



АНИМАЦИИ И CSS



АЛЕКСЕЙ СУДНИЧНИКОВ



АЛЕКСЕЙ СУДНИЧНИКОВ

Веб-разработчик



[@avsudnichnikov](https://www.telegram.me/avsudnichnikov)



ПЛАН ЗАНЯТИЯ

1. [Анимация](#)
2. [css animation](#)
3. [css transform](#)
4. [easing](#)
5. [css transition + js](#)
6. [transition & animation events](#)
7. [requestAnimationFrame](#)



АНИМАЦИЯ



АНИМАЦИЯ

Анимация — один из основных инструментов в современной frontend-разработке.

Элементы анимации позволяют улучшить пользовательский интерфейс, делая его дружелюбнее для обычного пользователя.

Также за счет анимаций можно сделать продукт привлекательней, что благоприятно будет влиять на уровень его продаж.

В ТЁМНЫЕ ВРЕМЕНА...

В 1990-х большую популярность получили «незадокументированные» теги `<blink>` и `<marquee>`. Это были одни из немногих возможных вариантов «оживить» сайт. Вместе с ними для анимаций использовались gif-изображения.

Практически любой сайт в это время *blink*'ал и *marquee*'ал.

Пример (привет из 90-х)

Эти теги не входят (и никогда не входили) в спецификацию HTML, их наличие приведет к невалидному коду.



НО СЕГОДНЯ...

С появлением `css` появилась возможность (и настоятельная рекомендация) разделять стиль и верстку.

Постепенно `css` стал основным инструментом создания стилей и анимаций.



АНИМАЦИИ СЕГОДНЯ

Многие современные анимации можно сделать только с помощью CSS.

Однако в некоторых случаях разумно использовать CSS совместно с javascript.

CSS TRANSITION

Самый простой способ «оживить» объект — добавить css-переход (`transition`).



ХАМЕЛЕОН

Начнём с простого: создадим объект, при наведении на который меняется его фоновый цвет.

ХАМЕЛЕОН

Сначала создадим объект.

Подключим иконочный шрифт:

```
1 <head>
2   ...
3   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
4   ...
5 </head>
```

Блок:

```
1 <div class="chameleon chameleon-animation">
2   <i class="fa fa-hand-lizard-o" aria-hidden="true"></i>
3 </div>
```

ХАМЕЛЕОН

```
.chameleon{  
  display: inline-block;  
  color: white;  
  border: 2px solid #333;  
  border-radius: 6px;  
  font-size: xx-large;  
  cursor: pointer;  
  padding: 12px 24px;  
}
```

ХАМЕЛЕОН

Теперь сделаем так, чтобы при наведении на объект, его цвет изменялся.

```
.chameleon-animation{  
  color: red;  
}  
.chameleon-animation:hover{  
  color: blue;  
}
```

ХАМЕЛЕОН

Сделаем так, чтобы при наведении на объект, его цвет изменялся плавно:

```
.chameleon-animation{  
  color: red;  
  transition: color 2s;  
}
```



CSS TRANSITION

CSS TRANSITION

`transition` — это способ контролировать скорость анимации при изменении CSS-свойств.

Вместо того чтобы свойство применилось сразу, вы можете сделать это действие происходящим в течение какого-то момента времени.

ПАРАМЕТРЫ TRANSITION

```
.chameleon-animation{
  color: red;

  /* свойство, обрабатываемое переходом */
  transition-property: color;

  /* Длительность перехода */
  transition-duration: 3s;

  /* Временная функция */
  transition-timing-function: ease;

  /* Задержка перехода*/
  transition-delay: 0s;
}
```

TRANSITION-PROPERTY

Задаёт свойство, которое обрабатывается с помощью `transition`.

Не любое свойство можно обработать таким образом.

С перечнем допустимых значений можно ознакомиться в официальной документации.

Для указания всех свойств используется значение `all`.

Допустимо перечисление нескольких свойств через запятую.

```
transition-property: color;
```

TRANSITION-DURATION

Продолжительность перехода. По умолчанию — 0.

Допустимо задание в секундах (3s) и миллисекундах (3000ms)

```
transition-duration: 3s;
```

TRANSITION-TIMING-FUNCTION

Параметр, определяющий временную функцию (функцию плавности) воспроизведения анимации.

```
transition-timing-function: ease;
```

Остановимся на временных функциях подробнее чуть позже.

TRANSITION-DELAY

Параметр, определяющий задержку начала перехода. По умолчанию — 0.

Допустимо задание в секундах (3s) и миллисекундах (3000ms)

```
transition-delay: 0s;
```

Остановимся на временных функциях подробнее чуть позже.

КРАТКАЯ ЗАПИСЬ

Допустима короткая запись `transition:`

```
transition: color 2s ease 0s;
```



CSS ANIMATION



CSS ANIMATION

Теперь попробуем реализовать «пиратский» тег из 90-х с помощью CSS.

МИГАНИЕ

`<blink>` представляет собой постоянное изменение прозрачности объекта.

Для постоянного изменения какого-либо CSS-свойства используется свойство `animation`.



ANIMATION

`animation` — CSS-свойство, позволяющее «оживить» какой-либо объект.

`animation` действует на объект постоянно.

ПАРАМЕТРЫ ANIMATION

```
1 .blink {  
2   /* Наименование анимации */  
3   animation-name: blink;  
4  
5   /* Длительность анимации */  
6   animation-duration: 1s;  
7  
8   /* Временная функция */  
9   animation-timing-function: linear;  
10  
11  /* Количество повторений */  
12  animation-iteration-count: infinite;  
13  
14  /* Задержка анимации*/  
15  animation-delay: 2s;  
16  
17  /* Направление анимации */  
18  animation-direction: alternate-reverse;  
19  
20  /* Применение стилей до и после ее выполнения*/  
21  animation-fill-mode: none;  
22  
23  /* Пауза*/  
24  animation-play-state: running;  
25  
26 }
```

ANIMATION-NAME

Параметр, указывающий на то, какой набор ключевых кадров (`@keyframes`) будет выбран для анимации.

```
animation-name: blink;
```

@KEYFRAMES

Для задания точек анимации используется правило `@keyframes`.
У набора ключевых кадров есть своё имя.

Каждый ключевой кадр задается на определенный момент анимации, при этом этот момент указывается в процентах от продолжительности всей анимации.

```
1  @keyframes blink {  
2      0% {  
3          opacity: 0;  
4      }  
5      100% {  
6          opacity: 1;  
7      }  
8  }
```

@KEYFRAMES

Для обозначения 0% и 100% можно использовать также `from` и `to`

```
1 @keyframes blink {  
2   from {  
3     opacity: 0;  
4   }  
5   to {  
6     opacity: 1;  
7   }  
8 }
```

ANIMATION-DURATION

Параметр, определяющий длительность анимации.

Может быть указан в секундах (`1s`) или миллисекундах (`1000ms`).

По умолчанию равен нулю.

```
animation-duration: 1s;
```

ANIMATION-DELAY

Параметр, определяющий продолжительность задержки анимации. Может быть указан в секундах (`1s`) или миллисекундах (`1000ms`). По умолчанию равен нулю.

```
animation-delay: 2s;
```


ANIMATION-ITERATION-COUNT

Параметр, определяющий количество раз повторений анимации. Может быть указан как определенное количество (5) так и бесконечным (`infinite`). По умолчанию равен единице.

```
animation-iteration-count: infinite;
```

ANIMATION-DIRECTION

Параметр, определяющий направление воспроизведения анимации. Может быть указан как определенное количество (`5`) так и бесконечным (`infinite`). По умолчанию равен `normal` .

Возможные значения:

- `normal` — анимация проигрывается с начала до конца;
- `reverse` — анимация проигрывается с конца на начало;
- `alternate` — анимация проигрывается с начала до конца, потом с конца на начало;
- `alternate-reverse` — анимация проигрывается с конца на начало, потом с начала до конца;

```
animation-direction: alternate-reverse;
```

ANIMATION-TIMING-FUNCTION

Параметр, определяющий временную функцию (функцию плавности) воспроизведения анимации.

```
animation-timing-function: linear;
```

Остановимся на временных функциях подробнее чуть позже.

ANIMATION-FILL-MODE

Параметр, определяющий, как нужно применять стили к объекту анимации до и после ее выполнения

```
animation-timing-function: none;
```

Возможные значения:

- `none` — значение по умолчанию
- `forwards` — по окончании анимации элемент сохранит стили последнего ключевого кадра
- `backwards` — элемент сохранит стиль первого ключевого кадра на протяжении задержки перед началом анимации
- `both` — одновременное выполнение `forwards` и `backwards`

ANIMATION-PLAY-STATE

Параметр, определяющий состояние анимации: пауза или проигрыш.

Это можно использовать, чтобы определить текущее состояние анимации, например, в скриптах или при псевдоклассах.

```
animation-play-state: running;
```

Возможные значения:

`running` — анимация выполняется, значение по умолчанию

`paused` — анимация приостановлена

КОРОТКАЯ ЗАПИСЬ

Короткая запись:

```
1 | .blink {  
2 |   animation: blink 1s linear infinite 2s alternate-reverse;  
3 | }
```



КОРОТКАЯ ЗАПИСЬ

Не любое свойство можно обработать с помощью `animation`.

С перечнем допустимых значений можно ознакомиться в официальной документации.

СОВРЕМЕННЫЙ BLINK

Наш `blink` выглядит таким образом:

```
<h1 class="blink">Новинка!</h1>
```

```
1 .blink {  
2   animation: blink 1s linear infinite 2s alternate-reverse;  
3 @keyframes blink {  
4   100% { opacity: 0; }  
5 }
```




CSS TRANSFORM



МОНЕТКА

Теперь такая задача — некий объект производит кручение вокруг оси Y на 360 градусов каждые 10 секунд.

САМА МОНЕТКА

Подключим иконочный шрифт:

```
1 <head>
2   ...
3   <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css">
4   ...
5 </head>
```

```
1 <div class="coin coin-animation">
2   <i class="fa fa-phone" aria-hidden="true"></i>
3 </div>
```

CAMA MOHETKA

```
1  .coin{
2    width: 60px;
3    height: 60px;
4    border-radius: 50%;
5    display: flex;
6    justify-content: center;
7    align-items: center;
8    flex-direction: column;
9    cursor: pointer;
10   font-weight: 700;
11   font-size: xx-large;
12   background: seagreen;
13   color: white;
14 }
```

АНИМАЦИЯ МОНЕТКИ

```
1  .coin-animation {  
2    animation: coin-animation 10s infinite;  
3  }  
4  
5  @keyframes coin-animation{  
6    0%{/* изменение */}  
7    10%{/* возвращение к исходному состоянию */}  
8  }
```

TRANSFORM

CSS-свойство `transform` позволяет сдвигать, поворачивать и масштабировать элементы.

Действие `transform` действует только на изменяемый элемент, не затрагивая остальные элементы веб-страницы, т.е. другие элементы не сдвигаются относительно него.

TRANSFORM: MATRIX(A, C, B, D, X, Y)

Преобразование с использованием матрицы из девяти элементов.

Имеет в своей основе математическую модель матрицы, желающие могут ознакомиться подробнее на официальном сайте MDN.

Для простоты можно представить матрицу несколькими параметрами:

a — масштаб по горизонтали

d — масштаб по вертикали

b — наклон по горизонтали

c — наклон по вертикали

x — смещение по горизонтали в пикселях

y — смещение по вертикали в пикселях

Все прочие 2D-трансформации основаны на `matrix()`

MATRIX(1,0,0,1,0,0)

```
matrix(1,0,0,1,0,0)
```



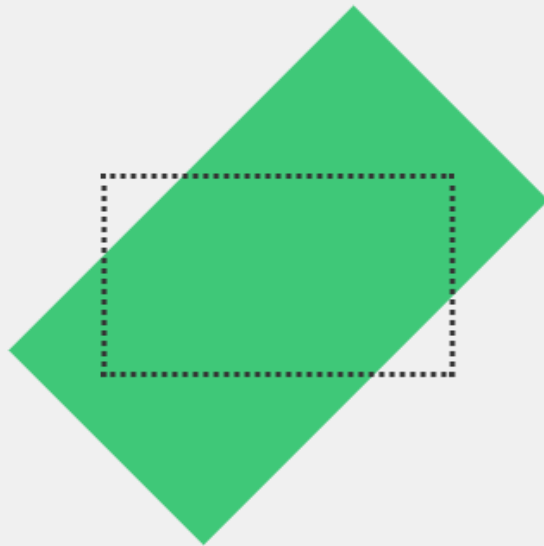
MATRIX(1,0,1,1,0,0)

```
matrix(1,0,1,1,0,0)
```



MATRIX(1,-1,1,1,0,0)

```
matrix(1,-1,1,1,0,0)
```



TRANSFORM – 2D

`translate(x, y)` – перенос по осям x и y

`translateX(x)` – перенос по оси x

`translateY(y)` – перенос по оси y

`scale(x, y)` – масштаб (растягивание)

`scaleX(x)` – масштаб (растягивание) по горизонтали

`scaleY(y)` – масштаб (растягивание) по вертикали

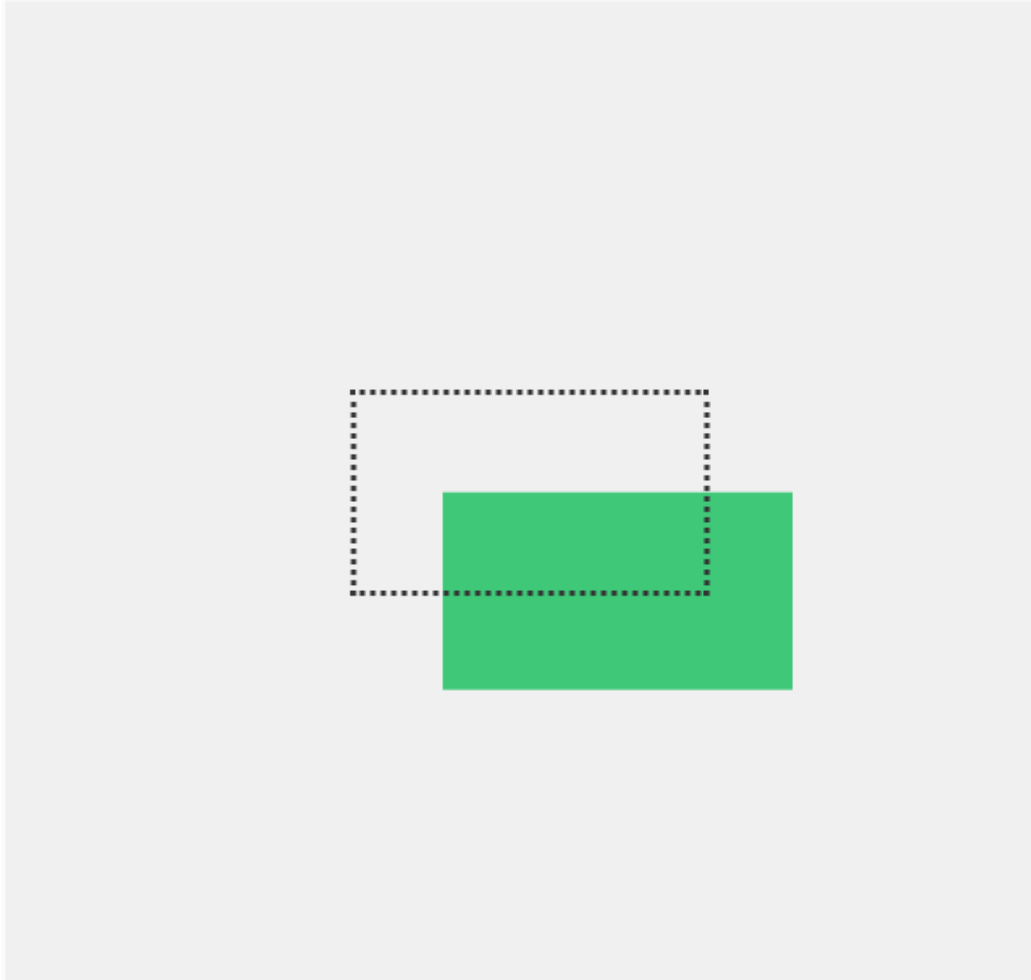
`rotate(a)` – поворот по часовой стрелке (в градусах, например, *'30deg'*)

`skew(x-угол, y-угол)` – наклон (в градусах, например *'30deg'*)

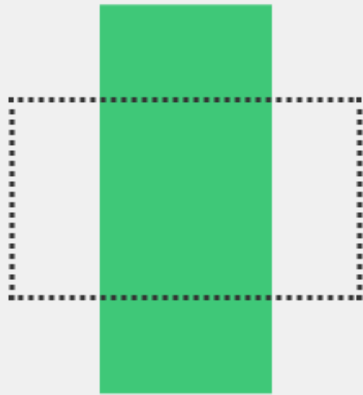
`skewX(x-угол)` – наклон по оси X (в градусах, например *'30deg'*)

`skewY(y-угол)` – наклон по оси Y (в градусах, например *'30deg'*)

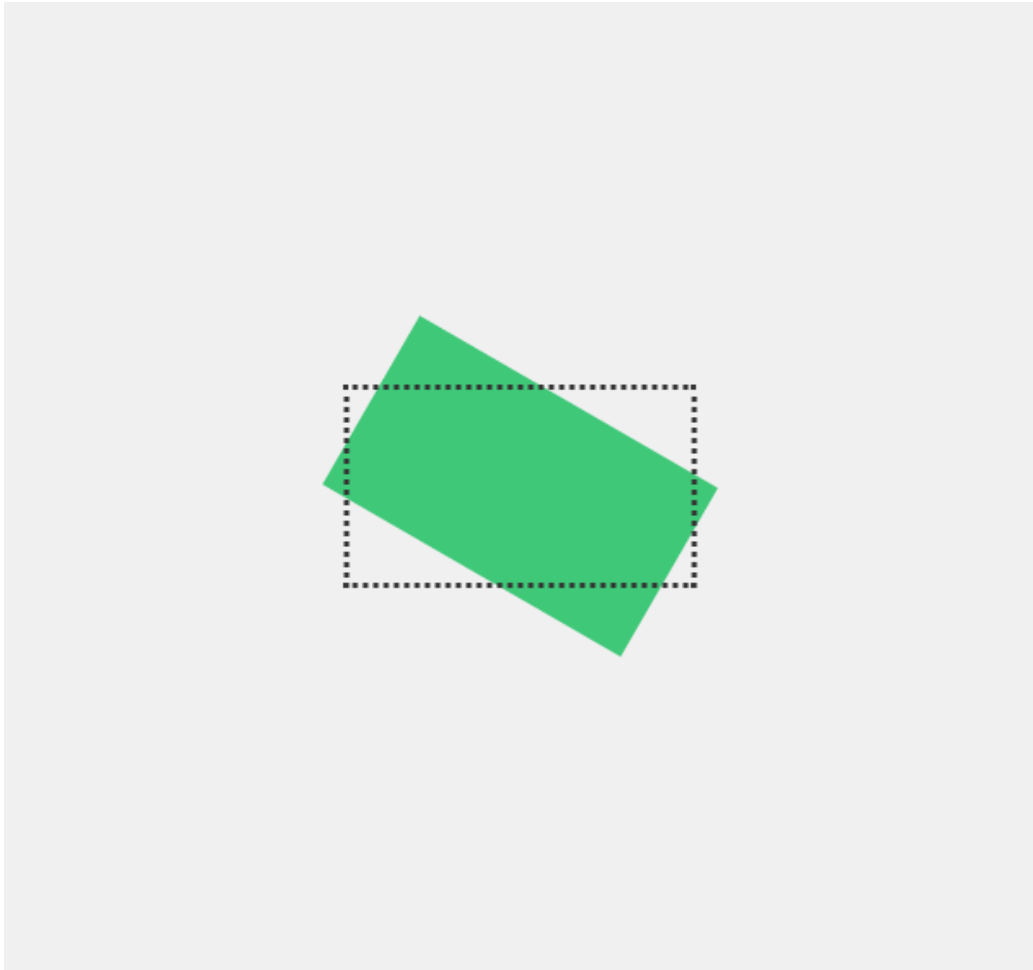
TRANSFORM: TRANSLATE(45PX,40PX)



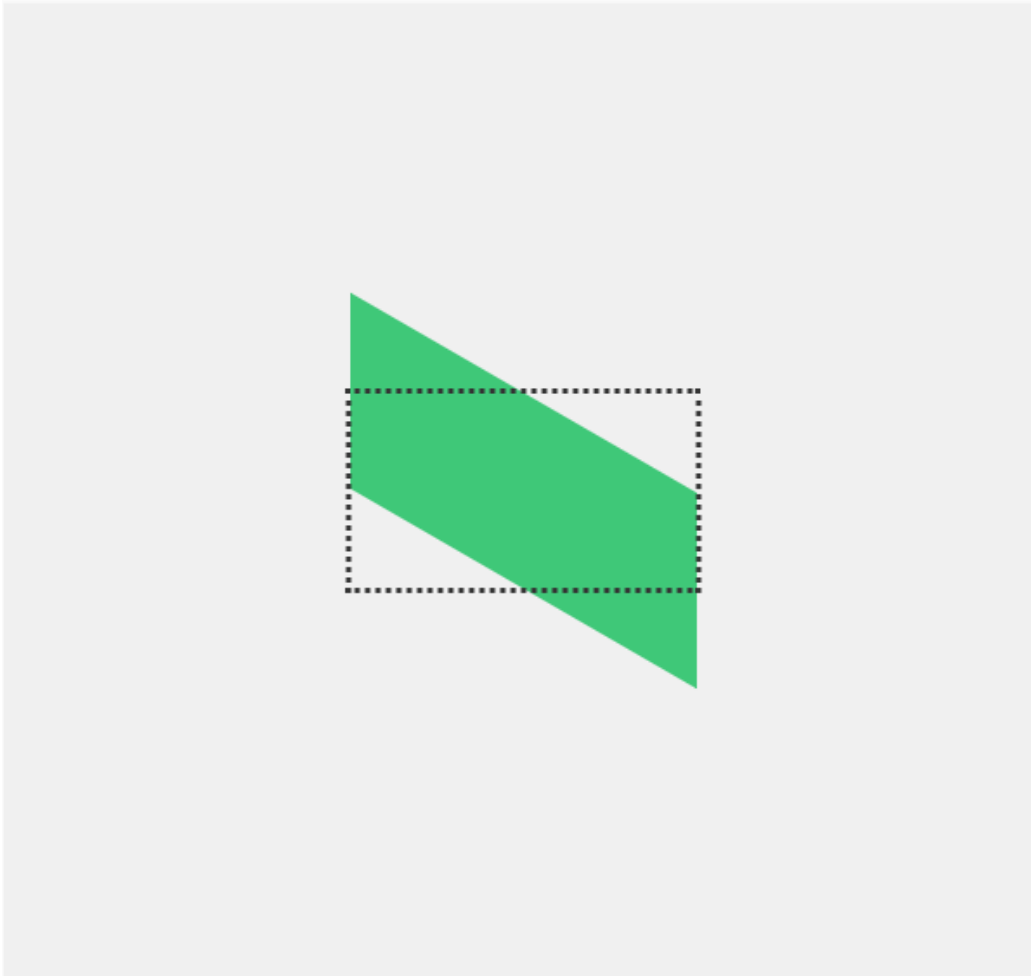
TRANSFORM: SCALE(0.5,2)



TRANSFORM: ROTATE(30DEG)



TRANSFORM: SKEWY(30DEG)



MATRIX3D

`matrix3d()` – преобразование, основанное на матрице из 16 элементов.

Нормальное положение блока:

```
matrix3d(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1)
```

Все прочие 3D-трансформации основаны на `matrix3d()`

TRANSFORM – 3D

`translate3d(x, y, z)` – перенос по осям x,y,z

`translateZ(z)` – перенос по оси z

`scale3D(x, y, z)` – масштаб (растягивание)

`scaleZ(x)` – масштаб (растягивание) по оси Z, делает элемент больше или меньше

`rotate3d()` – поворот по осям x,y,z

`rotateX()` – поворот по оси x

`rotateY()` – поворот по оси y

`rotateZ()` – поворот по оси z

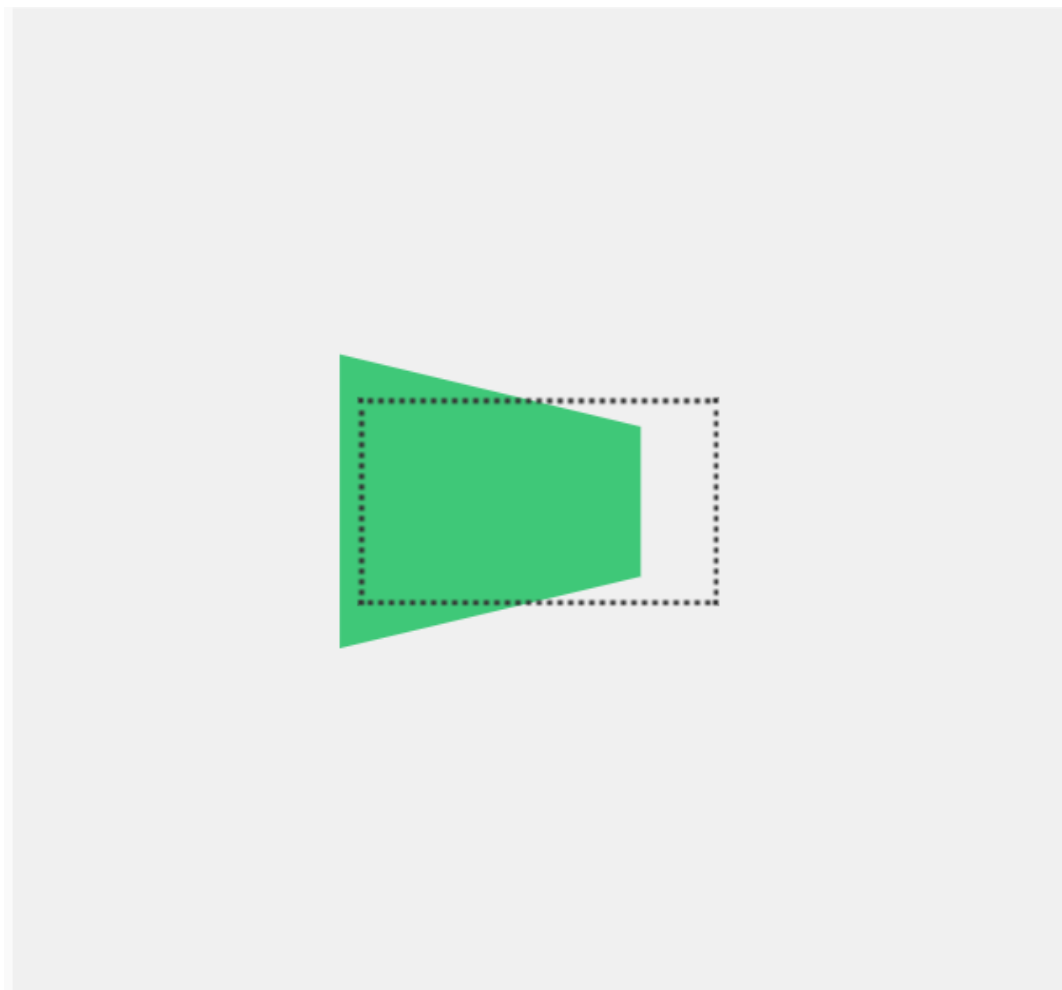
TRANSFORM: ROTATEX(40DEG)

У родительского элемента установлено `perspective: 20vh`



TRANSFORM: ROTATEY(40DEG)

У родительского элемента установлено `perspective: 20vh`



MOHETKA

```
1  .coin-animation {  
2    animation: coin 10s infinite;  
3  }  
4  
5  @keyframes coin{  
6    0%{transform: rotateY(360deg);}  
7    10%{transform: rotateY(0deg)}  
8  }
```



EASING



EASING

Временная функция (функция плавности) воспроизведения анимации.

Задаёт плавность анимации.



EASING

Временная функция задает скорость каждого этапа анимации, например, плавное начало или конец, продолжительность этого промежутка анимации.



EASING










Существует несколько предустановленных значений:

- ease (*значение по умолчанию*)
- ease-in
- ease-out
- ease-in-out
- linear

EASING

Существует возможность задания пошаговой анимации через `steps` и создание пользовательской временной функции через `cubic-bezier`.

EASING

linear	
ease	
ease-in	
ease-out	
ease-in-out	
steps(5, end)	
cubic-bezier(1,0,0,1)	
cubic-bezier(1,0,1,0)	
cubic-bezier(0.5,-1,0.5,2)	

МОНЕТКА

Добавим возможность смены цвета объекта при повороте.

```
1  .coin-animation {
2      animation: coin-animation 10s linear 2s infinite,
3          bgChange 10s steps(1) 2s infinite;
4  }
5
6  @keyframes coin-animation{
7      0%{transform: rotateY(360deg);}
8      10%{transform: rotateY(0deg)}
9  }
10
11 @keyframes bgChange{
12     2.5%{background: darkblue;}
13     7.5%{background: seagreen}
14 }
```



CSS TRANSITION + JS

МОНЕТКА №2

Рассмотрим другую задачу.

Требуется выполнять описанное выше кручение монетки при клике на объект.

Сделаем эту анимацию следующим образом:

1. Создадим класс объекта в «спокойном» состоянии
2. Создадим класс объекта в состоянии, которое достигается при анимации
3. С помощью скрипта будем производить «навешивание» класса на объект при клике

Для плавного перехода между состояниями используем `transiton`

МОДЕТКА №2

```
1 <div class="coin coin_2" id="coin_2">
2   <i class="fa fa-phone" aria-hidden="true"></i>
3 </div>
```

МОДЕТКА №2

```
1  .coin_2 {  
2    transform: rotateY(0deg);  
3  }  
4  .coin_2_second_state {  
5    transform: rotateY(360deg);  
6    transition: transform 1s linear;  
7  }
```

```
1  document.getElementById('coin_2').addEventListener('click', ({currentTarget}) => {  
2    currentTarget.classList.add('coin_2_second_state');  
3    setTimeout(() => {  
4      currentTarget.classList.remove('coin_2_second_state');  
5    }, 1000);  
6  });
```



МОНЕТКА №2

В этом случае мы производили анимацию совместным использованием javascript и css.

С помощью css были описаны различные состояния объекта, переход между этими состояниями контролируется скриптом, анимация же представляет собой плавный переход между состояниями с помощью `transition`.



МОНЕТКА №2

Однако использование `setTimeout` для определения окончания анимации — достаточно «костыльное» решение.



TRANSITION & ANIMATION EVENTS

TRANSITION EVENTS

У объектов, использующих `transition` есть возможность «отловить» следующие события:

- `transitionstart`
- `transitionrun`
- `transitioncancel`
- `transitionend`



TRANSITIONSTART

Событие, которое происходит при начале перехода (после задержки, если она есть).



TRANSITIONRUN

Событие, которое происходит при начале перехода (до задержки, если она есть).

`transitionrun` произойдет, даже если переход отменен до истечения срока задержки.



TRANSITIONCANCEL

Событие, которое произойдёт при отмене перехода.



TRANSITIONEND

Событие, которое произойдёт по окончании перехода.

TRANSITIONEND

Добавим в скрипт прослушивание окончания перехода:

```
1 document.getElementById('coin_2').addEventListener('click', ({currentTarget}) => {  
2   currentTarget.classList.add('coin_2_second_state');  
3   currentTarget.addEventListener('transitionend', ({currentTarget}) => {  
4     currentTarget.classList.remove('coin_2_second_state');  
5   });  
6 });
```


ANIMATION EVENTS

Подобные события есть и у `animation`

- `animationstart`
- `animationcancel`
- `animationiteration`
- `animationend`



ANIMATIONSTART

Событие, которое происходит при начале анимации (после задержки, если она есть).



ANIMATIONCANCEL

Событие, которое произойдёт при отмене анимации.



ANIMATIONITERATION

Событие, которое произойдёт при окончании одного повторения анимации и начале другого.



ANIMATIONEND

Событие, которое произойдёт при окончании анимации. Не происходит, если после окончания анимация повторяется.



REQUESTANIMATIONFRAME

REQUESTANIMATIONFRAME()

Периодически, ознакомливаясь с темой анимации в JS, вы можете встретить функцию `requestAnimationFrame`.

`window.requestAnimationFrame` указывает браузеру, что требуется произвести перерисовку на следующем кадре анимации.

В качестве параметра метод получает функцию, которая будет вызвана перед перерисовкой.

REQUESTANIMATIONFRAME

По факту, анимация (как мы уже рассмотрели) — это изменения одного из параметров объекта во времени. Поэтому возникает вопрос, зачем нам эта функция?

Q: Зачем она нужна? Ведь можно работать через `setTimeout` / `setInterval` ?

A: Да, можно и так, но здесь весь вопрос в оптимизации и производительности. У браузера есть специальные «этапы», когда он собирается заняться перерисовкой экрана.

Именно использование `requestAnimationFrame` позволит вам «встроиться» в эти этапы и показывать вашу анимацию «плавнее». Кроме того, браузер «сгруппирует» все анимации и более эффективно их отрисует.

REQUESTANIMATIONFRAME

```
1 <div class="square requestAnimationFrame"  
2 id="requestAnimationFrameSquare"></div>
```

```
1 .square{  
2   width: 60px;  
3   height: 60px;  
4   background: #3fc878;  
5   position: absolute;  
6   left: 0;  
7 }
```

REQUESTANIMATIONFRAME

```
1  const start = Date.now();
2  const obj = document.getElementById('requestAnimationFrameSquare');
3  const posMax = (obj.parentNode.clientWidth - obj.clientWidth) * 10;
4
5  function draw() {
6      const progress = Date.now() - start;
7      obj.style.left = progress/10 + "px";
8      if (progress <= posMax) {
9          window.requestAnimationFrame(draw);
10     }
11 }
12
13 window.requestAnimationFrame(draw);
```

REQUESTANIMATIONFRAME

В большинстве браузеров в фоновых вкладках или скрытых `<iframe>`, вызовы `requestAnimationFrame()` приостанавливаются, для того чтобы повысить производительность и время работы батареи.

REQUESTANIMATIONFRAME

В большинстве случаев предпочтительно использование CSS-переходов и CSS-анимаций. `requestAnimationFrame` стоит же использовать тогда, когда вы делаете сложную анимацию (взаимодействие нескольких элементов и т.д.).



JS VS CSS

Стоит ещё раз отметить, что предпочтительным является использование CSS-анимаций и переходов, а JS - для реагирования на события, установки нужных классов (запускающих анимацию) и конечных значений при переходах.

НАМИ БЫЛИ РАССМОТРЕНЫ:

- CSS — переходы
- CSS — анимация
- CSS-свойство `transform`
- easing
- события анимаций и переходов

ИНТЕРЕСНОЕ ЧТИВО

- [Привет из 90-х](#)
- [MDN – transition](#)
- [MDN – animation](#)
- [MDN – использование animation](#)
- [MDN – animation-fill-mode](#)
- [MDN – transform](#)
- [MDN – transform: типы преобразований на русском языке](#)
- [w3.org – transform \(не для слаонервных\)](#)
- [MDN – perspective](#)
- [basicweb.ru – perspective](#)
- [MDN – backface-visibility](#)
- [https://easings.net](#)



Спасибо за внимание!

Время задавать вопросы

АЛЕКСЕЙ СУДНИЧНИКОВ



[@avsudnichnikov](https://www.instagram.com/avsudnichnikov)