Теория

CSS3-переходы позволяют анимировать исходное значение CSS-свойства на новое значение с течением времени, управляя скоростью смены значений свойств.

Смена свойств происходит при наступлении определенного события, которое описывается соответствующим псевдоклассом. Чаще всего используется псевдокласс `:hover`. Данный псевдокласс не работает на мобильных устройствах, таких как iPhone или Android. Универсальным решением, работающим в настольных и мобильных браузерах, будет обработка событий с помощью JavaScript (например, переключение классов при клике).
Переходы применяются ко всем элементам, а также к псевдоэлементам `:before` и `:after`.
Для задания всех свойств перехода обычно используют краткую запись свойства `transition`.
CSS3-переходы могут применяться не ко всем свойствам и их значениям.
Название свойства transition-property
Содержит название CSS-свойств, к которым будет применен эффект перехода. Значение свойства может содержать как одно свойство, так и список свойств через запятую. При создании перехода можно использовать как начальное, так и конечное состояние элемента. Свойство не наследуется.

transition-property

Значения:

`none`	Отсутствие свойства для перехода.
`all`	Значение по умолчанию. Применяет эффект перехода ко всем свойствам элемента.
свойство	Определяет список CSS-свойств, перечисленных через запятую, участвующих в переходе.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

Продолжительность перехода transition-duration
Задаёт промежуток времени, в течение которого должен осуществляться переход. Если разные свойства имеют разные значения для перехода, они указываются через запятую. Если продолжительность перехода не указана, то анимация при смене значений свойств происходить не будет. Свойство не наследуется.

transition-duration

Значения:

время	Время перехода указывается в секундах или миллисекундах, например, `2s` или `5ms`.
`initial`	Устанавливает значение свойства в значение по умолчанию.

inherit	Наследует значение свойства от родительского элемента.

Функция перехода transition-timing-function
Свойство задаёт временную функцию, которая описывает скорость перехода объекта от одного значения к другому. Если вы определяете более одного перехода для элемент, например, цвет фона элемента и его положение, вы можете использовать разные функции для каждого свойства. Свойство не наследуется.

transition-timing-function

Значения:

ease	Функция по умолчанию, переход начинается медленно, разгоняется быстро и замедляется в конце. Соответствует cubic-bezier(0.25,0.1,0.25,1).
linear	Переход происходит равномерно на протяжении всего времени, без колебаний в скорости. Соответствует cubic-bezier(0,0,1,1).
ease-in	Переход начинается медленно, а затем плавно ускоряется в конце. Соответствует cubic-bezier(0.42,0,1,1).
ease-out	Переход начинается быстро и плавно замедляется в конце. Соответствует cubic-bezier(0,0,0.58,1).
ease-in-out	Переход медленно начинается и медленно заканчивается. Соответствует cubic-bezier(0.42,0,0.58,1).
cubic-bezier(x1, y1, x2, y2)	Позволяет вручную установить значения от 0 до 1 для кривой ускорения. На этом сайте вы сможете построить любую траекторию перехода.
initial	Устанавливает значение свойства в значение по умолчанию.
inherit	Наследует значение свойства от родительского элемента.

Задержка перехода transition-delay
Необязательное свойство, позволяет сделать так, чтобы изменение свойства происходило не моментально, а с некоторой задержкой. Не наследуется.

transition-delay

Значения:

время	Время задержки перехода указывается в секундах или миллисекундах.
initial	Устанавливает значение свойства в значение по умолчанию.

inherit	Наследует значение свойства от родительского элемента.

1. Создайте новую страницу и новый файл со стилями. Добавьте на страницу три круга, при наведении на которые они будут увеличиваться с разной скоростью. (Результат 1.gif)

Пример html

```
 <body>


<div class="wrap">
    <div class="box1 box"></div>

    <div class="box2 box"></div>

    <div class="box3 box"></div>

</div>

</body>
```

Пример css

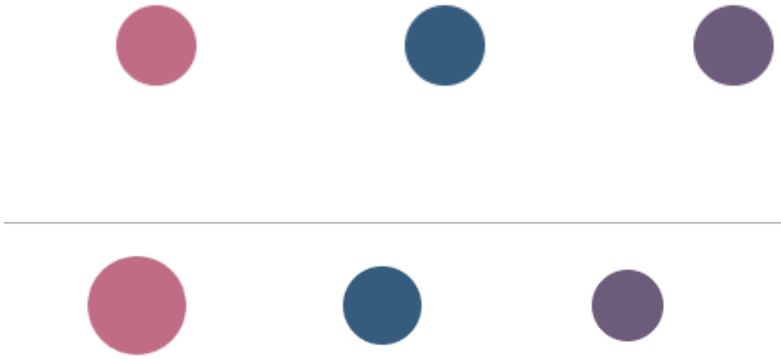
```
].box {
   height: 40px;
   width: 40px;
   border-radius: 50%;
   margin: 20px 50px 0;
   display: inline-block; /*расположение блоков в строку*/

}

.wrap:hover .box {
  -webkit-transform: scale(2); /*применяет трансформацию к элементу. в нашем случае масштабирует*/
}

.box1 {
  background: #C06C84;
  transition: .5s;
}
.box2 {
  background: #355C7D;
  transition: 1s;
}
.box3 {
  background: #6C5B7B;
  transition: 2s;
}
```

Результат



2. Добавьте на старице прямоугольник, при наведении на который, будет изменяться его форма и цвет и появляться граница. (Результат 2.gif)

Пример css

```
].Ric {
  height: 100px;
  width: 100px;
  margin: 50px 150px 0;

-webkit-transition: .5s ease-in-out;
   -moz-transition: .5s ease-in-out;
   -o-transition: .5s ease-in-out;
   transition: .5s ease-in-out;
}
.one {
background-color: #F67280;
}
.one:hover {

   border: 1px solid #000000;
   background-color: #355C7D;
   border-top-left-radius: 100% 20px;
   border-bottom-right-radius: 100% 20px;
}
```

Пример html

```
<div class="Ric one"></div>
```

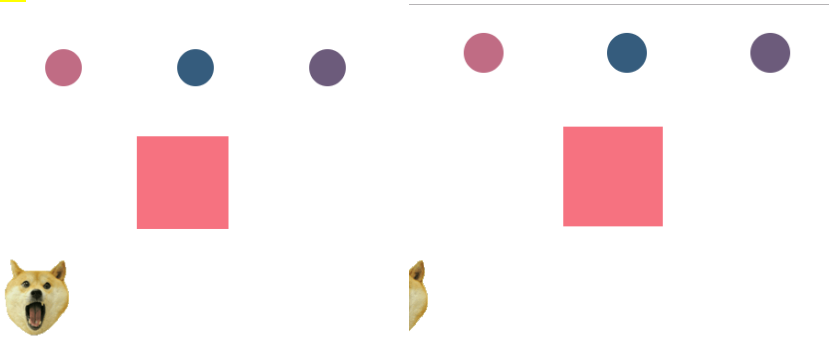
Результат

Результат

Relative Положение элемента устанавливается относительно его исходного места. Добавление свойств left, top, right и bottom изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения.

3. Добавьте на страницу картинку, которая будет слегла скрыта за границей, но при наведении на нее курсором мыши, она как бы «выезжала» из-за границы. (Результат3.gif)

Результат



Пример css

```
].posDog {
 width: 50%;
   margin-top: 10px;
   position: relative;

   width: 100px;
   height: 100px;
   background-image: url('H66FzV4YxDo.jpg');

   position: relative;
   left: -70px;
}
].posDog:hover {
   left: 100px;
}
```

Результат

```
<div class="posDog"></div>
```

4. Добавьте прямоугольник, который бы трансформировался при наведении курсора мышки. (Результат 4.gif)

Пример css

```
].RicBack {
   display: inline-block;
   margin: 20px 40px 2em 50px;
 background: rgba(228, 225, 228, .5);

}

]   .RicFront{

   width: 170px;
   height: 100px;
   line-height: 100px;

   -o-transition: all 0.5s ease-in-out;
   -moz-transition: all 0.5s ease-in-out;
   -webkit-transition: all 0.5s ease-in-out;
   transition: all 0.5s ease-in-out;
}
].four {
   background: rgba(255, 153, 0, .4);
}
].four:hover {
  -o-transform: translate(-20px, 20px);
   -ms-transform: translate(-20px, 20px);
   -moz-transform: translate(-20px, 20px);
   -webkit-transform: translate(-20px, 20px);
   transform: translate(-20px, 20px);
}
```

Пример html

```
<div class = "RicBack">
<div class="RicFront four"> </div></div>
```

Результат

5. Создайте новую страницу. Анимируйте картинки как показано в 5.gif

Результат

Пример html

```
<!DOCTYPE html>
<html>
 <head>
   <meta charset="utf-8">
   <title>dog</title>
<link rel="stylesheet" type="text/css" href="dog.css" />
 </head>
 <body>
<div>
   <span class="one"></span>
   <span class="two"></span>
   <span class="three"></span>
   <span class="four"></span>
 </div>
</body>
</html>
```

Пример css

```
body {
   margin: 0;
   background: #F4EDE5;
}
div {
   width: 100px;
   height: 100px;
   transform: translate(-50%, -50%); /* сдвигаем*/
   position: absolute;
   top: 50%;
   left: 50%;
}
span {
   display: block;
   position: absolute;
   width: 50px;
   height: 50px;
   border-radius: 50%;
   box-shadow: 2px 2px 3px rgba(0,0,0,.4);
   -webkit-animation: run 4s infinite ease-in-out;
   animation: run 4s infinite ease-in-out;
}

.one {
   background-image: url('1.png');
   -webkit-animation-delay:3s;
   animation-delay: 3s;
}
.two {
   background-image: url('2.png');
   -webkit-animation-delay: 2s;
   animation-delay: 2s;
}
.three {
   background-image: url('3.png');
   -webkit-animation-delay: 1s;
   animation-delay: 1s;
}
.four {
   background-image: url('4.png');

}
```

Внизу продолжение

```css
@-webkit-keyframes run {
  0% {
    transform: translate(0%);
    border-radius: 50%;
  }
  25% {
    transform: translate(150%) scale(0.5);
    border-radius: 0%;
  }
  50% {
    transform: translate(150%, 150%);
    border-radius: 50%;
  }
  75% {
    transform: translate(0%, 150%) scale(0.5);
    border-radius: 0%;
  }
}
@keyframes run {
  0% {
    transform: translate(0%);
    border-radius: 50%;
  }
  25% {
    transform: translate(150%) scale(0.5);
    border-radius: 0%;
  }
  50% {
    transform: translate(150%, 150%);
    border-radius: 50%;
  }
  75% {
    transform: translate(0%, 150%) scale(0.5);
    border-radius: 0%;
  }
}
```

Теория

Функции 2D-трансформации transform

Свойство задаёт вид преобразования элемента. Свойство описывается с помощью **функций трансформации**, которые смещают элемент относительно его текущего положения на странице или изменяют его первоначальные размеры и форму. Не наследуется.

Допустимые значения:

`matrix()` — любое число

`translate()`, `translateX()`, `translateY()` — единицы длины (положительные и отрицательные), `%`

`scale()`, `scaleX()`, `scaleY()` — любое число

`rotate()` — угол (deg, grad, rad или turn)

`skew()`, `skewX()`, `skewY()` — угол (deg, grad, rad)

Функция	Описание
`none`	Значение по умолчанию, означает отсутствие трансформации. Также отменяет трансформацию для элемента из группы трансформируемых элементов.

`matrix(a, c, b, d, x, y)`	Смещает элементы и задает способ их трансформации, позволяя объединить несколько функций 2D-трансформаций в одной. В качестве трансформации допустимы поворот, масштабирование, наклон и изменение положения. Значение `a` изменяет масштаб по горизонтали. Значение от 0 до 1 уменьшает элемент, больше 1 — увеличивает. Значение `c` деформирует (сдвигает) стороны элемента по оси Y, положительное значение — вверх, отрицательное — вниз. Значение `b` деформирует (сдвигает) стороны элемента по оси X, положительное значение — влево, отрицательное — вправо. Значение `d` изменяет масштаб по вертикали. Значение меньше 1 уменьшает элемент, больше 1 — увеличивает. Значение `x` смещает элемент по оси X, положительное — вправо, отрицательное — влево. Значение `y` смещает элемент по оси Y, положительное значение — вниз, отрицательное — вверх.
`translate(x,y)`	Сдвигает элемент на новое место, перемещая относительно обычного положения вправо и вниз, используя координаты X и Y, не затрагивая при этом соседние элементы. Если нужно сдвинуть элемент влево или вверх, то нужно использовать отрицательные значения.
`translateX(n)`	Сдвигает элемент относительно его обычного положения по оси X.
`translateY(n)`	Сдвигает элемент относительно его обычного положения по оси Y.
`scale(x,y)`	Масштабирует элементы, делая их больше или меньше. Значения от 0 до 1 уменьшают элемент. Первое значение масштабирует элемент по ширине, второе — по высоте. Отрицательные значения отображают элемент зеркально.
`scaleX(n)`	Функция масштабирует элемент по ширине, делая его шире или уже. Если значение больше единицы, элемент становится шире, если значение находится между единицей и нулем, элемент становится уже.

	Отрицательные значения отображают элемент зеркально по горизонтали.
`scaleY(n)`	Функция масштабирует элемент по высоте, делая его выше или ниже. Если значение больше единицы, элемент становится выше, если значение находится между единицей и нулем — ниже. Отрицательные значения отображают элемент зеркально по вертикали.
`rotate(угол)`	Поворачивает элементы на заданное количество градусов, отрицательные значения от `-1deg` до `-360deg` поворачивают элемент против часовой стрелки, положительные — по часовой стрелке. Значение `rotate(720deg)` поворачивает элемент на два полных оборота.
`skew(x-угол,y-угол)`	Используется для деформирования (искажения) сторон элемента относительно координатных осей. Если указано одно значение, второе будет определено браузером автоматически.
`skewX(угол)`	Деформирует стороны элемента относительно оси X.
`skewY(угол)`	Деформирует стороны элемента относительно оси Y.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.

1. Ключевые кадры

Ключевые кадры используются для указания значений свойств анимации в различных точках анимации. Ключевые кадры определяют поведение одного цикла анимации; анимация может повторяться ноль или более раз.

Ключевые кадры указываются с помощью правила `@keyframes`, определяемого следующим образом:

```
@keyframes имя анимации { список правил }
```

CSS

Создание анимации начинается с установки **ключевых кадров** правила `@keyframes`. Кадры определяют, какие свойства на каком шаге будут анимированы. Каждый кадр может включать один или более блоков объявления из одного или более пар свойств и значений. Правило `@keyframes` содержит имя анимации элемента, которое связывает правило и блок объявления элемента.

```
@keyframes shadow {
from {text-shadow: 0 0 3px black;}
50% {text-shadow: 0 0 30px black;}
to {text-shadow: 0 0 3px black;}
}
CSS
```

Ключевые кадры создаются с помощью ключевых слов `from` и `to` (эквивалентны значениям `0%` и `100%`) или с помощью процентных пунктов, которых можно задавать сколько угодно. Также можно комбинировать ключевые слова и процентные пункты. Если кадры имеют одинаковые свойства и значения, их можно объединить в одно объявление:

```
@keyframes move {
from,
to {
top: 0;
left: 0;
}
25%,
75% {top: 100%;}
50% {top: 50%;}
}
CSS
```

Если `0%` или `100%` кадры не указаны, то браузер пользователя создает их, используя вычисляемые (первоначально заданные) значения анимируемого свойства.

Если несколько правил `@keyframes` определены с одним и тем же именем, сработает последнее в порядке документа, а все предыдущие проигнорируются.

После объявления правила `@keyframes`, мы можем ссылаться на него в свойстве `animation`:

```
h1 {
font-size: 3.5em;
color: darkmagenta;
animation: shadow  2s infinite ease-in-out;
}
CSS
```

Не рекомендуется анимировать нечисловые значения (за редким исключением), так как результат в браузере может быть непредсказуемым. Также не следует создавать ключевые кадры для значений свойств, не имеющих средней точки, например, для значений свойства `color: pink` и `color: #ffffff`, `width: auto` и `width: 100px` или `border-radius: 0` и `border-radius: 50%` (в этом случае правильно будет указать `border-radius: 0%`).

1.1. Временная функция для ключевых кадров

Правило стиля ключевого кадра также может объявлять временную функцию, которая должна использоваться при перемещении анимации к следующему ключевому кадру.

Пример

```css
@keyframes bounce {
 from {
  top: 100px;
  animation-timing-function: ease-out;
 }
 25% {
  top: 50px;
  animation-timing-function: ease-in;
 }
 50% {
  top: 100px;
  animation-timing-function: ease-out;
 }
 75% {
  top: 75px;
  animation-timing-function: ease-in;
 }
 to {
  top: 100px;
 }
}
```

CSS

Пять ключевых кадров указаны для анимации с именем «bounce». Между первым и вторым ключевым кадром (то есть между 0% и 25%) используется функция замедления. Между вторым и третьим ключевым кадром (то есть между 25% и 50%) используется функция плавного ускорения. И так далее. Элемент будет перемещаться вверх по страницу на `50px`, замедляясь по мере того, как он достигает своей наивысшей точки, а затем ускоряясь, когда он падает до `100px`. Вторая половина анимации ведет себя аналогичным образом, но только перемещает элемент на `25px` вверх по странице.

Временная функция, указанная в ключевом кадре `to` или `100%`, игнорируется.

2. Название анимации: свойство animation-name

Свойство `animation-name` определяет список применяемых к элементу анимаций. Каждое имя используется для выбора ключевого кадра в правиле, которое предоставляет значения свойств для анимации. Если имя не соответствует ни одному ключевому кадру в правиле, нет свойств для анимации, отсутствует имя анимации, анимация не будет выполняться.

Если несколько анимаций пытаются изменить одно и то же свойство, то выполнится анимация, ближайшая к концу списка имен.

Имя анимации чувствительно к регистру, не допускается использование ключевого слова `none`. Рекомендуется использовать название, отражающее суть анимации, при этом можно использовать одно или несколько слов, перечисленных через дефис `-` или символ нижнего подчеркивания `_`.

Свойство не наследуется.

animation-name
Значения:

none	Означает отсутствие анимации. Также используется, чтобы отменить анимацию элемента из группы элементов, для которых задана анимация. Значение по умолчанию.
имя анимации	Имя анимации, которое связывает правило @keyframes с селектором.
initial	Устанавливает значение свойства в значение по умолчанию.
inherit	Наследует значение свойства от родительского элемента.

Синтаксис

```
animation-name: none;
animation-name: test-01;
animation-name: -sliding;
animation-name: moving-vertically;
animation-name: test2;
animation-name: test3, move4;
animation-name: initial;
animation-name: inherit;
```

CSS

3. Продолжительность анимации: свойство animation-duration

Свойство animation-duration определяет продолжительность одного цикла анимации. Задаётся в секундах s или миллисекундах ms. Если для элемента задано более одной анимации, то можно установить разное время для каждой, перечислив значения через запятую.

Свойство не наследуется.

animation-duration

Значения:

время	Указывает время, которое анимация занимает для завершения одного цикл Отрицательные значения недействительны. Если время равно 0s, ключевы кадры анимации не действуют, но сама анимация происходит мгновенн Значение по умолчанию 0s.

Синтаксис

```
animation-duration: .5s;
animation-duration: 200ms;
animation-duration: 2s, 10s;
animation-duration: 15s, 30s, 200ms;
```

CSS

4. Временная функция: свойство animation-timing-function

Свойство `animation-timing-function` описывает, как будет развиваться анимация между каждой парой ключевых кадров. Во время задержки анимации временные функции не применяются.
Свойство не наследуется.

animation-timing-function	
Значения:	
`linear`	Линейная функция, анимация происходит равномерно на протяжении всего времени, без колебаний в скорости.
функции Безье	
`ease`	Функция по умолчанию, анимация начинается медленно, разгоняется быстро и замедляется в конце. Соответствует `cubic-bezier(0.25,0.1,0.25,1)`.
`ease-in`	Анимация начинается медленно, а затем плавно ускоряется в конце. Соответствует `cubic-bezier(0.42,0,1,1)`.
`ease-out`	Анимация начинается быстро и плавно замедляется в конце. Соответствует `cubic-bezier(0,0,0.58,1)`.
`ease-in-out`	Анимация медленно начинается и медленно заканчивается. Соответствует `cubic-bezier(0.42,0,0.58,1)`.
`cubic-bezier(x1, y1, x2, y2)`	Позволяет вручную установить значения от 0 до 1. На этом сайте вы сможете построить любую траекторию скорости изменения анимации.
пошаговые функции	
`step-start`	Задаёт пошаговую анимацию, разбивая анимацию на отрезки, изменения происходят в начале каждого шага. Вычисляется в `steps(1, start)`.
`step-end`	Пошаговая анимация, изменения происходят в конце каждого шага. Вычисляется в `steps(1, end)`.

`steps(количество шагов,положение шага)`	Ступенчатая временная функция, которая принимает два параметра. Первый параметр указывает количество интервалов в функции. Это должно быть положительное целое число больше `0`, если вторым параметром не является `jump-none` — в этом случае оно должно быть положительным целым числом больше `1`. Второй параметр, который является необязательным, указывает позицию шага — момент, в котором начинается анимация, используя одно из следующих значений: ○ `jump-start` — первый шаг происходит при значении `0` ○ `jump-end` — последний шаг происходит при значении `1` ○ `jump-none` — все шаги происходят в пределах диапазона (0, 1) ○ `jump-both` — первый шаг происходит при значении `0`, последний — при значении `1` ○ `start` — ведет себя как `jump-start` ○ `end` — ведет себя как `jump-end` Со значением `start` анимация начинается в начале каждого шага, со значением `end` — в конце каждого шага с задержкой. Задержка вычисляется как результат деления времени анимации на количество шагов. Если второй параметр не указан, используется значение по умолчанию `end`.
`initial`	Устанавливает значение свойства в значение по умолчанию.
`inherit`	Наследует значение свойства от родительского элемента.