

# CSS Animations

Lesson 16



# Lesson Plan

1

Transformations

2

Transitions

3

Animations

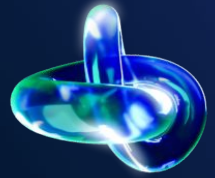
4

3D Transformations and Animations

5

Vector Animation – SVG

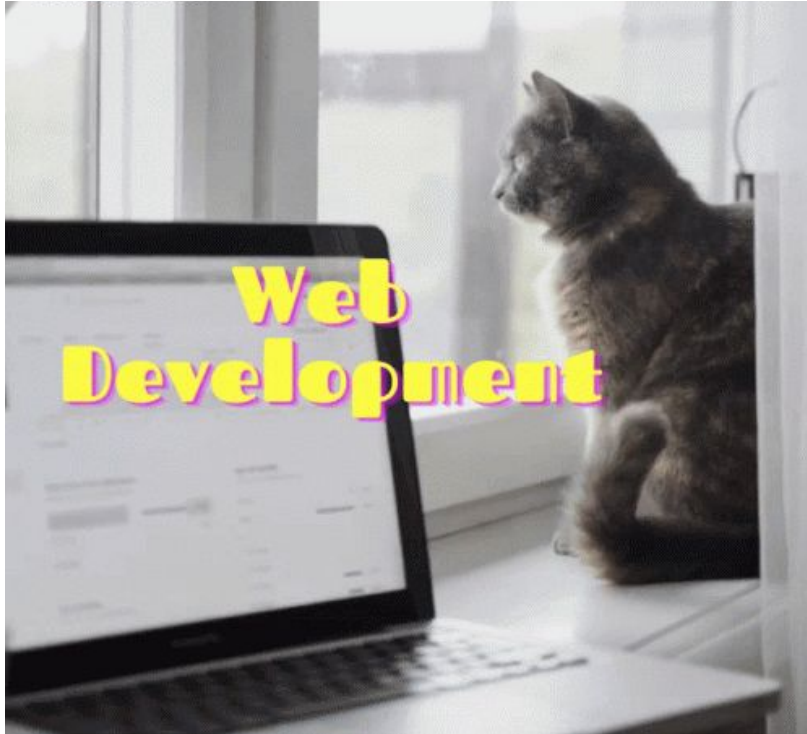




---

# Dynamic on the page

# gif / webp / avif / apng - image formats



demo



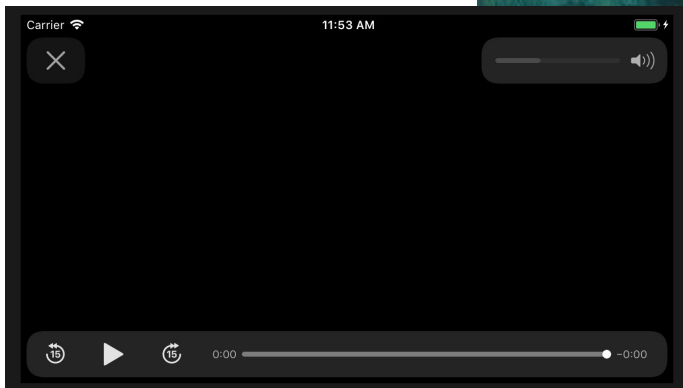
**CSS animation**

**SVG**

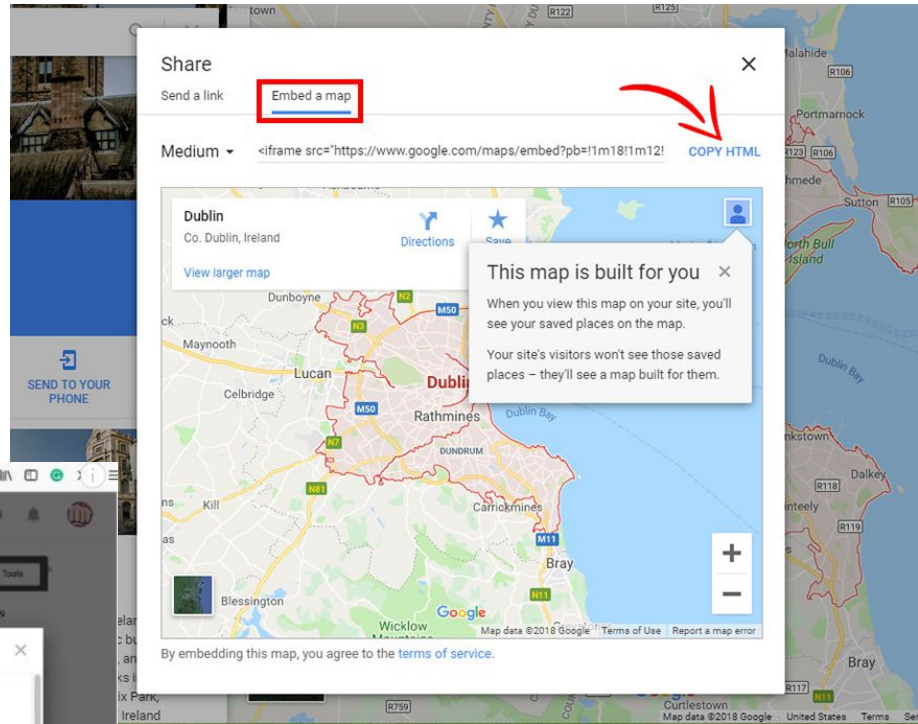
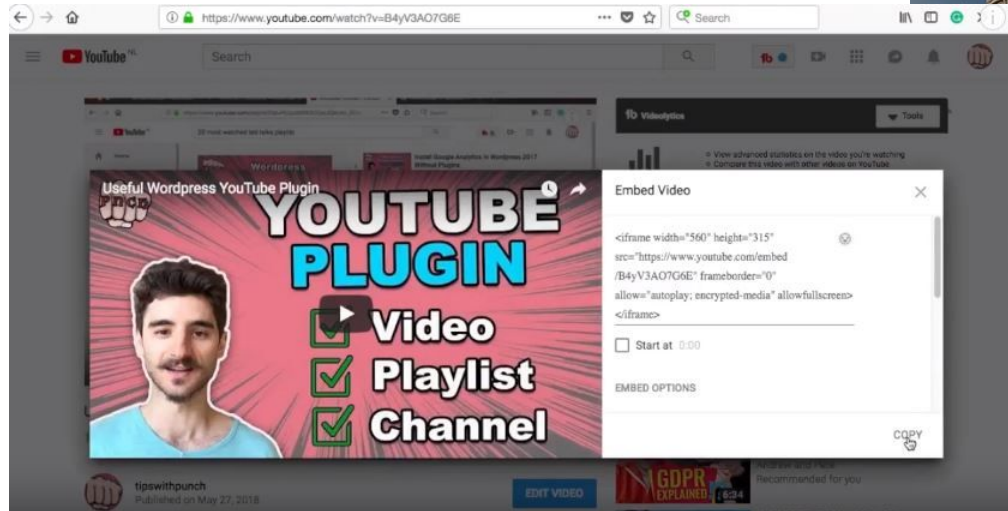
**Demo**  
**Demo**

# <video> tag

demo



# <iframe> tag



**demo**

# <canvas> tag (via Javascript)

[demo](#) [demo](#) [demo](#)



The screenshot shows a web browser interface. The top toolbar includes icons for home, menu, save, print, and a 'Run »' button. The main content area is split into two panels. The left panel displays HTML and JavaScript code for a canvas. The right panel shows the rendered output of the code, which is a black curve on a white background.

```
<!DOCTYPE html>
<html>
<body>

<canvas id="myCanvas" width="300" height="400" style="border:1px solid
#d3d3d3;">
Your browser does not support the HTML5 canvas tag.</canvas>

<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.beginPath();
ctx.moveTo(20, 20);
ctx.bezierCurveTo(30, 100, 200, 100, 200, 200);
ctx.stroke();
</script>

</body>
</html>
```



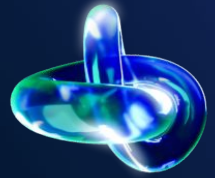


# WebGL

demo

game





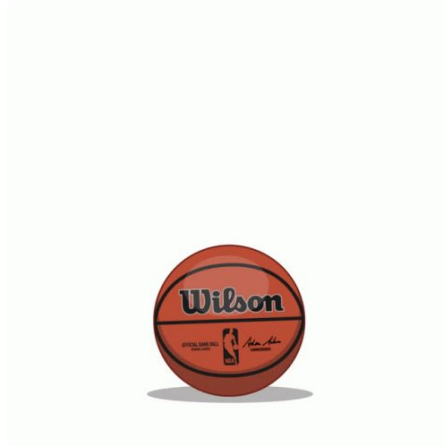
---

# Transformations

# Transform

- Changing the properties of an object **without affecting the document flow**.

You can transform everything except table elements and inline boxes.



LOADING VS LOADING  
translateX VS translateY

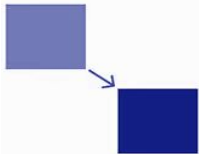
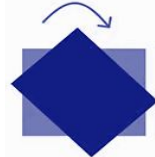


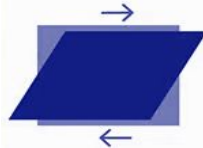
LOADING VS LOADING  
rotateX VS rotateY  
LOADING  
rotateZ

LOADING VS LOADING  
scaleX VS scaleY  
LOADING  
scale

LOADING VS LOADING  
skewX VS skewY

# Transform

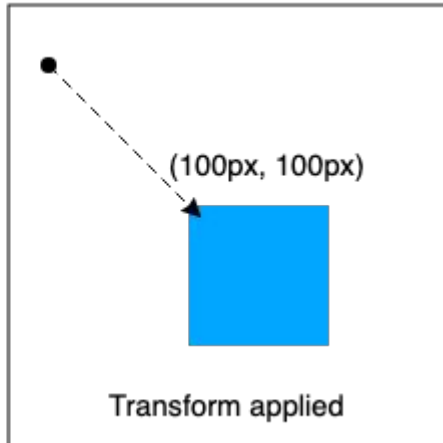
**demo**

<pre> : translate(x, y)/(-x/-y) : translateX(x) : translateY(y) </pre>	<pre> = translate: x y; = translate: x; = translate: 0 y; </pre>	<pre>% px</pre>	<b>Moving</b> relative to the document flow	
<pre> : rotate(angle)/(-angle) : rotateX(angle) : rotateY(angle) </pre>	<pre> = rotate: angle; = rotate: x angle; = rotate: y angle; </pre>	<pre>deg</pre>	<b>Rotating</b> clockwise or counterclockwise	
<pre> : scale(x, y)/(-x/-y) : scaleX(x) : scaleY(y) </pre>	<pre> = scale: x y; = scale: x; = scale: 0 y; </pre>	<pre>0-10 %</pre>	<b>Scaling up or down.</b> Negative values <b>flip</b> the element	
<pre> : skew(angle, angle)/(-angle) : skewX(angle) : skewY(angle) </pre>		<pre>deg</pre>	<b>Stretching</b> along a specific axis	

# Transform rules

Any transformation can be applied to **both axes** –  
**together or separately**

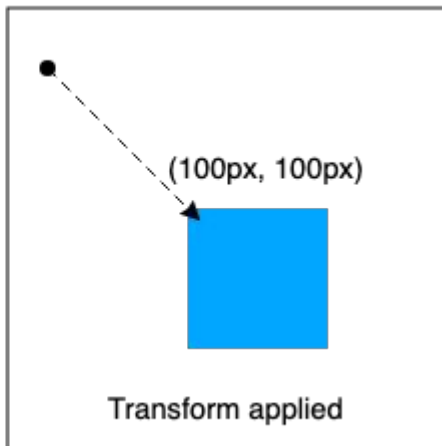
```
transform: translate(100px, 100px) =  
transform: translateX(100px)  
translateY(100px)
```



# Transform rules

If only **one value** is specified in the brackets, it applies to both axes

```
transform: translate(100px) =  
transform: translateX(100px)  
              translateY(100px)
```



# Transform rules

You can list multiple transformations separated by spaces; **order matters**, as they will be applied sequentially.

```
transform: translateX(10px) rotate(10deg)
          translateY(5px);
```

**The order of individual properties doesn't matter**, they will be applied in the established order.

```
scale: 2;
rotate: 15deg;
translate: 5px;
```



# Transform rules

It's more convenient to use new **shorthand properties**

```
translate: x y z;  
rotate  
scale
```

# Transform: **matrix**

– Combines all the transform values.

- It covers all properties at once.
- Maximum flexibility in settings.
- You might need to recall higher mathematics.
- You will probably never use this.

$$\text{matrix}[1, 2, 3, 4, 5, 6] = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{scale}(a) = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{scaleX}(x) = \begin{bmatrix} x & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{scaleY}(y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{translate}(x, y) = \begin{bmatrix} 0 & 0 & x \\ 0 & 0 & y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{translateX}(x) = \begin{bmatrix} 0 & 0 & x \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{translateY}(y) = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & y \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{skewX}(x) = \begin{bmatrix} 1 & \tan x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{skewY}(y) = \begin{bmatrix} 1 & 0 & 0 \\ \tan y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{rotate}(\theta) = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Transform-origin

The point relative to which transformations occur.

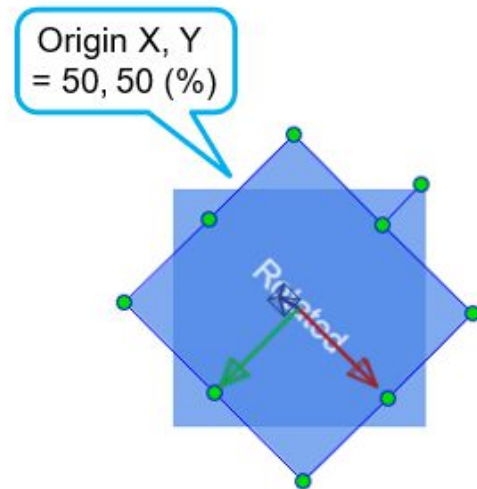
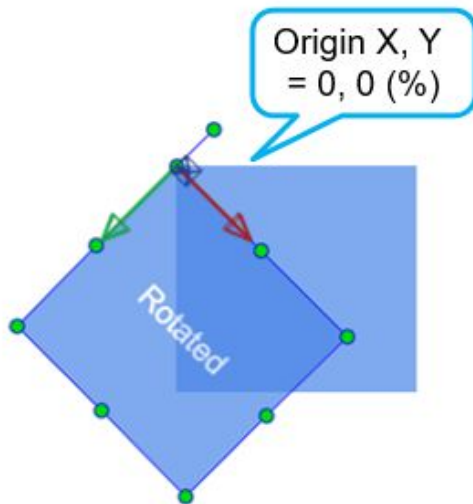
By default (0, 0) – top-left corner.

**center** – (50%, 50%)

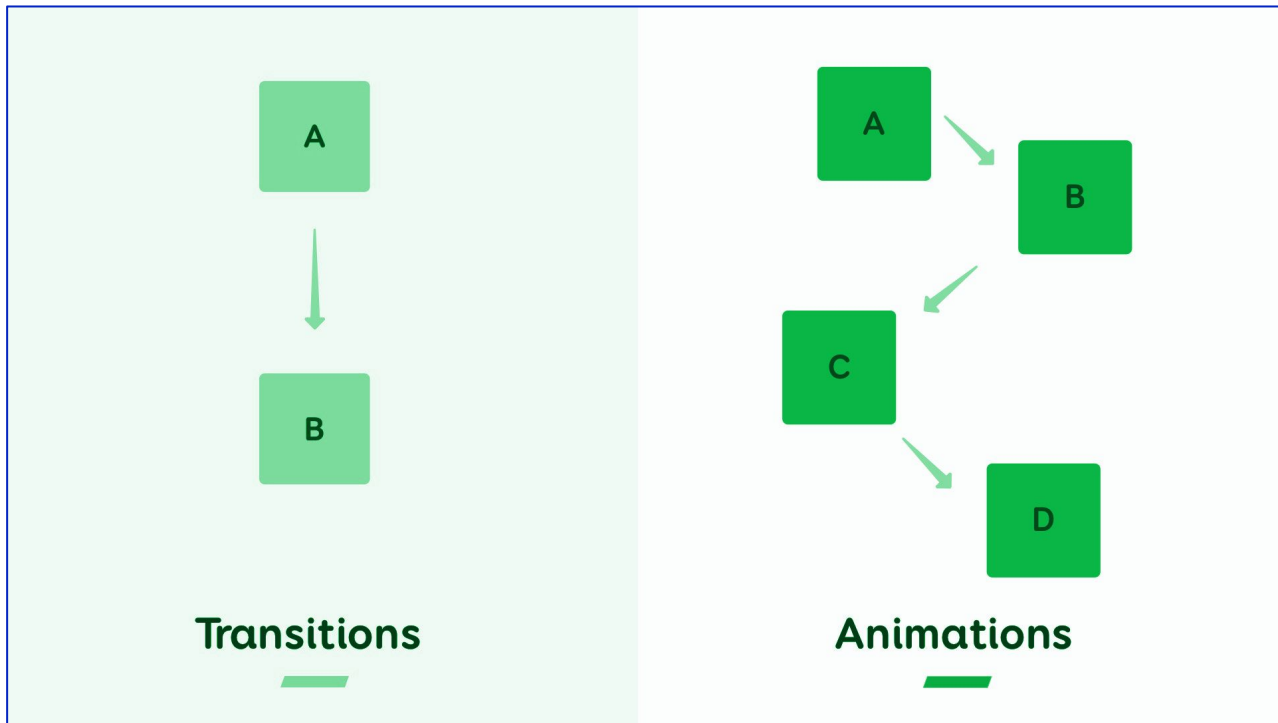
Units:

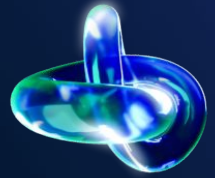
- px
- %

[demo](#)



# Ways to Animate an Element





---

# Transition

# Transition

- `transition-property(: all)` – What to animate
- `transition-duration(: 0s)` – Animation duration (ms, s)
- `transition-timing-function(: ease)` – Animation speed
- `transition-delay(: 0s)` – Delay before animation (ms, s)

```
transition: property duration timing-function delay;
```

# Timing functions

– describes how the animation process will be distributed over time (when it speeds up and when it slows down)

## Bézier curve

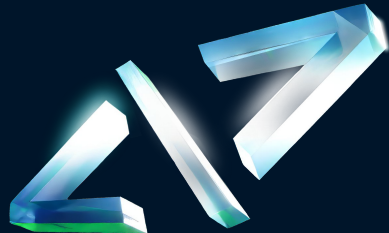
`cubic-bezier(x2, y2, x3, y3)`

## Keywords

`ease`, `ease-in`, `ease-out`,  
`ease-in-out`, `linear`

## Steps

```
steps(  
    the number of steps,  
    jump-start/-end/-none/-both  
)  
step-start = (1, jump-start)  
step-end = (1, jump-end)
```



# timing function: keywords

Demo

Uniform speed

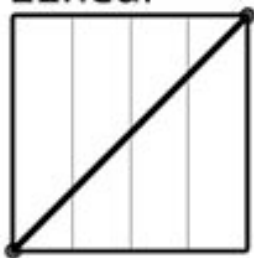
Speeds up in the middle and slows down towards the end (default value).

Starts slow, then speeds up.

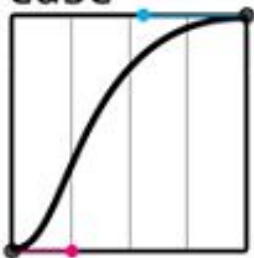
Starts fast, then gradually slows down.

Slow, fast, slow.

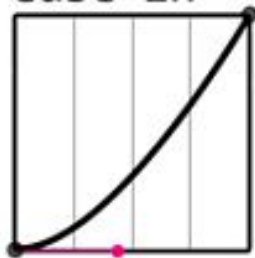
linear



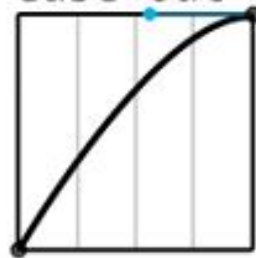
ease



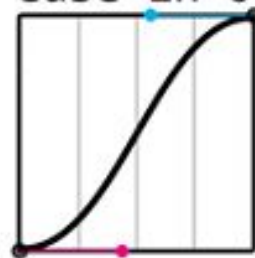
ease-in



ease-out



ease-in-out

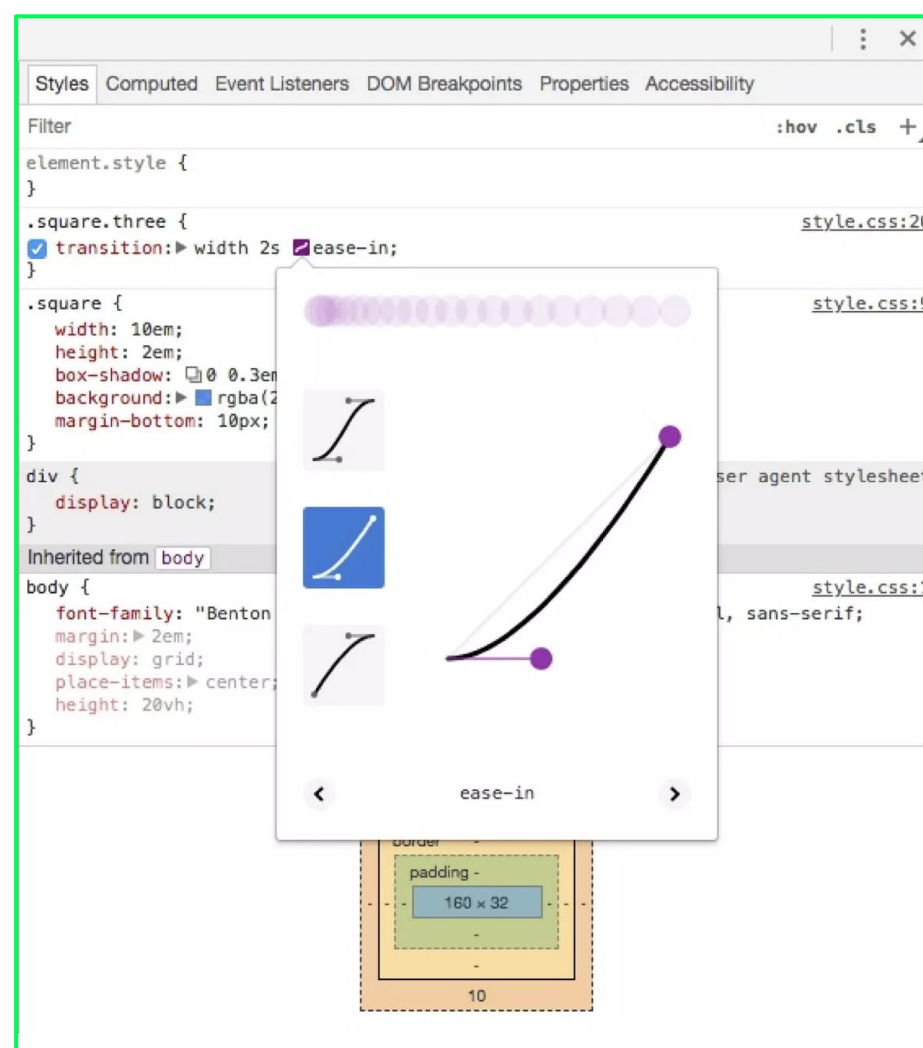




# in DevTools

**Provides a visual representation of the animation timing**

allows you to adjust the curve, including selecting cubic-bezier values.

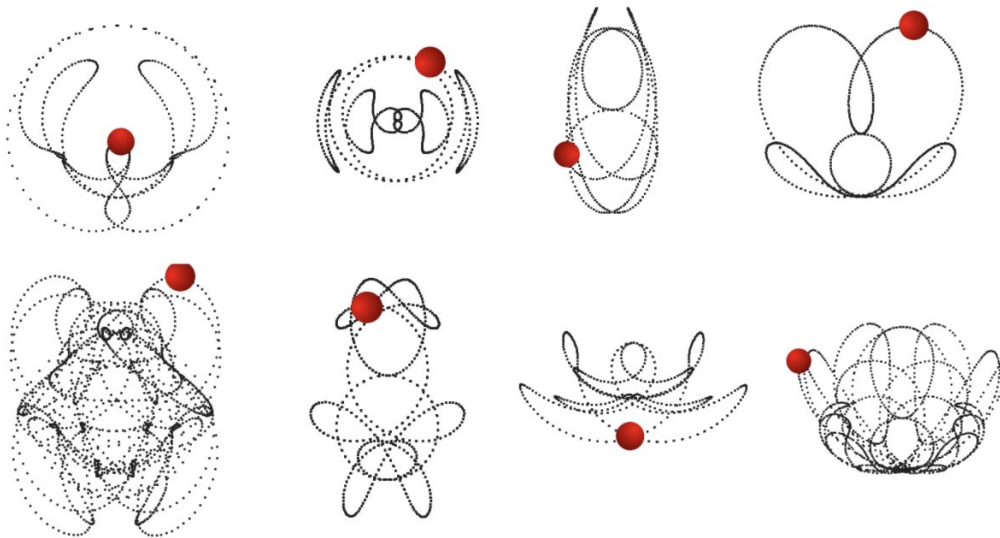


# timing function: cubic-bezier

demo

Demo loaders

– Allows you to define a custom animation timing curve.



- This provides the ability to create complex motion paths for objects
- and to set different positions for transitions instead of just two.

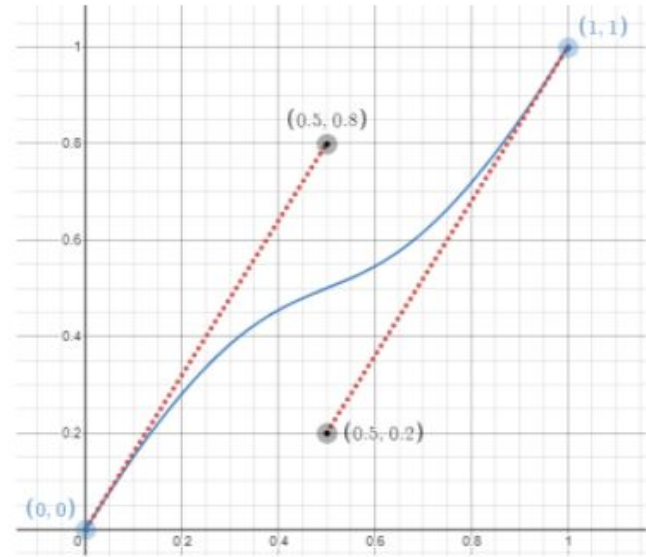
# timing function: cubic-bezier

`cubic-bezier(x1, y1, p1, p2, x2, y2)`

`x1, y1` – First coordinate

`x2, y2` – Final coordinate

`p1, p2` – Any other intermediate value pairs



Cubic-bezier(0.5,0.8,0.5,0.2)

# timing function: **steps**

The animation does not run smoothly but pauses at each step for the same amount of time.



Used together with sequences to create the effect of a moving picture (step or frame animation).

[demo](#)

[demo](#)



demo

demo

## timing function: steps

The animation does not run smoothly but pauses at each step for the same amount of time.

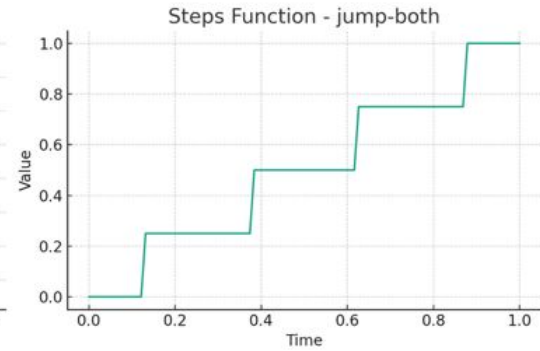
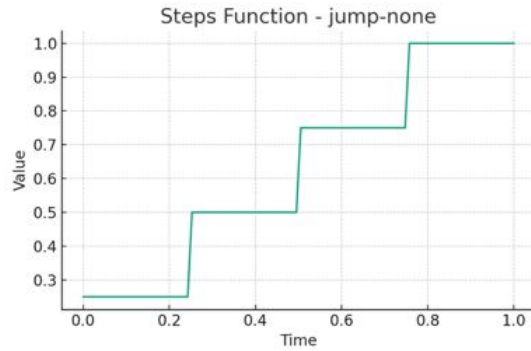
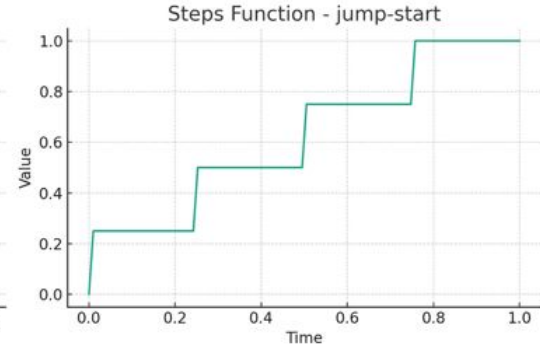
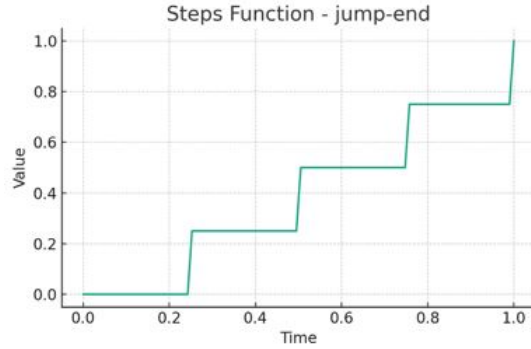
Used together with sequences to create the effect of a moving picture (step or frame animation).



# timing function: values for **steps**

- **start/jump-start** – The frame changes at the beginning of each second.
- **end/jump-end** – The frame changes at the end of each second.
- **jump-none** – The animation will not change on the first and last steps (-2 frames).
- **jump-both** – The frame sharply changes at the start and end of each step (+2 frames).

```
transition-timing-function: steps(10, jump-start);
```



**Demo**

`steps(4, *)`

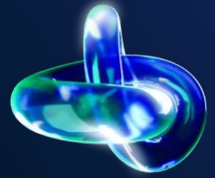
# will-change

Tells the browser that an element's property will change soon, giving the browser time to prepare.

- Use it sparingly and only for elements that change frequently.
- It's a good idea to remove this property after the animation is done (can be done using JavaScript).

```
.btn {  
    transition: transform 1s  
    ease;  
    background-color: white;  
    border: 1px solid blue;  
    transform: rotate(15deg);  
}  
  
.btn:hover {  
    will-change: transform;  
}  
  
.btn:active {  
    transform: none;  
}
```





---

# Animations

# Creating Animation with @keyframes

Sequentially describes the keyframes of the animation.

```
@keyframes animation-name {  
  from {  
    rotate: 0;  
  }  
  to {  
    rotate: 360deg;  
  }  
}
```

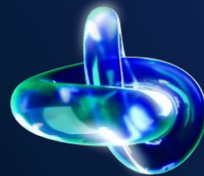
```
@keyframes animation-name {  
  0%, 50% {  
    rotate: 0;  
  }  
  33%, 66% {  
    rotate: 180deg;  
  }  
  100% {  
    rotate: 360deg;  
  }  
}
```

# @keyframes



- **Number of frames:** Any number
- **Frames:** from & to or from 0% to 100%
- **Animation name:** Can be any name, but it's better if it describes what the animation does.
- The same animation can be applied to different selectors
- is usually placed at the end of the CSS code.

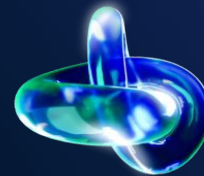
# ANIMATION



- **animation-name(: none)** – The name of the animation from @keyframes
- **animation-duration(: 0s)** – Duration of the animation (ms, s)
- **animation-timing-function(: ease)** – Speed of the animation
- **animation-delay(: 0s)** – Delay before the animation starts (ms, s)

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

# ANIMATION



- `animation-iteration-count`(: 1) – Number of repetitions (`infinite`, or a specific number from 1)
- `animation-direction`(: `normal`) – Direction of the animation
- `animation-fill-mode`(: `none`) – How styles should be applied before and after the animation
- `animation-play-state`(: `running`) – Controls the running state of the animation (`paused`)

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

# animation-**fill-mode**

— how styles should be applied before and after the animation

## demo

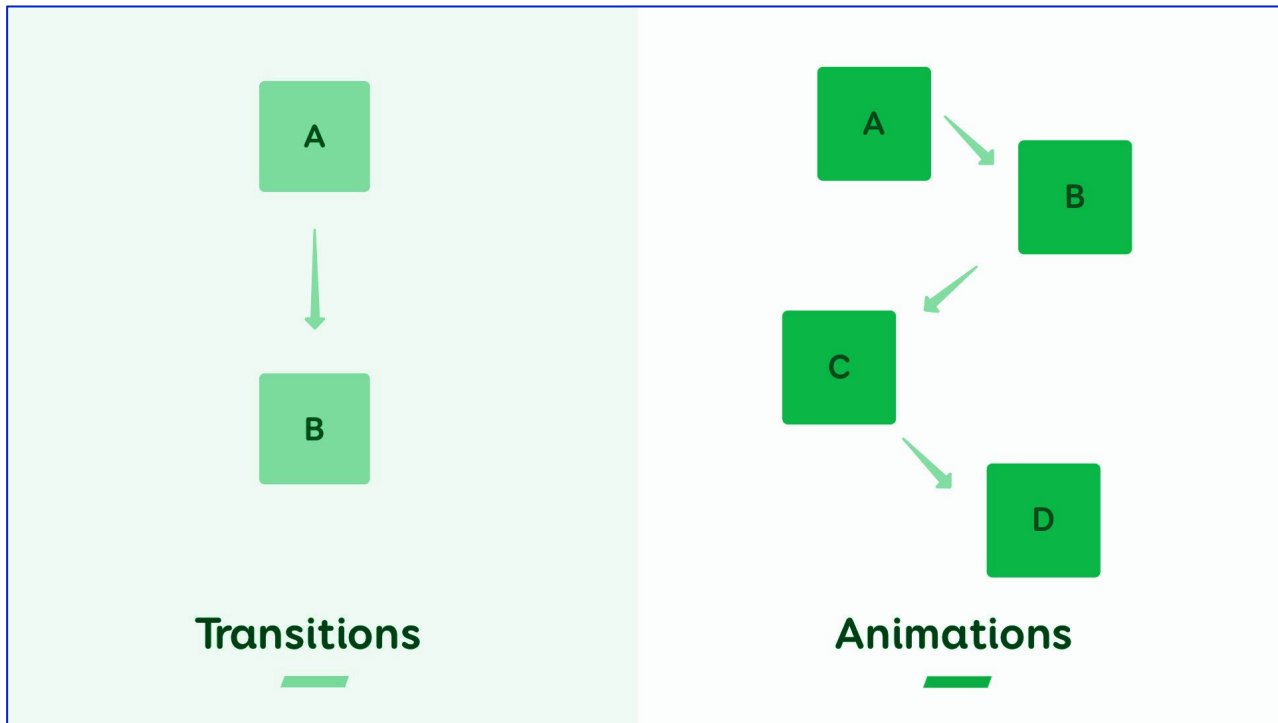
- 
- **none** — Animation styles will not be applied to the element before and after it runs.
  - **forwards** — At the end of the animation, the element retains the styles of the last keyframe.
  - **backwards** — The element will have the styles of the first keyframe before the animation starts (useful during **animation-delay**)
  - **both** — The animation will behave as if both **forwards** and **backwards** are applied simultaneously.
-

# animation-direction

## demo

- 
- **normal** – The animation plays from start to finish.
  - **reverse** – The animation plays from finish to start.
  - **alternate** – The animation alternates direction: first from start to finish, then from finish to start.
  - **alternate-reverse** – The animation alternates direction: first from finish to start, then from start to finish.
-

# Ways to Animate an Element

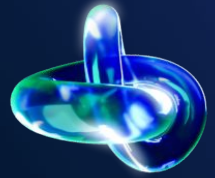




# VS

CSS Transition	CSS Animation
Triggered by <b>an event</b> (e.g., mouse hover).	Triggered by <b>an event or automatically</b> or controlled via <code>`animation-play-state`</code>
Can have <b>only two frames</b> – start and end.	Can have <b>multiple keyframes</b> .
Changes <b>one or a couple</b> of properties.	Can include <b>many</b> properties.
<b>Has a specific</b> duration (e.g., 1s, 200ms).	Can be <b>infinite</b> or have a <b>set duration</b> .
Suitable for <b>simple effects</b> like color change or movement.	Better for creating <b>complex animations</b> like moving an object along a specific path.
Applicable to any elements (except table elements and inline boxes) and pseudo-elements <b>::before</b> , <b>::after</b>	

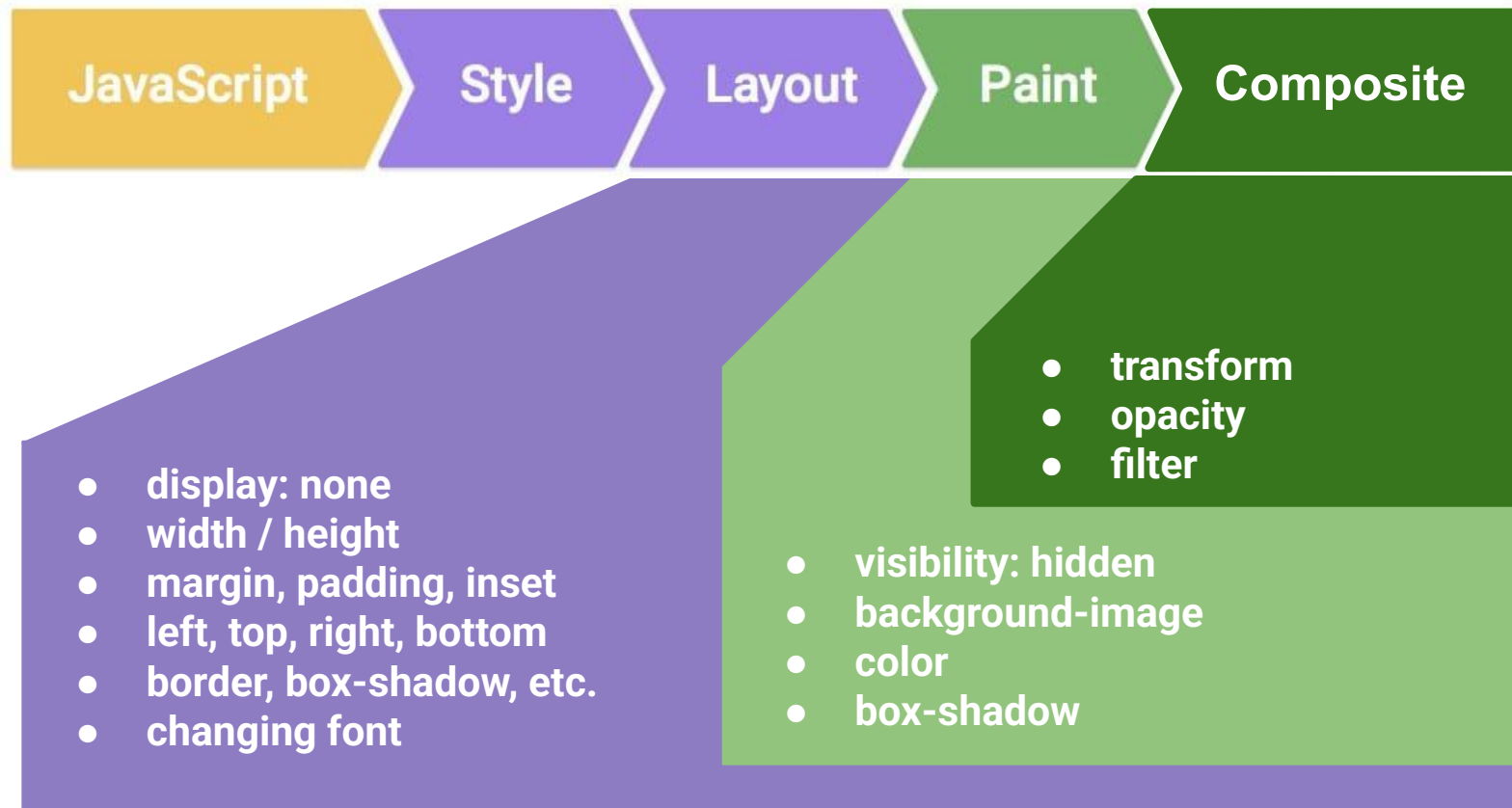
CSS Transition	CSS Animation
Simple State Changes	Complex Sequential Changes
Triggered by State Change (:hover, :focus)	Automatic Start
Limited Capabilities	Control Over Timing and Loops



---

# CSS animation performance

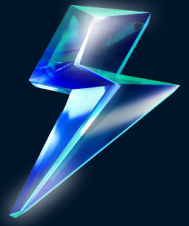
# Reflow and repaint



# What Causes the Reflows and Repaints:

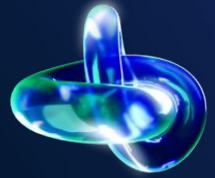
- Hiding DOM Element with **display: none** will cause both **reflow and repaint**
- Hiding DOM Element with **visibility: hidden** will cause the only **repaint**
- **Moving, animating** a DOM node will trigger **reflow and repaint**
- Activation of CSS pseudo classes such as **:hover**
- **Resizing the window** will trigger **reflow**
- Changing **font-\* properties** alters the geometry of the element will need to be a **repaint**
- Content changes, such as a user typing text in an input box
- Adding or removing Stylesheet will cause the reflow/repaint (by js)
- Reflow will happen when Adding, Removing, Updating the DOM nodes (by js)

# Properties for Changes



## CSS Triggers List – What Kind of Changes You Can Make

Cannot be Animated:	Can be Animated, but Better Not to:	Can be Animate	Better to Animate:
display position grid background-image z-index	width height margin padding flex-grow shrink left/right bottom/top Affected document flow	font-size border-width letter-spacing word-spacing	color background-color border-color transform opacity filter



---

# 3D transformation & animation

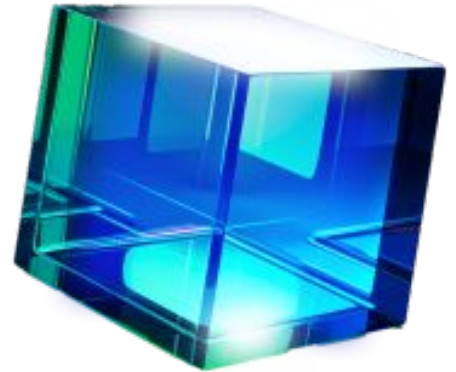
# Transform 3D

transform: translate3d / translateZ = translate: x y z;

transform: rotate3d / rotateZ = rotate: x y z;

transform: matrix3d;

transform: scale3d = scale: x y z;



example

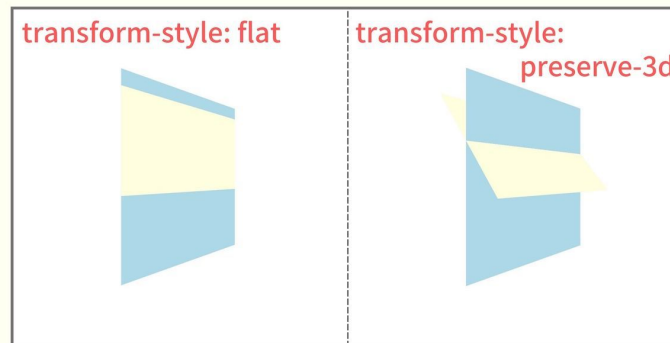


## Step 1. transform-style

– Defines whether the child elements of a block are placed in 3D space (ensures that 3D effects work).

- **flat** – Child elements of the block lie in the plane of the element itself (default value).
- **preserve-3d** – Child elements of the block should be placed in 3D space.

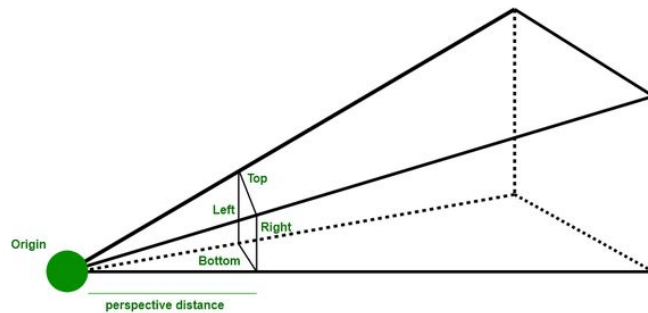
This is applied to the element being animated.



## Step 2. perspective

Defines how far the transformed object is from the viewpoint.

- Set on the **parent element** of the transformation.
- Specified in **px**.
- A **smaller** value results in a **more pronounced** 3D effect.



demo

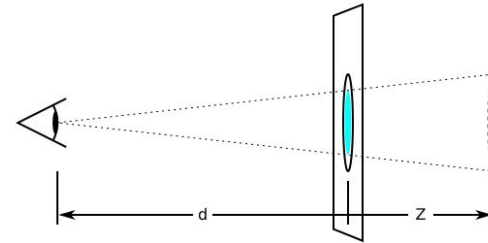
## Step 2.1 . perspective-origin

Sets the coordinates from which the object is viewed.

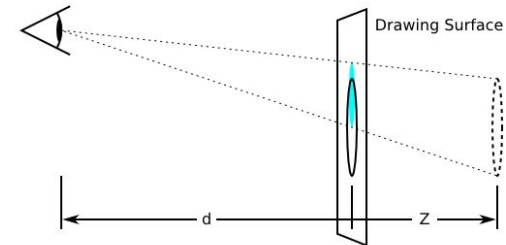
- axes x and y – Horizontal and vertical positions of the viewpoint.
- Specified in **px** and **%**.

Set along with **perspective**.

[demo](#)



*original position of the viewer's eye is at the center*



*Effect of moving the position of the viewer's eye upwards using perspective-origin*

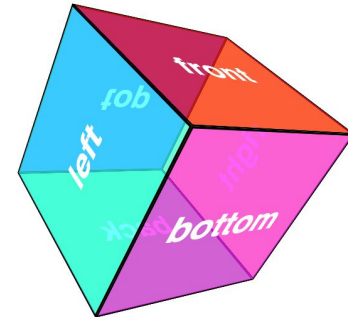
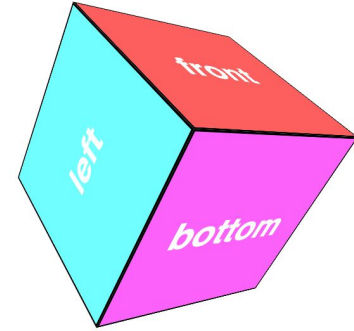
d = distance from the viewer's eye to the screen  
Z = position of the element on the z-axis

## Step 3. backface-visibility

Determines whether the back side of an element, which is not facing the user, is visible.

- Useful when an element should be **transparent**.
- Values – **visible** / **hidden**.

[demo](#)



# Create a cube

## Step 1 . `transform-style`

For element (cube)

## Step 2 . `perspective`

Parent element (wrap)

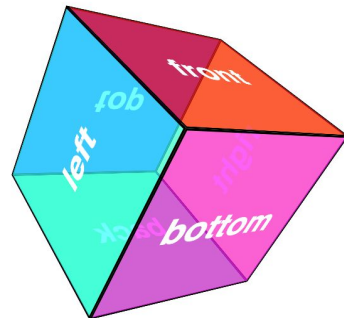
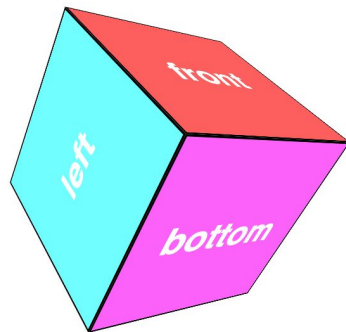
## Step 2.1 . `perspective-origin`

Parent element (wrap)

## Step 3 . `backface-visibility`

Child element (cube face)

demo



CSS animation

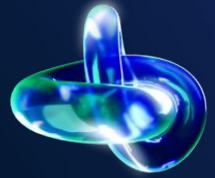
# libraries



If the animation is simple,  
sometimes it's easier to use  
a pre-made option.

[animista.net](https://animista.net)

[Animate.css](https://animate.css)



---

# SVG Animations

Demo

Demo

CSS animation

## SVG

- **SMIL**: Synchronized Multimedia Integration Language
  - Animation is described **inside the SVG** file
  - for drawing along the outline
- **CSS**
  - Changing the **color of an icon** on hover
  - circular preloaders
  - mask, clip-path, linear-gradient, filter
- **libraries**

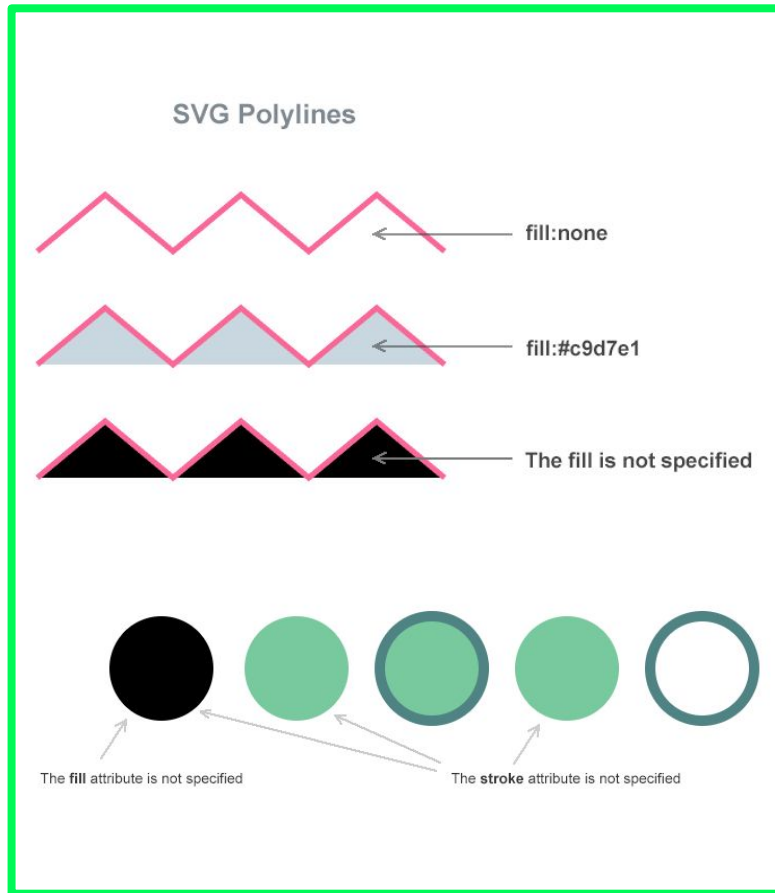


# SVG CSS

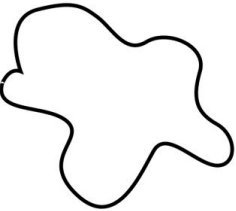
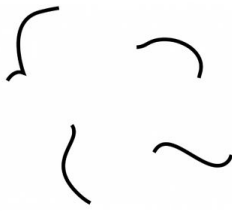
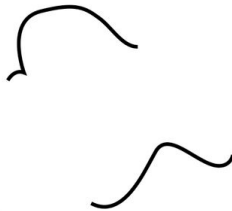
## Properties:

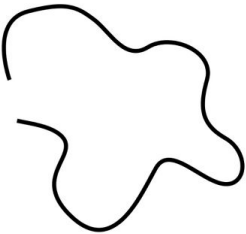
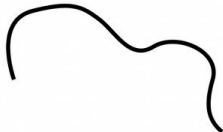

- **fill**
- **fill-opacity**
- **fill-rule**
- **stroke**
- **stroke-width**
- **stroke-opacity**

```
svg: hover { fill: green }
```



# Line Animation

원본(1000)	dasharray= 100	dasharray= 200
		

dashoffset= 150	dashoffset= 550	dashoffset= 850
		

- **stroke-dasharray** – Controls the appearance of the dashed stroke.
- **stroke-dashoffset** – Shifts the dashed stroke from its original position.

Works if the shape has a **stroke**.  
Ideal for preloaders, progress bars.

## step 1

# stroke-dasharray

- Value is an array with dash-space pairs
- If one number is specified, the dash equals the space
- The number of pairs can be unlimited and will repeat in a loop.

**stroke-dasharray:** 20; **without px**

**stroke-dasharray:** 20% 10% 30% 10%;



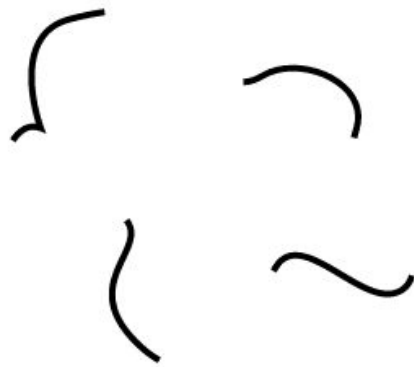
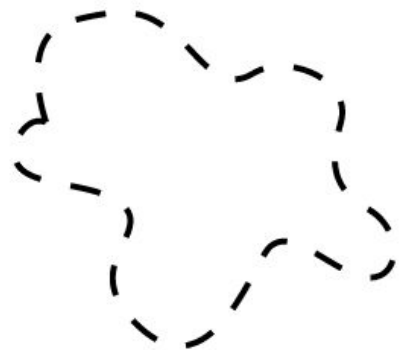
## step 2

# stroke-dashoffset

- Moves dashed lines along the contour of the object
- used for line drawing animation.

**stroke-dasharray = stroke-dashoffset = path length**

```
stroke-dashoffset: 20; without px  
stroke-dashoffset: -20;  
stroke-dashoffset: 20%;
```

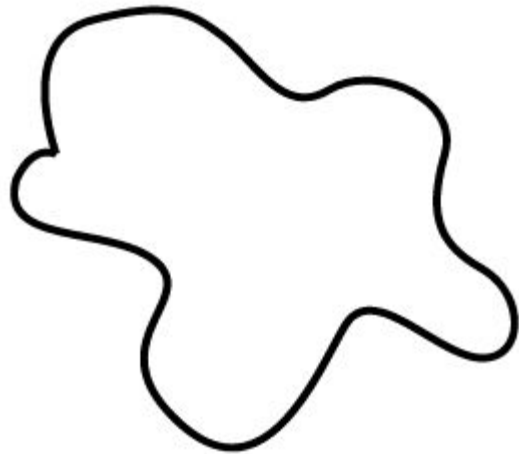


step 3

## animation @keyframe

```
.path {  
  stroke-dasharray: 1000;  
  stroke-dashoffset: 1000;  
  animation: dash 5s linear forwards;  
}  
1000 - path length  
  
@keyframes dash {  
  to {  
    stroke-dashoffset: 0;  
  }  
}
```

demo



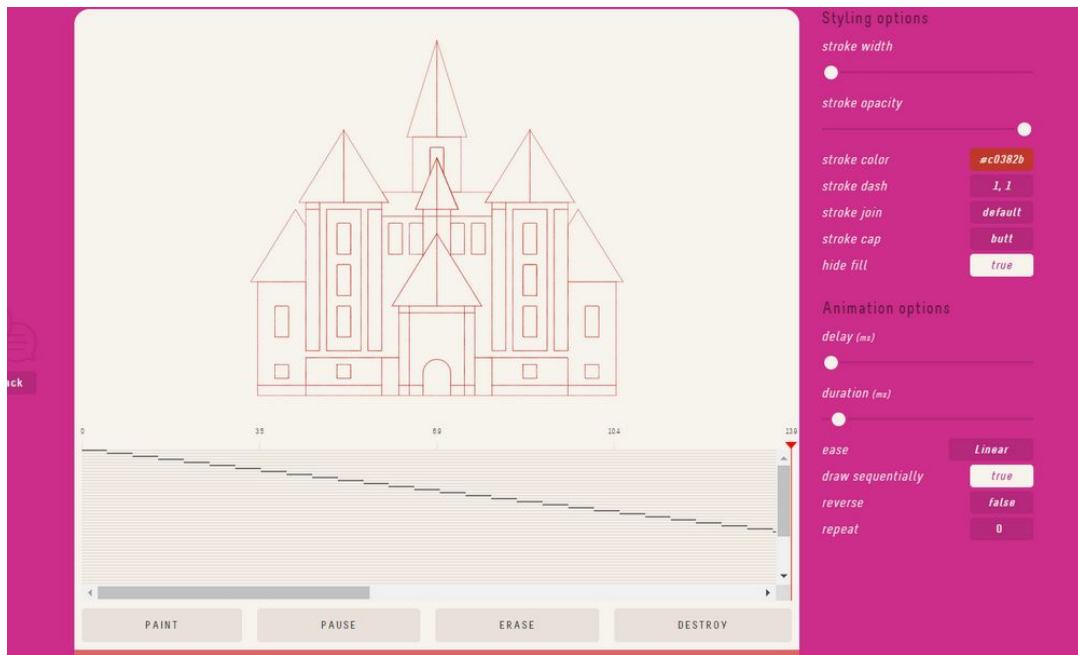
SVG

# libraries

When to Create Something  
More Complex Than a  
Single Line Animation:

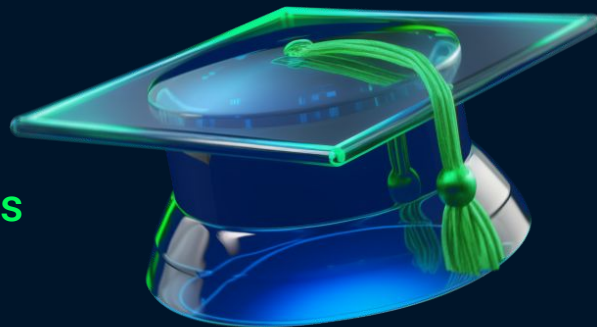
<https://www.svgator.com/>

<https://svgartista.net/>



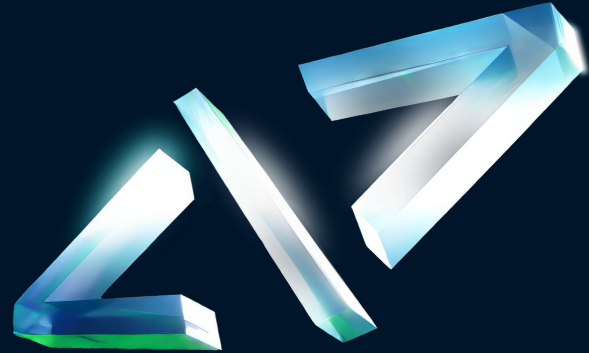
# Summary

1. Which CSS properties **can and cannot be animated**
2. How to **transform** objects
3. The differences between **animation methods**
4. How to apply **smooth transitions**
5. All animation **parameter settings**
6. How to create **3D transformations and animations**
7. How to **animate SVGs**



# Homework

1. **Add transitions** for all changing states to ensure smoothness.
2. Add animations using the **animate.css** library.





**B** Academy  
**RO**



**QUESTIONS?**

**Please fill out the feedback form**  
**It's very important for us**





**THANK YOU!**

**Have a good evening!**

# \*background-attachment

demo

How the background behaves when the block is scrolled.

**local** – The background is fixed relative to the element's content. If we scroll the block, the background scrolls with the content.

**scroll** – The background is fixed relative to the element and stays in place when the content scrolls.

**fixed** – The background is fixed relative to the viewport. Even if the element scrolls, the background doesn't move with the element.

