: 2px solid red;
}

.triangle:after {
  position: absolute;
  content: "";
  top: -20px;
  left: 45%;
  width: 0;
  height: 0;
  border-left: 20px solid transparent;
  border-right: 20px solid transparent;
  border-bottom: 20px solid #ccc;
}

.triangle:before {
  position: absolute;
  content: "";
  top: -22px;
  left: 45%;
  width: 0;
  height: 0;
  border-left: 20px solid transparent;
  border-right: 20px solid transparent;
  border-bottom: 20px solid red;
}
```



5

Список использованных источников

- 🔗 [Центрирование в CSS: полное руководство](#)
- 🔗 [Проблемы CSS. Часть 1](#)
- 🔗 [Проблемы CSS. Часть 2](#)
- 🔗 [Чек-лист вёрстки](#)
- 🔗 [Фигуры на CSS](#)

6

Border-radius

Свойство устанавливает радиус скругления уголков рамки. Если рамка не задана, то скругление также происходит и с фоном.

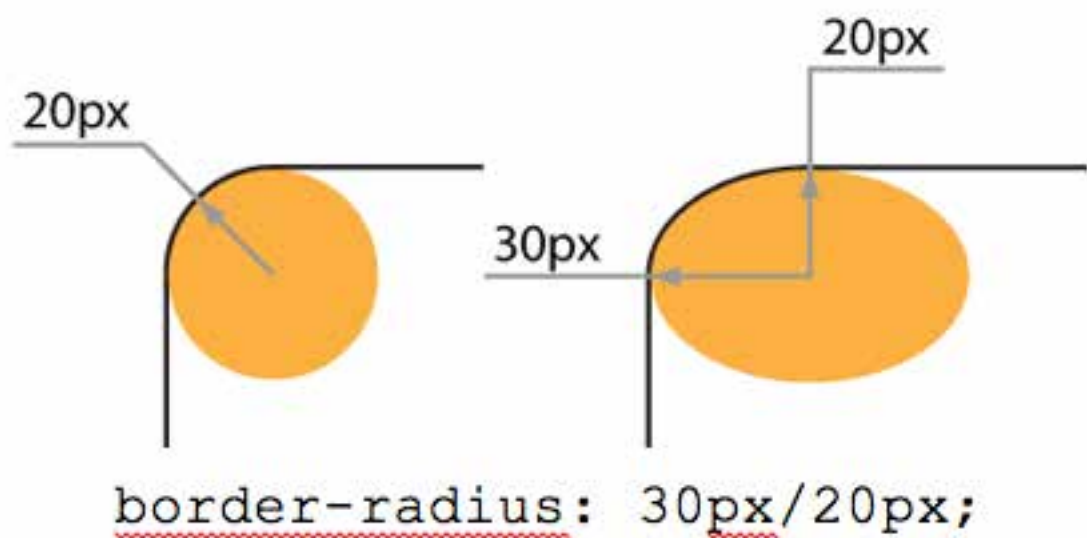
Синтаксис

```
border-radius: <радиус> [ / <радиус>]
```

Разрешается использовать одно, два, три или четыре значения, перечисляя их через пробел. Также допустимо писать два значения через слэш (/). В качестве значений указываются числа в любом допустимом для CSS формате. В случае применения процентов, отсчет ведется относительно ширины блока.

Число значений	Результат
1	Радиус указывается для всех четырех уголков.
2	Первое значение задает радиус верхнего левого и нижнего правого угла, второе значение — верхнего правого и нижнего левого угла.
3	Первое значение задает радиус для верхнего левого угла, второе — одновременно для верхнего правого и нижнего левого, а третье — для нижнего правого угла.
4	По очереди устанавливает радиус для верхнего левого, верхнего правого, нижнего правого и нижнего левого угла.

В случае задания двух параметров через слэш, то первый задает радиус по горизонтали, а второй по вертикали (эллиптические уголки).



```
border-radius: 30px/20px;
```

7

Градиент

Радиальный градиент создается с помощью функции **radial-gradient()**. По умолчанию форма градиента – круг, центр градиента совпадает с центром блока.

Градиенту можно придать форму эллипса, изменяя пару значений, идущих после формы градиента, заданных в **%** или **px**, например **background: radial-gradient(55% 65%, blue, green, yellow)**. Первое значение задает размер по оси X, второе – по оси Y.

Более подробно про градиенты можно почитать в этих статьях:

1. [Линейный градиент](#)
2. [CSS-градиенты: линейные градиенты](#)
3. [CSS3 градиент](#)

Но почти все веб-дазайнеры, при создании градиентов, используют специальные сервисы. Мы отобрали для вас два самых популярных и советуем использовать их для ускорения процессов разработки и для лучшей поддержки кроссбраузерности:

1. [Ultimate CSS Gradient Generator](#)
2. [The ultimate CSS tools for web designers](#)

8

Анимация

С помощью CSS можно создавать сложные анимации и очень гибко управлять ими. Описание CSS-анимации состоит из двух частей: набора ключевых кадров `keyframes` и параметров самой анимации.

Вот пример описания ключевых кадров анимации:

```
@keyframes stretching {  
  0% {  
    width: 100px;  
  }  
  100% {  
    width: 200px;  
  }  
}
```

Анимация в примере имеет название `stretching`, и в ней описывается, как будет меняться стиль блока от начальной до конечной точки. Эту анимацию можно применить к любому элементу, для этого достаточно добавить в CSS два свойства — **animation-name** (название анимации) и **animation-duration** (длительность) — и задать им нужные значения. Например:

```
.button {  
  animation-name: stretching;  
  animation-duration: 1s;  
}
```

Этот код назначит анимацию `stretching` элементам с классом `button`. В результате работы анимации элемент плавно увеличит ширину со 100px до 200px за 1 секунду.

Для каждой анимации нужно задать имя, описать начальный и конечный ключевые кадры, которые задаются с помощью зарезервированных слов **from** и **to** или значений 0% и 100%.

Также можно описать промежуточные ключевые кадры, которые задаются с помощью процентов.

Если не задан начальный ключевой кадр, то анимация будет проигрываться из исходного стилевого состояния элемента к ближайшему шагу из перечисленных в `keyframes` и далее.

Если не задан конечный кадр, то после достижения последнего промежуточного шага, анимация проиграется в обратном направлении до достижения изначального состояния элемента.

Ключевые кадры внутри **keyframes** могут быть написаны в произвольном порядке, но лучше их перечислять по хронологии от меньшего к большему.

Длительность анимации **animation-duration** задаётся в секундах или миллисекундах, например: **10s, 100ms**.

Одному элементу могут быть одновременно назначены несколько анимаций.

Если в этих анимациях меняются разные свойства элемента, то они будут проигрываться одновременно.

Теперь разберём, как добавить элементу вторую параллельную анимацию. Допустим, у нас есть две анимации:

```
@keyframes move {  
  to { left: 100px; }  
}  
@keyframes stretch {  
  to { width: 100px; }  
}
```


Чтобы назначить элементу вторую анимацию, нужно добавить её название и длительность через запятую в свойствах **animation-name** и **animation-duration**. Вот так:

```
.element {  
  animation-name: move, stretch;  
  animation-duration: 5s, 5s;  
}
```

В этом примере две анимации запустятся одновременно, элемент будет параллельно двигаться и удлиняться в течение 5-ти секунд.

Множественные анимации задаются так же, как и множественные фоны и тени — с помощью перечисления свойств через запятую.

Мы можем определять сколько раз будет повторяться анимация. Для этого используется свойство **animation-iteration-count**.

В качестве значения оно принимает положительные числа и ноль: при нуле анимация не будет выполнена, в остальных случаях она повторится указанное число раз.

Также в качестве значения **animation-iteration-count** может быть использовано служебное слово **infinite**. Оно означает, что анимация будет выполняться бесконечно и никогда не завершится.

Помимо количества проигрываний анимации, мы можем определить её направление с помощью свойства **animation-direction**. По умолчанию анимация имеет прямое направление **normal**.

Но можно назначить и обратный порядок анимации, чтобы проигрывание начиналось с конца и шло к началу (то есть за начальную точку считался

кадр **to**, а за конечную — **from**). Для этого используется значение **reverse** свойства **animation-direction**.

Кроме длительности анимации, мы можем управлять задержкой перед началом её выполнения. Синтаксис свойства **animation-delay**, с помощью которого и назначается задержка начала, идентичен синтаксису свойства **animation-duration**. Например, при задании значения **animation-delay: 10s** анимация начнётся не сразу, а только через десять секунд.

Свойство, которое определяет, будет ли видимым эффект от анимации, когда сама анимация уже закончилась — это **animation-fill-mode**. При задании свойству значения **forwards** элемент будет сохранять состояние после завершения анимации.

Другое значение свойства **animation-fill-mode** — **backwards**. Это значение определяет состояние элемента до начала анимации.

Если элементу назначена анимация с задержкой начала проигрывания и **animation-fill-mode: backwards**, то стили, описанные в первом ключевом кадре **from** или **0%**, будут применены сразу, ещё до начала проигрывания анимации.

Третье значение свойства **animation-fill-mode** — **both**. Оно объединяет действия **forwards** и **backwards**. То есть до начала анимации элементу присваивается состояние первого ключевого кадра, а после завершения — конечное состояние анимации сохраняется.

Действие **animation-fill-mode: both** распространяется и на многократную, и на чередующуюся анимацию.

Ещё одно управляющее свойство CSS-анимаций — **animation-play-state**. С его помощью можно поставить анимацию «на паузу», а потом возобновить

с места остановки. Свойство принимает два значения **running** и **paused**. Как видно из названий, **paused** приостанавливает анимацию, а **running** начинает или возобновляет анимацию, поставленную на паузу. Значение **running** задано по умолчанию.

И, наконец, самое интересное свойство — **animation-timing-function**. Оно определяет, как именно будет происходить анимация,: с какой скоростью и ускорением будут меняться свойства, задействованные в ней.

ease, linear, ease-in, ease-out и ease-in-out.

Как вы уже догадались существует универсальное свойство которое задаёт сразу несколько параметров анимации.

```
animation: animation-name || animation-duration ||  
animation-timing-function || animation-delay ||  
animation-iteration-count || animation-direction ||  
animation-fill-mode || animation-play-state
```

9

Анимации с помощью библиотеки Animate.css

Анимации в CSS, удобнее, быстрее и изящнее, чем аналогичные эффекты в JavaScript. Строительство большой анимации не просто, хотя бы потому, что подразумевает много терпения, проб, ошибок, тестирования и махинаций со стилевыми префиксами.

[Animate.css](#) делает кодирование чуть более сносным. Просто выберите эффект, посмотрите его в действии, скачайте код и добавтье пару классов к вашему HTML-элементу. Вы можете скачать весь CSS-файл.

Animate.css – это библиотека собравшая в себе более 60 красивых CSS-анимаций, она кроссбраузерна, и крайне удобна в использовании. Просто подключите ее к вашей странице, и добавьте пару классов к необходимому элементу.

Подключаем библиотеку к нашей странице, с помощью тега **link**

```
<link rel="stylesheet" href="animate.min.css">
```

Теперь мы можем использовать **Animate.css**, для этого нужно добавить 2 класса **"animated"** и класс необходимого эффекта.

```
<h3 class="animated shake">Пример 1, обычная анимация при загрузке страницы</h3>
```

Сюда же можно добавить 3 класс **«infinite»**, если вы хотите что бы анимация не заканчивалась

```
<h3 class="animated shake infinite">Пример 2, бесконечная анимация</h3>
```

10

Вендорные префиксы

Это приставки (префиксы), используемые производителями (вендорами) браузеров для экспериментальных, еще не принятых в стандарт, CSS-свойств.

Вендорные префиксы самых распространенных браузеров приведены в таблице ниже:

Вендорный префикс	Производитель	Браузер	Браузерный движок
-o-, -op-, -xv-	Opera Software	Opera	Presto
-moz-	проект Mozilla	Firefox, SeaMonkey, Camino и др.	Gecko
-ms-	Microsoft	Internet Explorer 8	Trident
-khtml-	проект KDE	Safari до версии 3, Konqueror и др.	KHTML послужил основой для WebKit
-webkit-	Apple	Safari 3+, Google Chrome и др.	WebKit

Благодаря вендорным префиксам производители браузеров уже внедряют экспериментальные CSS3 свойства на свой страх и риск.

Верстальщик уже сейчас может реализовать большинство возможностей предоставляемых CSS3, в том числе разнообразные переходы и анимации без использования скриптов, но используя вендорные префиксы.

Наглядным примером такой реализации может быть использование CSS3 свойства transition.

Для это используйте следующий сервис:

[Автопрефиксер онлайн \(autoprefixer css online\)](#)