**Анимации и трансформации. Введение**

Сайт должен быть удобным и практичным. Но не стоит забывать про эстетичность и красоту, которые создают приятные впечатления для пользователей. Например, приятнее прочитать книгу на сайте, где интерфейс приближен к реальной книге, чем на чёрно-белом статичном фоне. Поэтому веб-разработчики используют декоративные эффекты. В этой теме мы расскажем про анимации, трансформации, градиенты и тени.

Сначала мы познакомим вас с трансформациями — изменениями формы элементов. Современные интерфейсы полны объектов, которые изменяются при взаимодействии с ними. Так пользователь понимает: «Элемент интерактивный и с ним можно что-то сделать». Вы умеете применять эффекты по наведению мыши. Но что, если нужно сделать эффект плавным, управлять ускорением и замедлением, поворачивать, менять форму и пропорции объектов? Мы покажем, как это сделать.

Затем расскажем о градиентах и тенях. Они помогают разнообразить цветовые дизайнерские решения.

И напоследок раскроем тонкости самого необычного декоративного эффекта — анимации. Иногда дизайнерская задумка выходит за пределы возможностей трансформации. Например, нужно сделать поочерёдное появление элементов на странице, плавное затухание или увеличение прозрачности, постоянное движение элемента. С такими задачами справляется анимация. В JavaScript есть более мощные инструменты для анимаций, но вы убедитесь, что можно сотворить многое и на CSS.

В этой теме не будет единого проекта на все уроки. Но наши дизайнеры придумали для вас несколько примеров, на которых можно попрактиковаться и получить удовольствие от результата.

Принимая от дизайнера макет, веб-разработчик всегда уточняет, как ведут себя элементы в разных состояниях: это избавляет от ошибок, связанных с пониманием, какие из элементов интерактивные. После этой темы вы станете осознанно подходить к таким вещам.

**Трансформации**

Трансформации — это способ изменения формы элементов. В CSS за трансформации отвечает свойство transform (англ. transform, «превратить, преобразовать»).

В общем виде любую трансформацию можно описать значением matrix c параметрами — матрицей трансформаций, или матрицей преобразований.

Вот как она выглядит:

Скопировать кодCSS

div {

transform: matrix(2, 3, 4, 5, 6, 7);

}

Вот что означает каждый из шести параметров:

2 — изменение масштаба по горизонтали (элемент увеличен вдвое по оси X);

3 — скос по вертикали (положительное значение поднимает левую половину элемента и опускает правую, отрицательное наоборот);

4 — скос по горизонтали (положительное значение наклоняет элемент влево, отрицательное вправо);

5 — изменение масштаба по вертикали (элемент растянут в 5 раз по оси Y);

6 — сдвиг по оси X в пикселях;

7 — сдвиг по оси Y в пикселях.

Попробуйте сами:

Матрицей можно описать практически любое изменение формы объекта, но для самых популярных есть специальные обозначения.

Скопировать кодCSS

transform: translate(10px, 20px); */\* сдвинет объект на 10 пикселей вправо и на 20 пикселей вниз \*/*

transform: scale(1.5, 0.5); */\* увеличит объект в полтора раза по горизонтали и уменьшит наполовину по вертикали \*/*

transform: scale(1.5); */\* увеличит объект в полтора раза и по горизонтали и по вертикали \*/*

transform: rotate(30deg); */\* повернёт объект на 30 градусов по часовой стрелке \*/*

transform: skew(30deg, 40deg); */\* наклонит объект на 30 градусов влево и скосит левую сторону на 40 градусов вверх, а правую на 40 градусов вниз \*/*

Этих четырёх специальных значений хватает для большинства повседневных задач, а понимание устройства матрицы преобразований нужно, когда вы делаете анимацию на JavaScript.

Обратите внимание на единицы измерения.

Сдвиги translate заданы в пикселях, но возможно определять их и в относительных величинах.

Масштабирование scale определяется относительно 1. Значение 1 означает «один к одному», то есть элемент не изменится. Числа больше единицы увеличивают объект, а меньше единицы — уменьшают.

Повороты rotate и скосы skew можно определять в градусах deg (от англ. degree, обычный градус, каких в окружности 360). Допустимы отрицательные значения, чтобы реализовать повороты против часовой стрелки и скосы вправо.

Помимо двумерных трансформаций, существуют трёхмерные. Они создают эффект псевдо-3D. Здесь мы их разбирать не будем — ограничимся самыми необходимыми.

# Плавность переходов

Вы заметили, как резко меняются элементы страницы при трансформациях? До наведения мыши элемент выглядит так, а по наведении — сразу совершенно иначе. Такое превращение приемлемо далеко не всегда.

Можно изменять стиль элемента плавнее. Эффекты перехода устанавливает универсальное свойство transition (англ. transition, «переход»). У него есть особые значения:

* transition-property (англ. property, «свойство») — сообщает, какие именно свойства должны меняться плавно;
* transition-duration (англ. duration, «продолжительность») — время перехода в секундах;
* transition-timing-function (англ. timing function, «функция отсчёта времени») — описывает, как будет вести себя анимация (она может ускоряться, замедляться или двигаться линейно);
* transition-delay (англ. delay, «отсрочка») — задержка в секундах между изменением состояния элемента и запуском анимации.

Все эти значения можно задавать по отдельности:

Скопировать кодCSS

div {

background-color: black;

transition-property: background-color;

transition-duration: 1s;

transition-timing-function: linear;

transition-delay: 2s;

}

div:hover {

background-color: white;

}

Или сокращённой записью:

Скопировать кодCSS

div {

background-color: black;

transition: background-color 1s linear 2s;

}

div:hover {

background-color: white;

}

Такой код сделает переход от чёрного фона к белому плавным: трансформация продлится одну секунду, а начнётся через две секунды после наведения мыши на блок.

Значения timing-function можно задать ключевыми словами или в формате cubic-bezier.

Вот самые популярные варианты, которые можно задать ключевыми словами:

* ease (непринуждённо)
* linear (линейно)
* ease-in (осторожно)
* ease-out (отпустить)
* ease-in-out (сначала осторожно, потом отпустить)

Если нужно что-то особенное, переходное поведение можно определить кривой Безье — инструментами, подобными [http://cubic-bezier.com](http://cubic-bezier.com/)

Вы сможете спроектировать ускорения и замедления анимации направляющими и скопировать код animation-timing-function в такой форме:

Скопировать кодCSS

transition-timing-function: cubic-bezier(1,.2,.52,.46);

Этот код задаёт медленный процесс перехода в начале и нелинейное ускорение в конце.

У свойства transition есть особенность — если оно задано элементу изначально, независимо от состояния, то плавность будет работать при любом изменении состояния.

Например, такой код укажет, что плавно нужно изменять фон по такому-то закону при наведении мыши и после того, как мы убрали курсор с элемента:

Скопировать кодCSS

div {

background-color: black;

transition: background-color 1s linear 0.5s;

}

div:hover {

background-color: white;

}

А такой код будет менять фон плавно только при наведении мыши на элемент, потому что свойство transition определено лишь на случай, когда работает :hover:

Скопировать кодCSS

div {

background-color: black;

}

div:hover {

background-color: white;

transition: background-color 1s linear 0.5s;

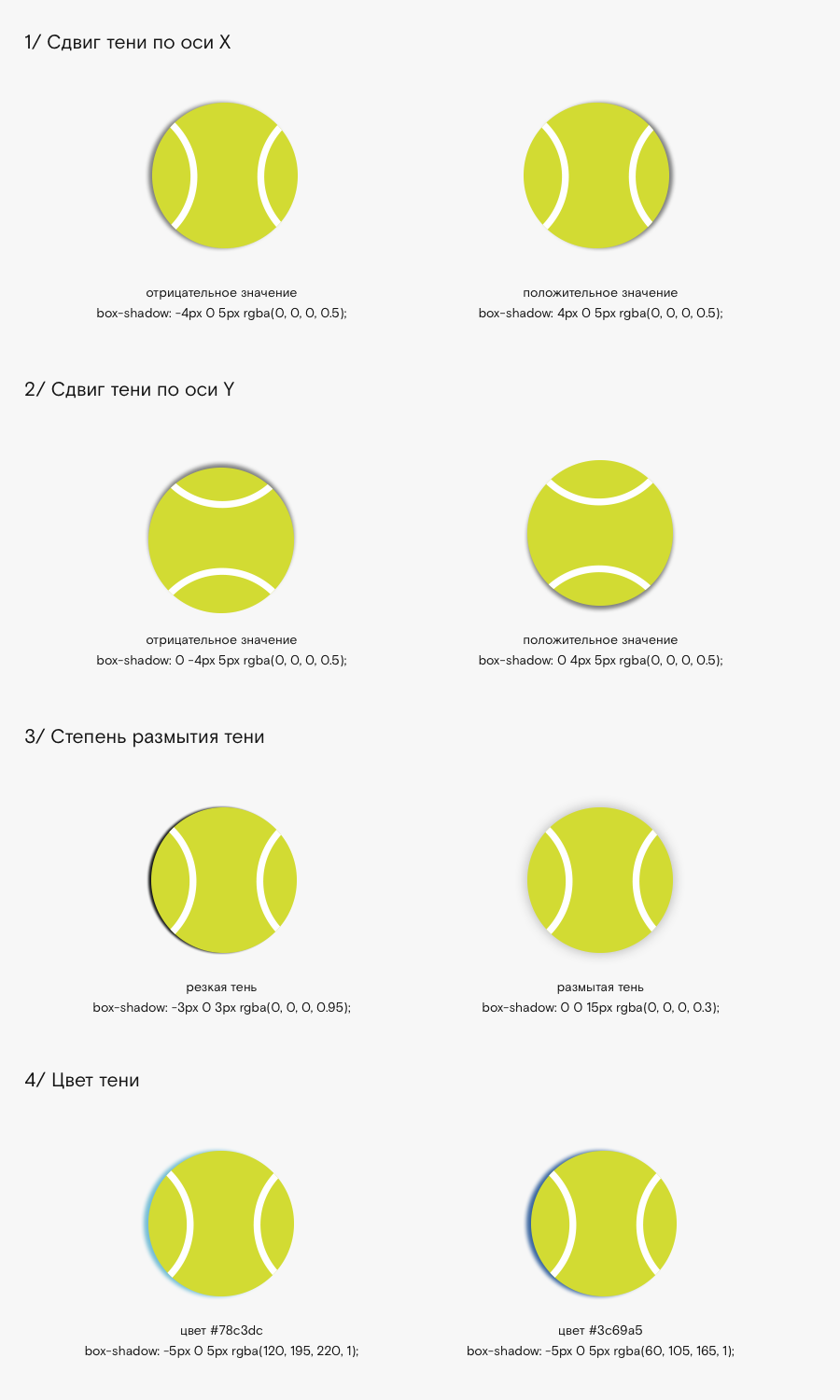
}

**Тени**

Тени — полезный инструмент дизайна. Их можно применять к блочным элементам и текстам. За тень блока отвечает свойство box-shadow (от англ. box shadow, «тень блока»), а за тень текста — text-shadow (от англ. text shadow, «тень текста»).

Последовательность значений, которые описывают тень, для текстов и блоков одинакова:

* горизонтальный сдвиг тени (отрицательное значение создаёт тень на таком-то расстоянии слева, положительное справа);
* вертикальный сдвиг тени (отрицательное значение создаёт тень на таком-то расстоянии сверху, положительное снизу);
* радиус размытия тени (чем это значение больше, тем тень шире и светлей);
* цвет тени (задаётся так же, как цвет текста или фона).



В коде эти значения записаны так:

Скопировать кодCSS

div {

box-shadow: 5px 6px 7px red;

}

h1 {

text-shadow: 5px 6px 7px red;

}

В обоих примерах тень сдвинута вправо на 5 пикселей, вниз на 6 пикселей, радиус её размытия — 7 пикселей, а цвет — красный.

Часто тень не нужно сдвигать, достаточно только размытия. В этом случае первые два значения приравнивают к нулю.

# Линейный градиент

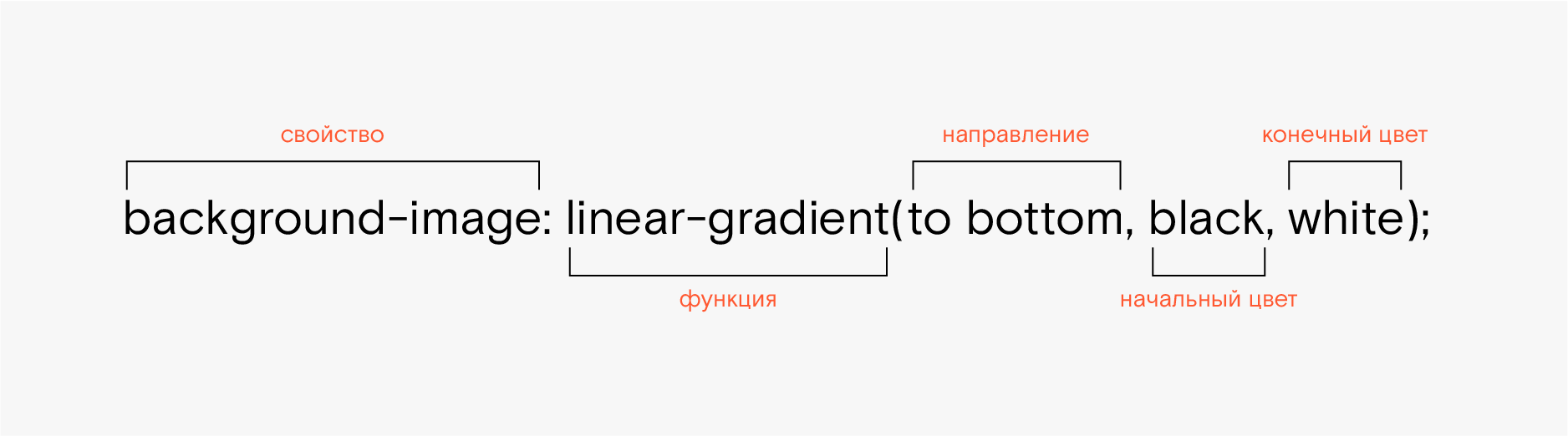
Градиент — это вид заливки блока цветом, когда контрольные цвета фиксируют в определённых точках, между которыми браузер создаёт плавные переходы.

В CSS любой градиент описывают значением свойства background-image. Фактически отрисовка градиента сравнима с отрисовкой изображения.

Есть два вида градиентов — линейный (англ. linear gradient) и радиальный (англ. radial gradient).

### Линейный градиент

Определяется в коде через значение linear-gradient() для background-image.



В самом простом случае указывают в скобках через запятую два цвета — начальный и конечный. Браузер сделает между ними плавный переход.

Скопировать кодCSS

div {

background-image: linear-gradient(#0078FF, #C2E3E3);

}

По умолчанию браузер отрисовывает линейный градиент сверху вниз. Это поведение можно изменить. Один из вариантов — задать поворот градиента в градусах.

Например такой код повернёт градиент на 90 градусов.:

Скопировать кодCSS

div {

background-image: linear-gradient(90deg, #0078FF, #C2E3E3);

}

Для самых используемых направлений существуют специальные параметры, которыми можно заменять указатель в градусах.

Для построения более сложных градиентов можно использовать более одного цвета:

Скопировать кодCSS

div {

background-image: linear-gradient(#0B2337, #126DDC, #76C2E0, #D1DC9D, #F09174);

}

После указания каждого цвета можно задать интервал, в котором нужно применить цвет. Первое значение — начальная точка указанного направления, второе — финальная точка.

Скопировать кодCSS

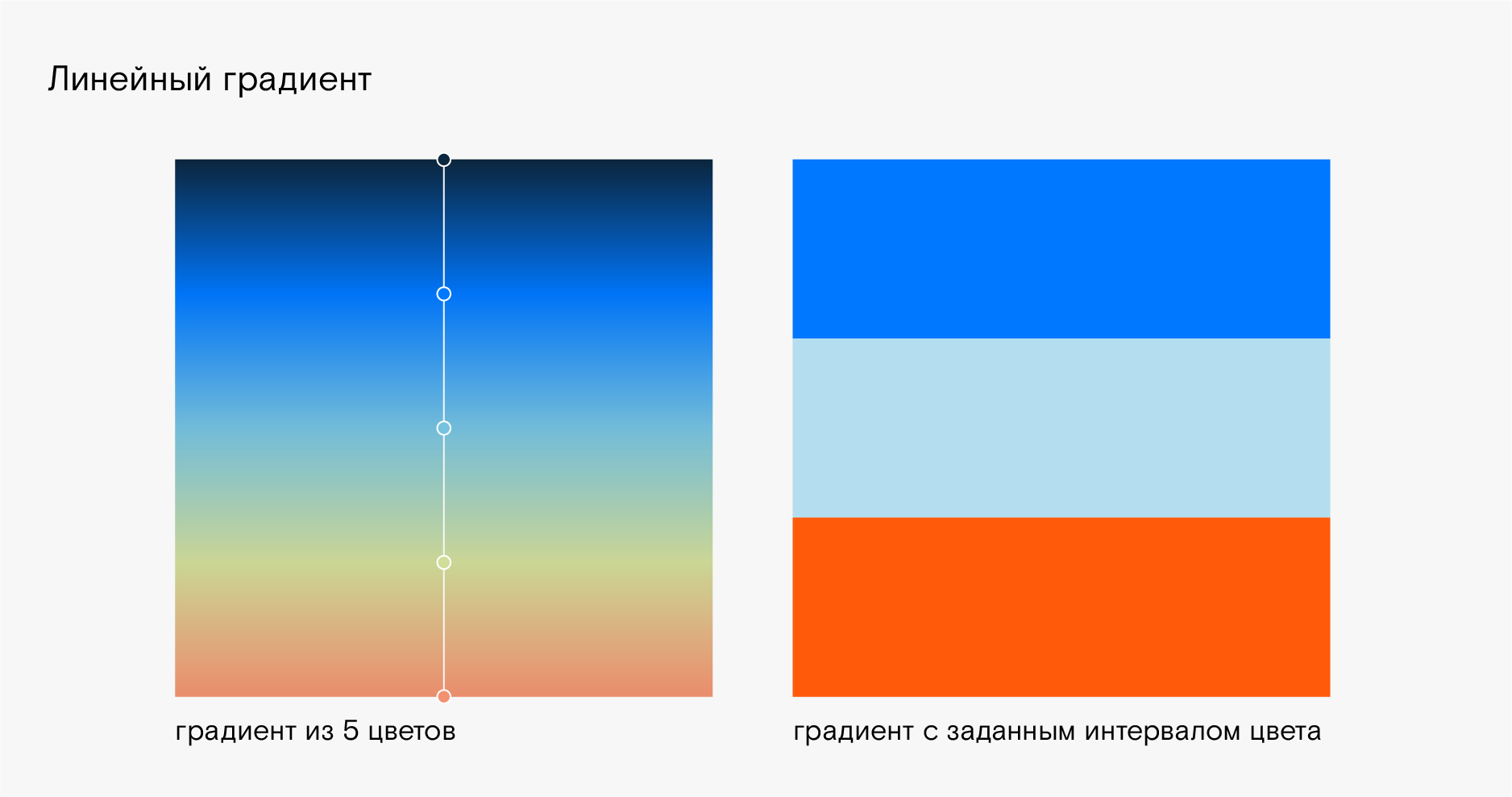
div {

width: 300px;

height: 300px;

background-image: linear-gradient(#0078FF 0px 100px, #B4DEEF 100px 200px, #FF5A0A 200px 300px);

}



Можно также определять точки перелома в процентах:

Скопировать кодCSS

div:first-of-type {

width: 300px;

height: 300px;

background-image: linear-gradient(#0078FF 20%, #C2E3E3 100%);

}

div:last-of-type {

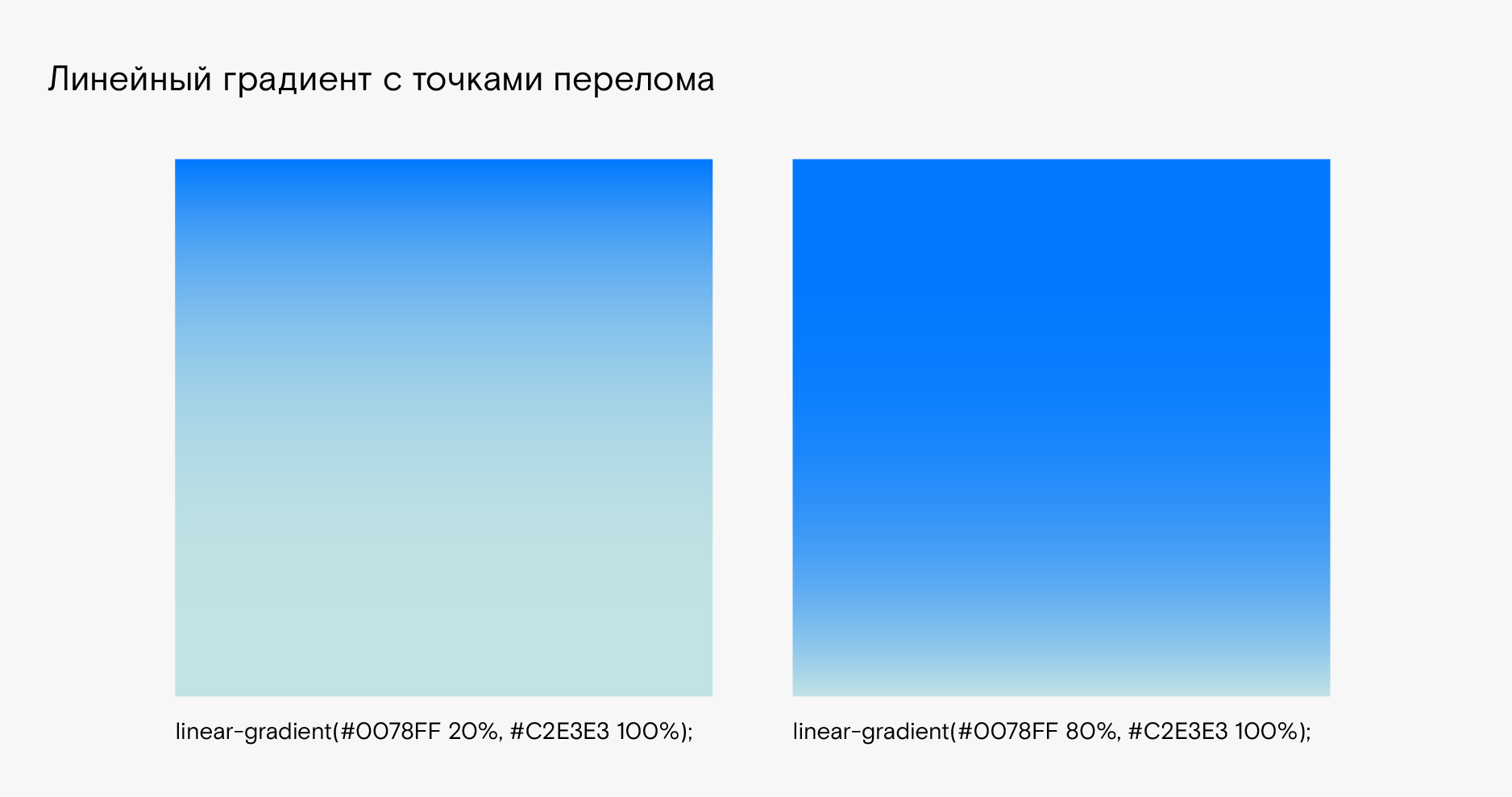
width: 300px;

height: 300px;

background-image: linear-gradient(#0078FF 80%, #C2E3E3 100%);

}

Результат:



# Радиальный градиент

Если цвета линейных градиентов меняются вдоль линии, задающей направление, то при радиальном градиенте изменения цвета расходятся от заданного центра, а сам градиент может принимать форму круга или эллипса.

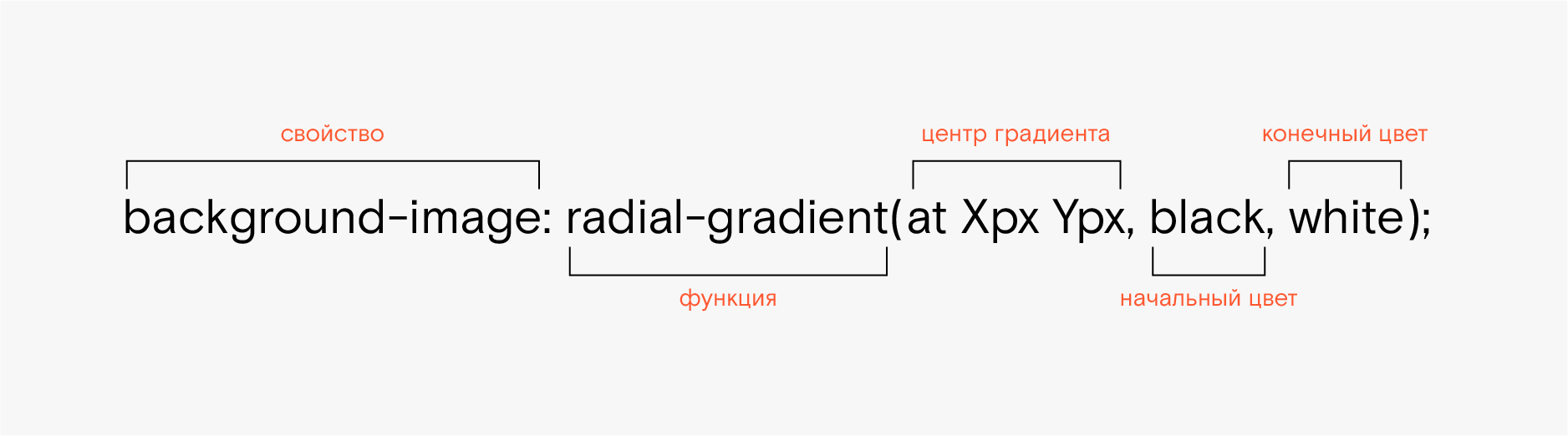
Самый простой радиальный градиент задаётся значением radial-gradient() так:

Скопировать кодCSS

div {

background-image: radial-gradient(#0078FF, #C2E3E3);

}



По умолчанию центр градиента находится в середине блока, но его можно сместить, задав позицию ключевым словом at (здесь в значении «в такой-то точке»).

Скопировать кодCSS

div {

width: 300px;

height: 300px;

background-image: radial-gradient(at 40px 50px, #0078FF, #C2E3E3);

}

Такой код поместит центр градиента на 40 пикселей правее левого края блока и на 50 пикселей ниже верхнего края.

Как с линейным градиентом, можно определять несколько цветов и точки перелома в процентах:

Скопировать кодCSS

div {

width: 300px;

height: 300px;

background-image: radial-gradient(#002918 10%, #B4DEEF 20%, #FFFFFF 50%, #FF5831 100%);

}

или в пикселях

Скопировать кодCSS

div {

width: 300px;

height: 300px;

background-image: radial-gradient(#002918 50px, #B4DEEF 70px, #FFFFFF 100px, #FF5831 200px);

}

Определение точных интервалов тоже доступно:

Скопировать кодCSS

div {

width: 300px;

height: 300px;

background-image: radial-gradient(#002918 0px 20px, #B4DEEF 20px 40px, #FFFFFF 40px 60px, #FF5831 60px 80px);

}



Если вы задаёте ширину и высоту первыми параметрами в скобках, то эллипс растянется или сожмётся:

Скопировать кодCSS

div {

width: 300px;

height: 300px;

background-image: radial-gradient(40px 100px, #0078FF, #C2E3E3);

}

или

Скопировать кодCSS

div {

width: 300px;

height: 300px;

background-image: radial-gradient(100px 40px, #0078FF, #C2E3E3);

}

В документации ещё множество особенностей радиальных градиентов. Это лишь основные настройки, которых будет достаточно на первом этапе их изучения.

**Переполнение блоков**

Что происходит, если попробовать поместить блок размерами 500x500 в блок размером 300x600?

Вложим элемент div с классом inner (англ. «внутренний») в родительский div с классом outer (англ. «внешний»):

Скопировать кодCSS

<div class="outer">

<div class="inner"></div>

</div>

А теперь зададим в CSS блокам с классом inner ширину больше, чем у внешних блоков outer; установим тем и другим разную окраску:

Скопировать кодCSS

.outer {

width: 300px;

height: 600px;

background-color: rgb(1, 163, 114);

}

.inner {

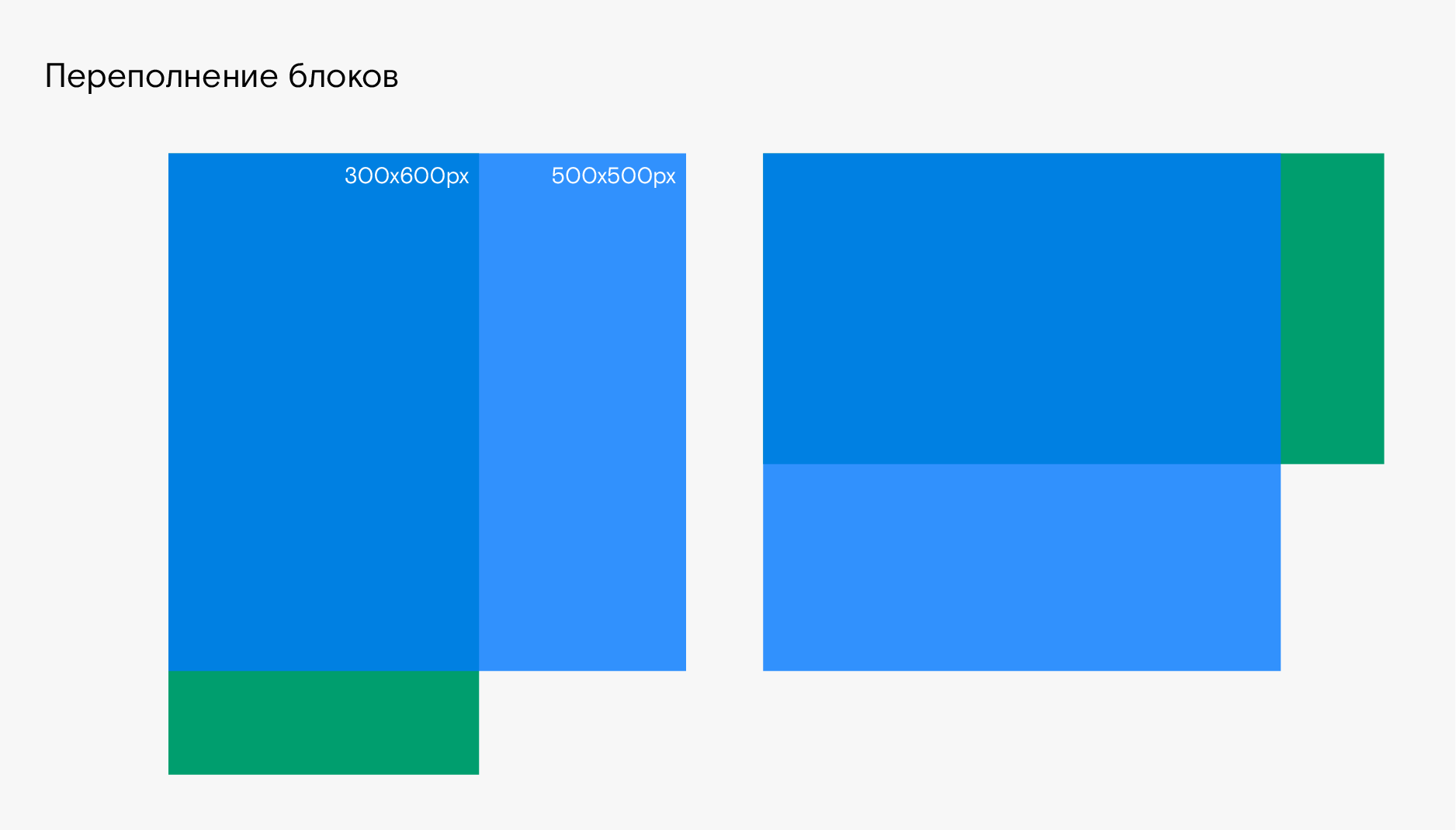
width: 500px;

height: 500px;

background-color: rgba(0, 120, 255, .8);

}

Блоки не вмещаются друг в друга:



Больший блок вышел за рамки меньшего, и это будет видно. Такая ситуация называется в CSS переполнением, а поведение родительского блока при переполнении можно определить свойством overflow (англ. overflow, «переполнение»). У него есть четыре возможных значения:

* visible — значение по умолчанию,
* auto — скрывать зоны, которые выходят за границы блока, но показывать скролл (полосу прокрутки),
* scroll — всегда показывать скролл, если его не прячет браузер
* hidden — скрыть не поместившуюся часть целиком без скролла.

Для примера добавим overflow со значением hidden:

Скопировать кодCSS

.outer {

width: 300px;

height: 600px;

background-color: rgb(1, 163, 114);

overflow: hidden;

}

.inner {

width: 500px;

height: 500px;

background-color: rgba(0, 120, 255, .8);

}

Такой код скроет не уместившееся содержимое:

У свойства overflow есть подтипы с теми же значениями:

* overflow-x — определяет поведение по горизонтали,
* overflow-y — определяет поведение по вертикали.

У этих свойств есть особенность: добавив одному из свойств значение, отличное от visible, оно перестает работать у другого.

# Кейфреймы

В предыдущих уроках мы говорили о простых трансформациях. Теперь перейдём к более сложным. И понадобятся нам для этого ключевые кадры, или кейфреймы.

Сначала разберёмся, как работает анимация в целом. К примеру, фильм — это набор неподвижных кадров. Они меняются так быстро, что мы видим, будто изображение движется. Анимация устроена примерно также. Браузер отрисовывает разные состояния элемента одно за другим с такой скоростью, что мы видим плавный переход. Каждая такая отрисовка называется кадром.

Во втором уроке этой темы мы рассматривали такой пример:

Скопировать кодCSS

div {

background-color: black;

transition: background-color 2s linear 0s;

}

Этот код заставляет div плавно менять свой цвет. В такой анимации уже есть два ключевых кадра: первый и последний. В первом фон прозрачный, в последнем — чёрный. И состояние элемента плавно меняется от одного к другому.

Ещё раз: анимация состоит из кадров. Ключевые кадры — это те, в которых мы задаём состояние элемента. Остальные кадры автоматически рассчитываются браузером так, чтобы выполнить плавный переход от одного кейфрейма к другому.

Из ключевых кадров анимация создаётся так:

1. Описываем состояние элемента в каждом ключевом кадре.
2. Селектором выбираем элемент, который нужно анимировать.
3. Задаём уникальные особенности анимации этого элемента. Например, сколько раз повторяется анимация, её хронометраж, её функция времени — далее мы рассмотрим большинство возможностей.

### Этап 1. Описываем преобразования

Кейфреймы объявляются директивой @keyframes — все директивы пишут после знака @.

Объявление (декларация) должно обязательно содержать имя. По нему мы будем вызывать анимацию. Имя следует прямо за директивой через пробел:

@keyframes name\_of\_keyframe (здесь name\_of\_keyframe — от английского выражения «имя кейфрейма», в напоминание, что имена лучше всего давать английские и базовой латиницей).

После декларации открывают фигурные скобки «тела кейфреймов».

Скопировать кодCSS

@keyframes name\_of\_keyframe {

<здесь будут правила изменения стилей>

}

В фигурных скобках описаны CSS-свойства элемента в каждом ключевом кадре. Кейфремы при этом указывают в процентах. 0% соответствует началу анимации, 100% — концу, 50% — середине и так далее.

Скопировать кодCSS

@keyframes move {

0% {

left: 0;

top: 0;

}

25% {

left: 30px;

top: 30px;

}

50% {

left: 60px;

top: 0;

}

75% {

left: 90px;

top: 30px;

}

100% {

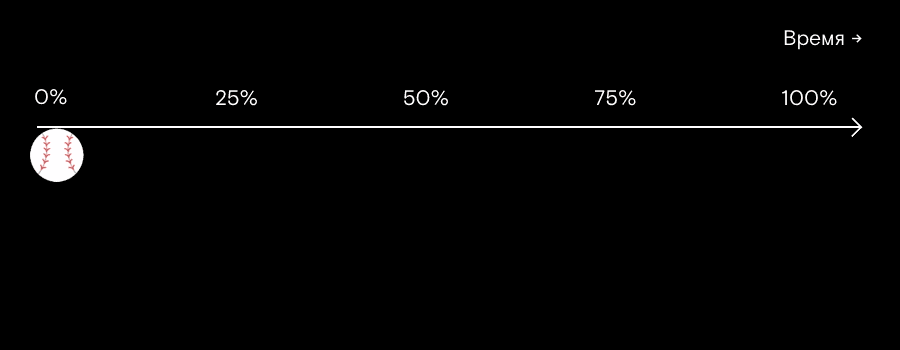
left: 120px;

top: 0;

}

}

Такой код определяет анимацию из пяти ключевых кадров. Каждый кейфрейм задаёт положение объекта. Между этими положениями элемент и будет плавно перемещаться:



Если состояний всего два — начальное и конечное, возможна упрощенная запись. Так, в этом кейфрейме rotation (англ. rotation, «вращение») описан полный оборот от 0° до 360°, заданный двумя значениями rotate свойства transform:

Скопировать кодCSS

@keyframes rotation {

from {

transform: rotate(0deg);

}

to {

transform: rotate(360deg);

}

}

## Важно:

Описав кейфреймы, мы лишь придумали характер анимации. Но описания недостаточно — анимацию ещё нужно применить к элементу. О том, как это делается, поговорим в следующих уроках.

# Анимации

За применение анимации к HTML-элементу отвечает группа свойств animation.

Чтобы анимация заработала, нужно обязательно указать два подсвойства:

* animation-name — имя набора кейфреймов;
* animation-duration — хронометраж анимации; время, за которое анимация проигрывается целиком.

Например:

Скопировать кодCSS

div {

animation-name: move;

animation-duration: 2s;

}

Такой код запустит анимацию из кейфреймов с именем move и завершит её через две секунды.

Можно также использовать сокращенную запись:

Скопировать кодCSS

div {

animation: move 2s;

}

В группе свойств animation есть много подсвойств, которые производят тонкую настройку анимации:

* animation-timing-function — функция времени. Аналогичное мы рассматривали для свойства transition. Можно задать в виде куба Безье или ключевыми словами (linear, ease-in, ease-in-out, ease, ease-out).
* animation-delay — задержка перед стартом анимации, в секундах.
* animation-iteration-count — сколько раз запустить анимацию (англ. iteration-count, «счёт итерациям»). Задаётся целым числом, но существует специальное ключевое слово infinite, чтобы воспроизводить анимацию бесконечно.
* animation-direction — направление анимации. Значений четыре, задаются ключевыми словами. Можно запустить от начала к концу — значение normal, с конца до начала — reverse, от начала к концу и обратно — alternate, от конца к началу и обратно — alternate-reverse.
* animation-fill-mode — указывает, как элемент ведёт себя после завершения анимации. Значение forwards сохранит элемент в том состоянии, в котором он оказался после выполнения анимации; backwards — вернёт состояние до начала анимации; both (то и другое) — самый сложный вариант. Он сначала установит элементу состояние первого кадра, которое сохраняется до старта анимации — столько времени, сколько задано в animation-delay. По окончании анимации устанавливается состояние последнего кадра.

### Как работает анимация в виджете

До запуска анимации у блока с мячом установлен фон — футбольный мяч. Первый кадр анимации меняет фон на бейсбольный мяч, а последний — на баскетбольный. У анимации установлена задержка.

**Режим forwards.** Сначала запускается задержка анимации, потом мяч переключится на первый кадр (бейсбольный мяч). А после выполнения мяч останавливается на финальном кадре.

**Режим backwards.** Сначала первый кадр, потом задержка, потом анимация до финального кадра. Но финальный кадр не сохраняется — мяч возвращается в исходную позицию и снова становится футбольным.

**Режим both**. Анимация ведёт себя похоже на режим forwards. Сначала произойдет переключение к первому кадру (бейсбольному мячу), потом запустится задержка. А по завершении финальный кадр сохранится.

* animation-play-state — указывает, запущена анимация или нет. Значение running запускает анимацию, paused — останавливает. Этим свойством часто управляют через JavaScript, запуская и останавливая анимацию при различных событиях.

Спецификация CSS описывает дополнительные значения для большинства этих свойств:

* initial — исходное состояние элемента на момент загрузки страницы;
* inherit — унаследовать состояние элемента от родителя.

Это бывает полезно в специфических случаях.

Вот как выглядит полный набор свойств в полной и краткой записях.

Скопировать кодCSS

div {

animation-name: move;

animation-duration: 2s;

animation-timing-function: ease-in-out;

animation-delay: 1s;

animation-iteration-count: 3;

animation-direction: reverse;

animation-fill-mode: forwards;

animation-play-state: running;

}

div {

animation: move 2s ease-in-out 1s 3 reverse forwards running;

}

В сокращённой записи порядок свойств неважен. Только задержку анимации нужно расположить позже длительности, поскольку они задаются в одних единицах измерения.

Элементу можно присвоить множество анимаций, перечислив их через запятую.

Скопировать кодCSS

div {

animation: move 2s, rotation 5s;

}