**Привет, animation!**

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, а затем вернётся в исходное состояние.

**@keyframes: раскадровка**

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.

**@keyframes: from и to**

Как уже говорилось в [предыдущем задании](https://htmlacademy.ru/courses/80/run/2), начальный и конечный ключевые кадры задаются с помощью слов from и to или значений 0% и 100%.

А промежуточные ключевые кадры задаются с помощью процентов. Вот пример анимации из 4 кадров:

@keyframes coloring {

from { background-color: red; }

33% { background-color: yellow; }

66% { background-color: green; }

to { background-color: blue; }

}

# @keyframes: группировка кадров

Ключевые кадры в keyframes можно группировать, для этого нужно перечислить их через запятую. Рассмотрим пример:

@keyframes stretching {

0%,

50% {

width: 100px;

}

100% {

width: 200px;

}

}

В этом примере первые два кадра сгруппированы. Анимируемый элемент сначала изменит свою ширину до 100px и останется в этом состоянии половину времени анимации. А за вторую половину времени он растянется от 100px до 200px.

# Множественная анимация, шаг 1

Одному элементу могут быть одновременно назначены несколько анимаций.

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

# Множественная анимация, шаг 2

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;

}

# Количество проигрываний анимации: animation-iteration-count

Во всех предыдущих примерах мы создавали анимации, которые проигрывались один раз, а потом элемент возвращался в исходное состояние. 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.

# Направление анимации: animation-direction, шаг 1

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.

# Направление анимации: animation-direction, шаг 2

У свойства animation-direction есть ещё два значения. Они используются, когда количество проигрываний анимации animation-iteration-count больше одного. И оба они определяют чередующееся направление анимации.

Если задано значение alternate, то нечётные проигрывания будут выполняться в прямом направлении, а чётные — в обратном.

.element {

animation-name: move;

animation-duration: 1s;

animation-iteration-count: 2;

animation-direction: alternate;

}

В примере анимация move выполнится два раза: в первый (нечётный) раз направление будет прямым, а во второй (чётный) — обратным.

Если задано значение alternate-reverse, то нечётные проигрывания наоборот будут выполняться в обратном направлении, а чётные — в прямом.

# Задержка начала анимации: animation-delay, шаг 1

Кроме длительности анимации, мы можем управлять задержкой перед началом её выполнения.

# Задержка начала анимации: animation-delay, шаг 3

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.

# Состояние до и после анимации: animation-fill-mode, шаг 1

В предыдущих примерах элементы после проигрывания анимации возвращались в исходное состояние. Но есть 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.

Сохранение свойств анимации по её завершению animation-fill-mode: forwards работает и в случае нескольких повторов анимации или чередующегося направления

# Состояние до и после анимации: animation-fill-mode, шаг 3

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.

Задание начального состояния анимации до начала её выполнения animation-fill-mode: backwards работает и в случае нескольких повторов или чередующихся направлений анимации.

# Состояние до и после анимации: animation-fill-mode, шаг 5

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 многоразовую, и на чередующуюся анимацию.

# Остановка и запуск анимации: animation-play-state

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

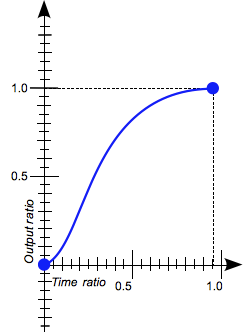
Свойство принимает два значения running и paused. Как видно из названий, paused приостанавливает анимацию, а running начинает или возобновляет анимацию, поставленную на паузу. Значение running задано по умолчанию.

**«Форма» анимации, animation-timing-function**

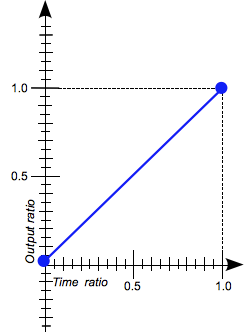
И, наконец, самое интересное свойство — animation-timing-function. Оно определяет, как именно будет происходить анимация: с какой скоростью и ускорением будут меняться свойства, задействованные в ней.

В предыдущих примерах анимация проигрывалась с одинаковой динамикой, мы меняли лишь её длительность, но не «форму». Эта «форма» по умолчанию соответствует первому графику, из которого видно, что анимация начинается медленно, затем ускоряется и к концу движения опять замедляется.

Так ведёт себя значение ease свойства animation-timing-function.

* 

*ease*

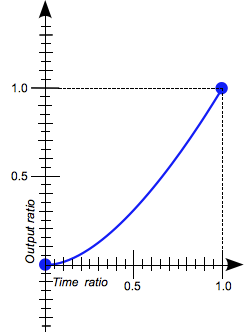
* 

*linear*

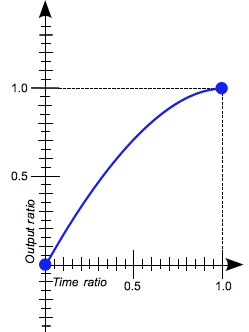
Но мы можем сделать проигрывание анимации равномерным, без ускорений и замедлений. Для этого нужно использовать значение linear. Как видно на втором графике, анимация будет проигрываться с неизменной скоростью.

**animation-timing-function, шаг 2**

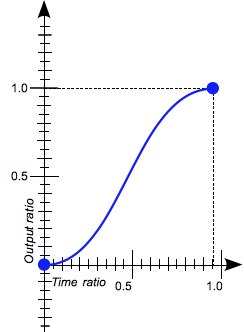
Вот ещё несколько форм анимации: ease-in, ease-out и ease-in-out.

* 

*ease-in*

* 

*ease-out*

* 

*ease-in-out*

Из графиков видно, что при значении ease-in анимация медленно начинается, а к концу ускоряется; при ease-out — начинается быстро, а к концу замедляется. Значение ease-in-out похоже на ease, то есть анимация начинается и заканчивается медленно, но происходит это чуть-чуть интенсивнее.

# animation-timing-function, шаг 3

Что же скрывается за названиями linear, ease и других функций? Довольно сложная математика кубических [кривых Безье](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%91%D0%B5%D0%B7%D1%8C%D0%B5).

По сути, именованные функции ease-in, ease-out и другие являются псевдонимами для универсального описания кривых, например:

cubic-bezier(0, 0, 1, 1) // это linear

cubic-bezier(0.42, 0, 1, 1) // это ease-in

В общем представлении cubic-bezier(x1, y1, x2, y2) значения x и y — это координаты точек кривых на графике. При этом верным считается значение x только в диапазоне от 0 до 1.

Существует [отличный сервис](http://cubic-bezier.com/), помогающий разобраться в функциональном представлении кривых Безье без необходимости штудировать учебники по математике.

А вот по этой [ссылке](https://easings.net/ru) можно найти целую коллекцию разных easing-функций на основе кривых Безье.

C помощью кривых Безье мы можем задавать любые формы анимации.

# animation-timing-function, шаг 4

Давайте разберёмся с ещё одним возможным классом значений animation-timing-function — это steps.

Они позволяют задать «ступеньки», по которым будет идти анимация. Синтаксис steps следующий:

animation-timing-function: steps(число\_шагов, направление);

Тут всё просто: число шагов — это целое число, за которое будет выполнена вся анимация; направление может принимать значение start или end.

При заданном start первый шаг выполняется одновременно с началом анимации, а в случае c end последний шаг будет выполнен вместе с завершением анимации. То есть при start пошаговая анимация идёт как бы с опережением, а при end — вдогонку.

# animation-timing-function, шаг 4

Давайте разберёмся с ещё одним возможным классом значений animation-timing-function — это steps.

Они позволяют задать «ступеньки», по которым будет идти анимация. Синтаксис steps следующий:

animation-timing-function: steps(число\_шагов, направление);

Тут всё просто: число шагов — это целое число, за которое будет выполнена вся анимация; направление может принимать значение start или end.

При заданном start первый шаг выполняется одновременно с началом анимации, а в случае c end последний шаг будет выполнен вместе с завершением анимации. То есть при start пошаговая анимация идёт как бы с опережением, а при end — вдогонку.