CSS3 Complete Comprehensive Guide

Table of Contents

---

01. Border Radius

What is Border Radius?

Border radius rounds the corners of an element's border. Before CSS3, developers had to use images or complex HTML structures to create rounded corners.

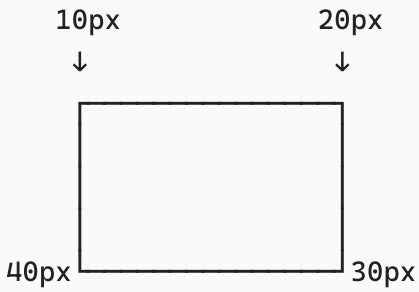
Basic Syntax

```
div {
    border-radius: 10px; /* All corners rounded equally */
}
```

"" - Individual Corner Control

```
/* border-radius: top-left top-right bottom-right bottom-left ; */
div {
    border-radius: 10px 20px 30px 40px;
}
```

Visual Representation:



Syntax Variations

1. One value (all corners):

```
border-radius: 15px;
/* All four corners = 15px */
```

2. Two values:

```
border-radius: 10px 30px;
/*
   top-left & bottom-right = 10px
   top-right & bottom-left = 30px
*/
```

3. Three values:

```
border-radius: 10px 20px 30px;
/*
   top-left = 10px
   top-right & bottom-left = 20px
   bottom-right = 30px
*/
```

4. Four values (clockwise from top-left):

```
border-radius: 10px 20px 30px 40px;
/*
   top-left = 10px
   top-right = 20px
   bottom-right = 30px
   bottom-left = 40px
*/
```

Perfect Circle - ""

```
/* border-radius: 50%; */
div {
    width: 200px;
    height: 200px;
    border-radius: 50%;
}
```

Why 50% Creates a Circle:

```
For a square:
- Width: 200px
- Height: 200px
- 50% of width = 100px (radius)
- 50% of height = 100px (radius)
Result: Perfect circle!

For a rectangle:
- Width: 300px
- Height: 200px
- 50% creates an ellipse (oval)
```

Individual Corner Properties

```
div {
    border-top-left-radius: 10px;
    border-top-right-radius: 20px;
    border-bottom-right-radius: 30px;
    border-bottom-left-radius: 40px;
}
```

Elliptical Corners

```
/* horizontal-radius / vertical-radius */
div {
    border-radius: 50px / 25px;
}

/* Individual elliptical corners */
div {
```

```
    border-top-left-radius: 50px 25px; /* width height */
}
```

Practical Examples

Pill button:

```
.button {
    padding: 10px 30px;
    border-radius: 25px;
}
```

Circle avatar:

```
.avatar {
    width: 100px;
    height: 100px;
    border-radius: 50%;
    overflow: hidden;
}
```

Semi-circle:

```
.semicircle {
    width: 200px;
    height: 100px;
    border-radius: 100px 100px 0 0;
}
```

---

02. Box Shadow

What is Box Shadow?

Box shadow adds shadow effects around an element's frame. Multiple shadows can be combined for complex effects.

Syntax - ""

```
/* box-shadow:    h    v  blur spread color    inset; */
div {
```

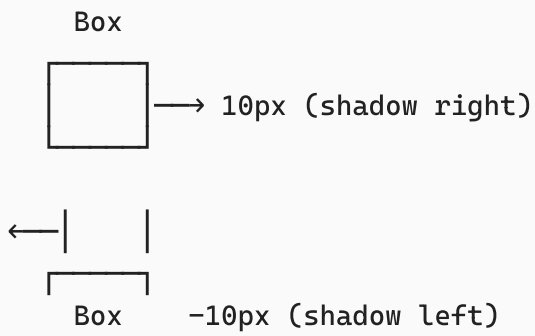
```
    box-shadow: 10px 10px 15px   0px   blue         ;
}
```

Parameters Explained in Detail

1. Horizontal Offset (h)

```
/* Positive = shadow to the right */
box-shadow: 10px 0 0 0 black;

/* Negative = shadow to the left */
box-shadow: -10px 0 0 0 black;

/* Zero = no horizontal offset */
box-shadow: 0 10px 0 0 black;
```

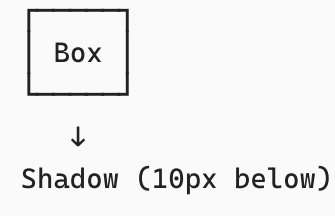
Visual:

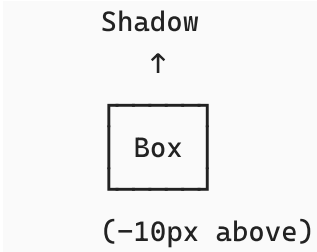


2. Vertical Offset (v)

```
/* Positive = shadow below */
box-shadow: 0 10px 0 0 black;

/* Negative = shadow above */
box-shadow: 0 -10px 0 0 black;
```

Visual:

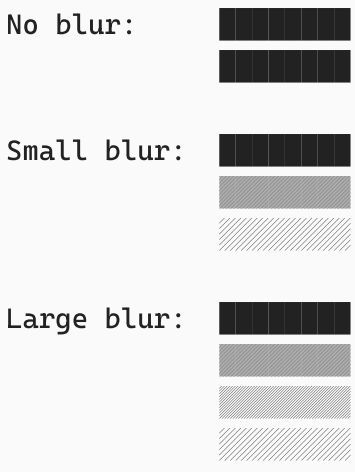




3. Blur Radius

```css
/* No blur (sharp edges) */
box-shadow: 5px 5px 0 0 black;

/* Small blur */
box-shadow: 5px 5px 5px 0 black;

/* Medium blur */
box-shadow: 5px 5px 10px 0 black;

/* Large blur (softer shadow) */
box-shadow: 5px 5px 20px 0 black;
```

Effect:

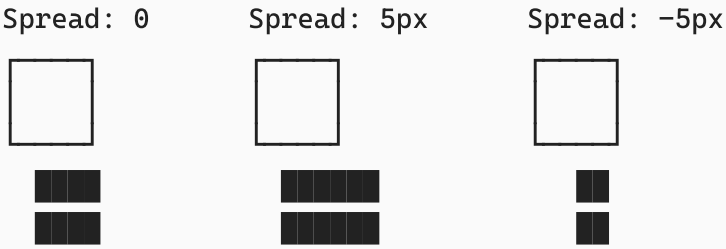


4. Spread Radius

```css
/* No spread */
box-shadow: 5px 5px 10px 0 black;

/* Positive spread (expands shadow) */
box-shadow: 5px 5px 10px 5px black;
```

```css
/* Negative spread (shrinks shadow) */
box-shadow: 5px 5px 10px -5px black;
```

Visual:



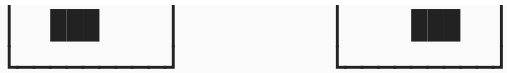
5. Color

```css
/* Named colors */
box-shadow: 5px 5px 10px 0 red;
box-shadow: 5px 5px 10px 0 blue;

/* Hex colors */
box-shadow: 5px 5px 10px 0 #333;

/* RGB */
box-shadow: 5px 5px 10px 0 rgb(0, 0, 0);

/* RGBA (with transparency) */
box-shadow: 5px 5px 10px 0 rgba(0, 0, 0, 0.5);
```

6. Inset (Optional)

```css
/* Outer shadow (default) */
box-shadow: 5px 5px 10px 0 black;

/* Inner shadow */
box-shadow: 5px 5px 10px 0 black inset;
```

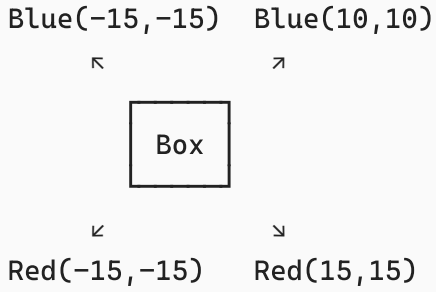
Visual:



Multiple Shadows - ""

```
box-shadow: 10px 10px 15px 0px blue,
            15px 15px 15px 0px red,
            -10px -10px 15px 0px blue,
            -15px -15px 15px 0px red;
```

Creates four shadows:



Realistic Shadow - ""

```
/* box-shadow: 5px 5px 12px 1px rgba(128, 126, 119, 1); */
div {
    box-shadow: 5px 5px 12px 1px rgba(0, 0, 0, 0.3);
}
```

Why RGBA for Shadows:

- `rgba(0, 0, 0, 1.0)` : 100% opaque (solid black)
- `rgba(0, 0, 0, 0.5)` : 50% transparent (semi-transparent)
- `rgba(0, 0, 0, 0.2)` : 80% transparent (very light shadow)

Common Shadow Patterns

1. Card elevation (Material Design):

```
.card-level-1 {
    box-shadow: 0 1px 3px rgba(0,0,0,0.12),
                0 1px 2px rgba(0,0,0,0.24);
}
```

```css
.card-level-2 {
    box-shadow: 0 3px 6px rgba(0,0,0,0.16),
                0 3px 6px rgba(0,0,0,0.23);
}

.card-level-3 {
    box-shadow: 0 10px 20px rgba(0,0,0,0.19),
                0 6px 6px rgba(0,0,0,0.23);
}
```

2. Subtle depth:

```css
div {
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}
```

3. Pressed button:

```css
button {
    box-shadow: 0 4px 6px rgba(0, 0, 0, 0.2);
}

button:active {
    box-shadow: 0 2px 3px rgba(0, 0, 0, 0.2) inset;
}
```

4. Glowing effect:

```css
.glow {
    box-shadow: 0 0 20px rgba(0, 255, 0, 0.8);
}
```

5. Neumorphism (soft UI):

```css
.neumorphic {
    box-shadow: 5px 5px 10px #d1d9e6,
                -5px -5px 10px #ffffff;
}
```

---

03. Text Shadow

What is Text Shadow?

Text shadow adds shadow effects to text content, creating depth and visual interest.

Syntax - ""

```
/* text-shadow:   h  v   blur color ; */
p {
    text-shadow: 1px 1px 2px   red;
}
```

Parameters

1. Horizontal offset:

```
text-shadow: 5px 0 0 black;  /* Right */
text-shadow: -5px 0 0 black; /* Left */
```

2. Vertical offset:

```
text-shadow: 0 5px 0 black;  /* Below */
text-shadow: 0 -5px 0 black; /* Above */
```

3. Blur radius:

```
text-shadow: 2px 2px 0 black;   /* Sharp */
text-shadow: 2px 2px 5px black;  /* Blurred */
```

4. Color:

```
text-shadow: 2px 2px 4px red;
text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.5);
```

""

```
/* text-shadow: 1px 1px 2px red; */
p {
    text-align: center;
    margin: 50px;
    text-shadow: 1px 1px 2px red;
}
```

Multiple Text Shadows

```
h1 {
    text-shadow: 1px 1px 2px red,
                 2px 2px 4px blue,
                 3px 3px 6px green;
}
```

Text Shadow Effects

1. Simple depth:

```
h1 {
    color: #333;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}
```

2. Outline effect:

```
h1 {
    color: white;
    text-shadow: -1px -1px 0 #000,
                  1px -1px 0 #000,
                 -1px  1px 0 #000,
                  1px  1px 0 #000;
}
```

3. 3D text:

```
h1 {
    color: #fff;
    text-shadow: 0 1px 0 #ccc,
                 0 2px 0 #c9c9c9,
                 0 3px 0 #bbb,
                 0 4px 0 #b9b9b9,
                 0 5px 0 #aaa,
                 0 6px 1px rgba(0,0,0,.1);
}
```

4. Neon glow:

```css
h1 {
    color: #fff;
    text-shadow: 0 0 10px #fff,
                 0 0 20px #fff,
                 0 0 30px #0ff,
                 0 0 40px #0ff,
                 0 0 50px #0ff;
}
```

5. Fire effect:

```css
h1 {
    color: #fff;
    text-shadow: 0 0 5px #ff6600,
                 0 0 10px #ff6600,
                 0 0 15px #ff6600,
                 0 0 20px #ff3300;
}
```

---

04. Box Sizing

The Box Model Problem

By default, CSS calculates element dimensions in a way that can be confusing.

Default Box Model - ""

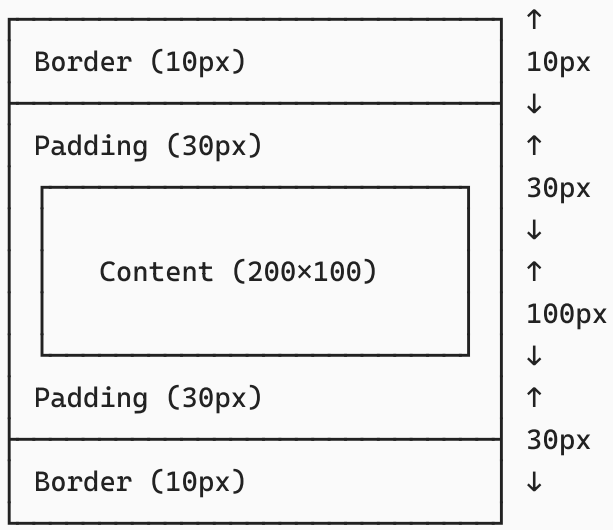
```css
/* total width : content width + padding-x + border-x */
/* total height : content height + padding-y + border-y */

div {
    width: 200px;
    height: 100px;
    padding: 30px 40px;
    border: 10px solid blue;
}
```

Calculation:

```
Content width: 200px
Padding left:  40px
```

```
Padding right: 40px
Border left:   10px
Border right:  10px
──────────────────────
Total width:   300px  ← Not what you specified!

Content height: 100px
Padding top:    30px
Padding bottom: 30px
Border top:     10px
Border bottom:  10px
──────────────────────
Total height:   180px  ← Not what you specified!
```

Visual:




```
Width: 40+200+40+20 = 300px
Height: 30+100+30+20 = 180px
```

Solution: box-sizing

```css
* {
    padding: 0;
    margin: 0;
    box-sizing: border-box;
}
```

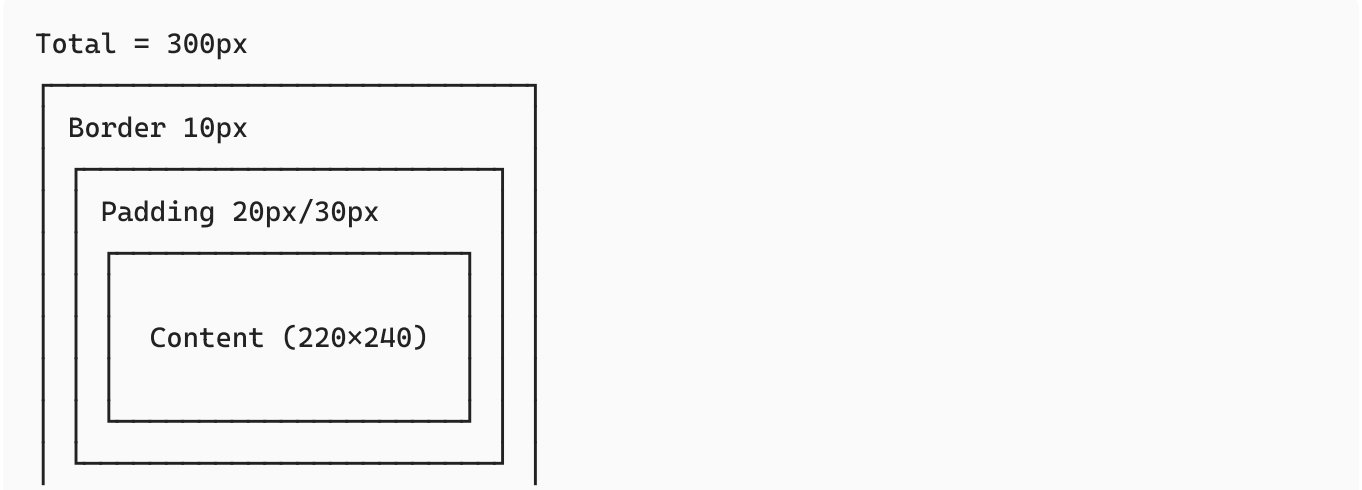
With `box-sizing: border-box`:

```
/* total width : 300 */
/* total height : 300 */
/* content width : total width - padding-x - border-x */
/* content height : total height - padding-y - border-y */

div {
    width: 300px;
    height: 300px;
    padding: 20px 30px;
    border: 10px solid blue;
    box-sizing: border-box;
}
```

Calculation:

```
Specified width: 300px
Border left:     10px
Border right:    10px
Padding left:    30px
Padding right:   30px
———————————————————
Content width:   220px  (300 - 20 - 60 = 220)

Specified height: 300px
Border top:       10px
Border bottom:    10px
Padding top:      20px
Padding bottom:   20px
———————————————————
Content height:   240px  (300 - 20 - 40 = 240)
```

Visual:



```
└──────────────────────────────┘
Width stays 300px!
```

Box Sizing Values

1. content-box (default):

```
div {
    box-sizing: content-box;
    width: 200px;
    padding: 20px;
    border: 10px solid;
}
/* Total width = 200 + 40 + 20 = 260px */
```

2. border-box (recommended):

```
div {
    box-sizing: border-box;
    width: 200px;
    padding: 20px;
    border: 10px solid;
}
/* Total width = 200px (includes padding and border) */
```

Why border-box is Better

```
/* Problem with content-box: */
.container {
    width: 100%;
    padding: 20px;
    border: 5px solid;
    /* Total width > 100% → Causes overflow! */
}

/* Solution with border-box: */
.container {
    box-sizing: border-box;
    width: 100%;
    padding: 20px;
    border: 5px solid;
    /* Total width = exactly 100% ✓ */
}
```

Best Practice

Always use this at the start of your stylesheet:

```
* {
    box-sizing: border-box;
}

/* Or more specific: */
*,
*::before,
*::after {
    box-sizing: border-box;
}
```

---

05. Transform

What is Transform?

Transform allows you to modify the appearance and position of elements without affecting the document flow. Elements can be rotated, scaled, skewed, or translated in 2D or 3D space.

"" - Transform Setup

```
div {
    width: 200px;
    height: 200px;
    padding: 20px;
    background-color: lightblue;
    margin: 200px auto;
    transition: transform 1s;
    transform-origin: bottom;
}
```

Transform Functions

1. Translate (Move)

"":

```
transform: translate(10px, -10px);
```

Individual functions:

```css
/* Move horizontally */
transform: translateX(50px);  /* Right */
transform: translateX(-50px); /* Left */

/* Move vertically */
transform: translateY(50px);  /* Down */
transform: translateY(-50px); /* Up */

/* Move in 3D */
transform: translateZ(-100px); /* Away */
transform: translateZ(100px);  /* Closer */

/* Combined */
transform: translate(50px, 100px);  /* X, Y */
transform: translate3d(50px, 100px, 20px);  /* X, Y, Z */
```

Visual:

```
Original position:
┌─────┐
│  □  │
└─────┘

translate(50px, 50px):
      ┌─────┐
      │  □  │
      └─────┘
   (moved right and down)
```

2. Scale (Resize)

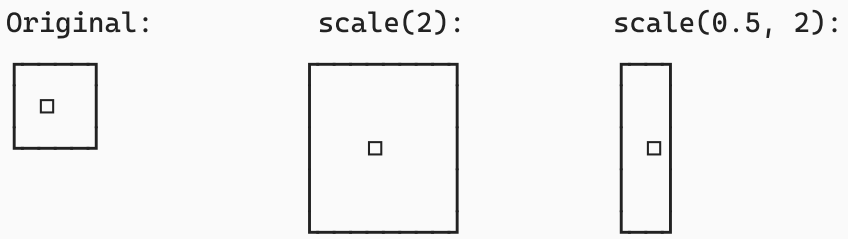
"":

```css
/* transform: scaleY(1.5, 1); */
/* scale: .5 1.5; */
/* scale: 2; */

div:hover {
    transform: scale(1.3, 1.5);
}
```

Scale functions:

```css
/* Uniform scale */
transform: scale(2);      /* 2× larger (both dimensions) */
transform: scale(0.5);    /* Half size */

/* Individual dimensions */
transform: scaleX(2);     /* 2× wider */
transform: scaleY(1.5);   /* 1.5× taller */
transform: scaleZ(2);     /* 2× depth (3D) */

/* Different X and Y */
transform: scale(2, 0.5); /* 2× wide, half tall */
```

Visual:



3. Rotate

"":

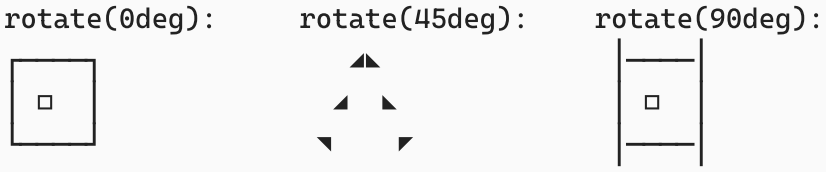
```css
/* transform: rotate(360deg); */
/* transform: rotateX(45deg); */

div:hover {
    transform: rotate(45deg);
}
```

Rotate functions:

```css
/* 2D rotation */
transform: rotate(45deg);    /* Clockwise */
transform: rotate(-45deg);   /* Counter-clockwise */

/* 3D rotation */
transform: rotateX(45deg);   /* Rotate around X-axis */
transform: rotateY(45deg);   /* Rotate around Y-axis */
transform: rotateZ(45deg);   /* Rotate around Z-axis (same as rotate) */
```

```css
/* 3D rotation with custom axis */
transform: rotate3d(1, 1, 1, 45deg);
```

Visual:



4. Skew (Slant)

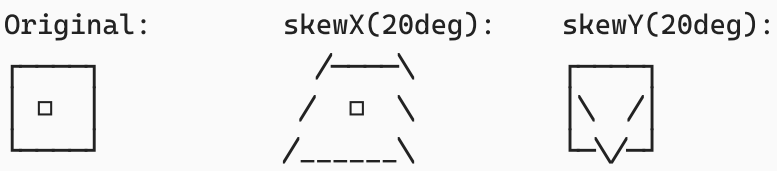
"":

```css
/* transform: skew(0deg, -20deg); */

div:hover {
    transform: skew(20deg, 10deg);
}
```

Skew functions:

```css
/* Skew horizontally */
transform: skewX(20deg);

/* Skew vertically */
transform: skewY(20deg);

/* Skew both */
transform: skew(20deg, 10deg);  /* X, Y */
```

Visual:



Combining Transforms

```css
/* transform: scale(1.3, 1.5) rotate(45deg) translate(10px, -10px); */
```

```css
div:hover {
    transform: scale(1.3, 1.5) rotate(45deg) translate(10px, -10px);
}
```

Order matters!

```css
/* These produce DIFFERENT results: */
transform: rotate(45deg) translate(100px, 0);
transform: translate(100px, 0) rotate(45deg);
```

Visual explanation:

```
rotate then translate:
1. Rotate box 45°      → │/
2. Move along new axis → │/  (moves diagonally)

translate then rotate:
1. Move right 100px    → □——→
2. Rotate 45°          →    ◢◣
```

Transform Origin

"":

```css
transform-origin: bottom;
```

What is transform-origin?
The point around which transformations occur.

```css
/* Keywords */
transform-origin: center;        /* Default */
transform-origin: top;
transform-origin: bottom;
transform-origin: left;
transform-origin: right;
transform-origin: top left;
transform-origin: bottom right;

/* Coordinates */
transform-origin: 50px 50px;
transform-origin: 50% 50%;       /* Same as center */
```

```
/* Three values (3D) */
transform-origin: 50% 50% 100px;
```

Visual:

```
transform-origin: center (default):
     ┌─────┐
     │  ●  │   ← Rotates around center
     └─────┘

transform-origin: top left:
  ●  ┌─────┐
     │     │   ← Rotates around top-left corner
     └─────┘

transform-origin: bottom:
     ┌─────┐
     │     │   ← Rotates around bottom center
     └──●──┘
```

3D Transforms

"":

```
/* transform: perspective(1000px) translateZ(-100px); */

body {
    perspective: 1000px;
}

div:hover {
    transform: rotateX(45deg);
}
```

Perspective explained:

```
/* On parent element */
.parent {
    perspective: 1000px;  /* Distance from viewer */
}

/* Or on element itself */
.element {
```

```
    transform: perspective(1000px) rotateY(45deg);
}
```

Perspective values:

```
perspective: 500px;   /* Strong 3D effect (close) */
perspective: 1000px;  /* Medium 3D effect */
perspective: 2000px;  /* Subtle 3D effect (far) */
```

Visual:

```
No perspective:          With perspective:

┌──────────┐             ┌──────────┐
│          │             │\        /│
│   Box    │             │ \      / │  ← Appears 3D
│          │             │  \    /  │
└──────────┘             │   Box    │
                         └──────────┘
```

Matrix Transform - ""

```
/* scaleX, skewY, skewX, scaleY, translateX, translateY */
/* transform: matrix(2, 0, .3, 1.5, 0, 100); */

div:hover {
    transform: matrix(2, 0, 0.3, 1.5, 0, 100);
}
```

Matrix syntax:

```
transform: matrix(a, b, c, d, tx, ty);
```

Parameters:

- `a` = scaleX
- `b` = skewY
- `c` = skewX
- `d` = scaleY
- `tx` = translateX
- `ty` = translateY

Example breakdown:

```
matrix(2, 0, 0.3, 1.5, 0, 100)

= scaleX(2)          /* 2× wider */
  skewY(0)           /* No Y skew */
  skewX(0.3)         /* Slight X skew */
  scaleY(1.5)        /* 1.5× taller */
  translateX(0)      /* No X movement */
  translateY(100)    /* Move 100px down */
```

---

06. Transition

What is Transition?

Transition creates smooth animations when CSS properties change. Instead of instant changes, values gradually transform over time.

Basic Transition

```
div {
    width: 200px;
    height: 50px;
    padding: 20px;
    background-color: lightblue;
    transition-property: all;
    transition-duration: 2s;
}

div:hover {
    width: 400px;
    height: 100px;
}
```

Without transition:

```
Hover: width instantly jumps from 200px → 400px
```

With transition:

```
Hover: width smoothly grows from 200px → 400px over 2 seconds
```

Transition Properties

1. transition-property

Specifies which properties to animate:

```
/* Animate all properties */
transition-property: all;

/* Animate specific properties */
transition-property: width;
transition-property: width, height;
transition-property: width, height, background-color;

/* Don't animate anything */
transition-property: none;
```

Example:

```
div {
    width: 200px;
    background-color: blue;
    transition-property: width;  /* Only animate width */
}

div:hover {
    width: 400px;          /* ✓ Animates smoothly */
    background-color: red; /* ✗ Changes instantly */
}
```

2. transition-duration

How long the transition takes:

```
transition-duration: 2s;    /* 2 seconds */
transition-duration: 1s;    /* 1 second */
transition-duration: 500ms; /* 0.5 seconds */

/* Different durations for different properties */
transition-property: width, height;
transition-duration: 2s, 1s;  /* width: 2s, height: 1s */
```

3. transition-timing-function

Complete Example:

```
.parent {
    background-color: lightgoldenrodyellow;
    padding: 30px;
}

.parent div {
    background-color: lightseagreen;
    padding: 30px;
    margin: 10px 0px;
    width: 200px;
    transition-property: width;
    transition-duration: 2s;
}

.parent:hover div {
    width: 400px;
}

.linear {
    transition-timing-function: linear;
}

.ease {
    transition-timing-function: ease;
}

.ease-in {
    transition-timing-function: ease-in;
}

.ease-out {
    transition-timing-function: ease-out;
}

.ease-in-out {
    transition-timing-function: ease-in-out;
}
```

Timing functions explained:

1. linear:

```
Speed: ████████████████  (constant speed)
Start ──────────────────→ End
```

Moves at constant speed from start to end.

2. ease (default):

```
Speed: ─▁▂▃▅▆▇▇▆▅▃▂▁─
       slow fast  slow
```

Starts slow, speeds up, then slows down at end.

3. ease-in:

```
Speed: ──▁▂▃▅▆▇▇▇▇
       slow    fast
```

Starts slow, accelerates to end.

4. ease-out:

```
Speed: ▇▇▇▇▇▇▇▆▅▃▂▁──
       fast    slow
```

Starts fast, decelerates to end.

5. ease-in-out:

```
Speed: ──▁▂▅▆▇▇▆▅▂▁──
       slow fast slow
```

Starts slow, speeds up in middle, slows at end.

Custom cubic-bezier:

```
transition-timing-function: cubic-bezier(0.42, 0, 0.58, 1);
```

Visual comparison:

```
Time:    0%           50%           100%
linear:  ●━━━━━━━━━━━━●━━━━━━━━━━━━●
ease:    ●━━━━━━━━●━━━━━━━━━━━●━━━━●
```

4. transition-delay

Wait before starting transition:

```
transition-delay: 0s;    /* Start immediately */
transition-delay: 1s;    /* Wait 1 second */
transition-delay: 500ms; /* Wait 0.5 seconds */
```

Example:

```
div {
    transition-property: opacity;
    transition-duration: 1s;
    transition-delay: 0.5s;
}

div:hover {
    opacity: 0;
}
/*
Timeline:
0s - 0.5s: Nothing happens (delay)
0.5s - 1.5s: Opacity fades (duration)
*/
```

Shorthand

```
/* transition: width 2s ease; */

/* Longhand: */
transition-property: width;
transition-duration: 2s;
transition-timing-function: ease;
transition-delay: 0s;

/* Shorthand: */
transition: width 2s ease;

/* Multiple properties: */
transition: width 2s ease,
            height 1s linear,
```

```
            background-color 0.5s;

/* All properties: */
transition: all 0.3s ease;
```

Transition All vs Specific

```
/* Animate all changes */
.element {
    transition: all 0.3s ease;
}

/* Better performance - specify properties */
.element {
    transition: transform 0.3s ease,
                opacity 0.3s ease;
}
```

Why specify properties?

- Better performance
- More control
- Avoid unintended animations

Transitionable Properties

Can transition:

- Colors: `color`, `background-color`, `border-color`
- Dimensions: `width`, `height`, `padding`, `margin`
- Position: `top`, `right`, `bottom`, `left`
- Transform: `transform`
- Opacity: `opacity`
- Many more...

Cannot transition:

- `display`
- `font-family`
- `position` (absolute, relative, etc.)

Common Use Cases

1. Hover effect:

```
.button {
    background-color: blue;
    transition: background-color 0.3s ease;
}

.button:hover {
    background-color: darkblue;
}
```

2. Modal fade-in:

```
.modal {
    opacity: 0;
    transition: opacity 0.5s ease;
}

.modal.show {
    opacity: 1;
}
```

3. Smooth resizing:

```
.box {
    width: 100px;
    height: 100px;
    transition: width 0.3s, height 0.3s;
}

.box:hover {
    width: 200px;
    height: 200px;
}
```

4. Transform with transition:

```
.card {
    transform: scale(1);
    transition: transform 0.3s ease;
}

.card:hover {
```

```
    transform: scale(1.1);
}
```

---

07. Animation

What is Animation?

CSS Animations allow you to create complex, multi-step animations without JavaScript. Unlike transitions (which need a trigger), animations can run automatically and loop.

Basic Animation

```
div {
    width: 50px;
    height: 50px;
    background-color: lightgreen;
    position: absolute;
    top: 5%;
    left: 5%;
    border-radius: 5px;

    /* Shorthand */
    animation: move 5s ease 2s 1 both;
}

div:hover {
    animation-play-state: paused;
}
```

Animation Properties

1. animation-name

```
animation-name: move;
```

Links to the @keyframes rule name.

2. animation-duration

```
animation-duration: 5s;
```

How long one cycle takes.

```
animation-duration: 2s;    /* 2 seconds */
animation-duration: 500ms; /* 0.5 seconds */
animation-duration: 0.5s;  /* Same as 500ms */
```

3. animation-iteration-count

```
animation-iteration-count: 3;  /* Run 3 times */
```

```
animation-iteration-count: 1;        /* Once (default) */
animation-iteration-count: 5;        /* 5 times */
animation-iteration-count: infinite; /* Forever */
```

4. animation-timing-function

```
animation-timing-function: ease;
```

Same values as transition-timing-function:

- `linear`
- `ease`
- `ease-in`
- `ease-out`
- `ease-in-out`
- `cubic-bezier(x1, y1, x2, y2)`

5. animation-delay

```
animation-delay: 2s;  /* Wait 2 seconds before starting */
```

6. animation-direction

```
animation-direction: normal;
```

Values:

```
animation-direction: normal;             /* 0% → 100% */
animation-direction: reverse;            /* 100% → 0% */
```

```
animation-direction: alternate;          /* 0%→100%, 100%→0%, 0%→100%... */
animation-direction: alternate-reverse;  /* 100%→0%, 0%→100%, 100%→0%... */
```

Visual:

```
normal:             A ──────────→ B
reverse:            A ←────────── B
alternate:          A ──────────→ B
 (2 iterations)       ←────────── A
alternate-reverse: A ←────────── B
                     ──────────→ A
```

7. animation-fill-mode

```
animation: move 5s ease 2s 1 both;
/*                            ↑
                          fill-mode */
```

Values:

```
animation-fill-mode: none;      /* No styles applied before/after */
animation-fill-mode: forwards;  /* Keep final state */
animation-fill-mode: backwards; /* Apply first state during delay */
animation-fill-mode: both;      /* Both forwards and backwards */
```

Example:

```
@keyframes fadeOut {
    from { opacity: 1; }
    to { opacity: 0; }
}

.element {
    animation: fadeOut 1s forwards;
    /* Without 'forwards': opacity returns to 1 after animation
       With 'forwards': opacity stays at 0 */
}
```

8. animation-play-state

```
div:hover {
    animation-play-state: paused;
```

```
}
```

Values:

```
animation-play-state: running;  /* Playing (default) */
animation-play-state: paused;   /* Paused */
```

Use case:

```
.loader {
    animation: spin 2s linear infinite;
}

.loader:hover {
    animation-play-state: paused;  /* Pause on hover */
}
```

Keyframes

Simple Animation

```
@keyframes move {
    0% {
        transform: translateX(0px);
    }

    50% {
        transform: translateX(600px);
    }

    100% {
        transform: translateX(0px);
    }
}
```

Timeline:

```
0%    25%   50%   75%   100%
|─────|─────|─────|─────|
○           ○           ○
Start    Middle      Back
```

Complex Animation

```
@keyframes move {
    0% {
        rotate: 0deg;
        top: 5%;
        left: 5%;
        background-color: lightcoral;
    }

    25% {
        top: 5%;
        left: 90%;
        background-color: lightgoldenrodyellow;
        rotate: 90deg;
    }

    50% {
        top: 90%;
        left: 90%;
        background-color: lightpink;
        rotate: 180deg;
    }

    75% {
        left: 5%;
        top: 90%;
        background-color: lightseagreen;
        rotate: 270deg;
    }

    100% {
        top: 5%;
        left: 5%;
        background-color: lightskyblue;
        rotate: 360deg;
    }
}
```

This creates a box that:

1. Starts at top-left, red, 0° rotation
2. Moves to top-right, yellow, 90° rotation
3. Moves to bottom-right, pink, 180° rotation

4. Moves to bottom-left, teal, 270° rotation
5. Returns to top-left, blue, 360° rotation

Visual:

```
┌─────────────────────┐
│ 0%      →   25%     │  (red → yellow)
│ ↓           ↓       │
│ 100%    ←   50%     │  (blue ← pink)
└─────────────────────┘
      ↑
   75% (teal)
```

Keyframe Syntaxes

1. Percentage-based:

```css
@keyframes slide {
    0% { left: 0; }
    25% { left: 25%; }
    50% { left: 50%; }
    75% { left: 75%; }
    100% { left: 100%; }
}
```

2. from/to:

```css
@keyframes fadeIn {
    from { opacity: 0; }
    to { opacity: 1; }
}
/* Equivalent to:
   0% { opacity: 0; }
   100% { opacity: 1; }
*/
```

3. Multiple stops at same percentage:

```css
@keyframes blink {
    0%, 100% {
        opacity: 1;
    }
    50% {
```

```
        opacity: 0;
    }
}
```

Shorthand Syntax - ""

```
/* animation: move 5s ease 2s 1 both; */

/* Expanded: */
animation-name: move;
animation-duration: 5s;
animation-timing-function: ease;
animation-delay: 2s;
animation-iteration-count: 1;
animation-fill-mode: both;

/* Shorthand order:
   name duration timing-function delay iteration-count fill-mode
*/
```

More shorthand examples:

```
/* All properties */
animation: move 2s ease-in-out 1s infinite alternate both;

/* Minimal */
animation: move 2s;

/* Multiple animations */
animation: fadeIn 1s, slideUp 0.5s 0.5s;
```

Common Animation Patterns

1. Spin/Rotate:

```
@keyframes spin {
    from { transform: rotate(0deg); }
    to { transform: rotate(360deg); }
}

.loader {
    animation: spin 2s linear infinite;
}
```

2. Pulse:

```
@keyframes pulse {
    0%, 100% {
        transform: scale(1);
        opacity: 1;
    }
    50% {
        transform: scale(1.1);
        opacity: 0.8;
    }
}

.button {
    animation: pulse 2s ease-in-out infinite;
}
```

3. Bounce:

```
@keyframes bounce {
    0%, 20%, 50%, 80%, 100% {
        transform: translateY(0);
    }
    40% {
        transform: translateY(-30px);
    }
    60% {
        transform: translateY(-15px);
    }
}

.ball {
    animation: bounce 2s infinite;
}
```

4. Shake:

```
@keyframes shake {
    0%, 100% { transform: translateX(0); }
    10%, 30%, 50%, 70%, 90% { transform: translateX(-10px); }
    20%, 40%, 60%, 80% { transform: translateX(10px); }
}

.error {
```

```
    animation: shake 0.5s;
}
```

5. Fade in/out:

```
@keyframes fadeIn {
    from {
        opacity: 0;
        transform: translateY(20px);
    }
    to {
        opacity: 1;
        transform: translateY(0);
    }
}

.element {
    animation: fadeIn 0.5s ease-out;
}
```

6. Typing effect:

```
@keyframes typing {
    from { width: 0; }
    to { width: 100%; }
}

@keyframes blink {
    50% { border-color: transparent; }
}

.typewriter {
    overflow: hidden;
    border-right: 2px solid;
    white-space: nowrap;
    animation: typing 3s steps(30) 1s both,
               blink 0.5s step-end infinite;
}
```

Performance Tips

Animate these (GPU accelerated):

```css
/* ✓ Good performance */
transform: translateX(100px);
transform: scale(1.5);
transform: rotate(45deg);
opacity: 0.5;
```

Avoid animating these:

```css
/* ✗ Poor performance */
width: 500px;
height: 300px;
margin-left: 100px;
top: 50px;
```

Why?

- `transform` and `opacity` don't trigger layout recalculation
- Other properties force browser to recalculate entire page layout

---

08. CSS Selectors

Selector Types

```
/*
  * :  all elements
  tagName :  p, div, ...
  class : .classname
  id :    #idname
  space :  .parent div
  > :   direct child
  + :   adjacent sibling
  ~ :   general sibling
  [] :  attribute
     *= : contains
     ^= : starts with
     $= : ends with
*/
```

Basic Selectors

1. Universal Selector (*)

```css
* {
    margin: 0;
    padding: 0;
}
```

Selects **all elements**.

2. Type Selector

```css
p {
    color: blue;
}

div {
    background-color: lightgreen;
}
```

Selects all elements of that type.

3. Class Selector

```css
.classname {
    color: red;
}
```

Selects all elements with `class="classname"`.

4. ID Selector

```css
#idname {
    font-size: 20px;
}
```

Selects element with `id="idname"`.

Combinators

1. Descendant Selector (space)

```css
/* .parent div */
.parent div {
```

```
    background-color: lightgreen;
}
```

HTML:

```
<div class="parent">
    <div>Selected ✓</div>
    <section>
        <div>Also selected ✓</div>
    </section>
</div>
```

Selects **all descendants** (children, grandchildren, etc.).

2. Child Selector (>)

```
/* .parent>p */
.parent > p {
    background-color: lightgreen;
    border: 2px solid #000;
    padding: 20px;
}
```

HTML:

```
<div class="parent">
    <p>Selected ✓</p>
    <section>
        <p>NOT selected ✗</p>
    </section>
</div>
```

Selects only **direct children**.

Difference:

```
<div class="parent">
    <p>Child</p>          ← .parent>p ✓   .parent p ✓
    <div>
        <p>Grandchild</p> ← .parent>p ✗   .parent p ✓
    </div>
</div>
```

3. Adjacent Sibling Selector (+)

```
/* div+p */
div + p {
    background-color: lightgreen;
    border: 2px solid #000;
    padding: 20px;
}
```

HTML:

```
<div>Div 1</div>
<p>Selected ✓</p>      ← Immediately after div
<p>NOT selected ✗</p>  ← Not immediately after div
```

Selects element **immediately following** another.

4. General Sibling Selector (~)

```
/* div~p */
div ~ p {
    background-color: lightgreen;
    border: 2px solid #000;
    padding: 20px;
}
```

HTML:

```
<div>Div 1</div>
<span>Span 1</span>
<p>Selected ✓</p>     ← After div (doesn't need to be immediate)
<p>Also selected ✓</p>
```

Selects **all siblings** after element.

Comparison (+ vs ~):

```
<div>Div</div>
<span>Span</span>
<p>P1</p>
<p>P2</p>
```

```
div + p:  Select P1 only ✓
div ~ p:  Select P1 ✓ and P2 ✓
```

Attribute Selectors

Basic Attribute Selector

```
/* [width] */
[width] {
    background-color: lightgreen;
}
```

HTML:

```
<p width="50">Has width attribute ✓</p>
<p>No width attribute X</p>
```

Attribute with Exact Value

```
[width="50"] {
    background-color: lightgreen;
}
```

HTML:

```
<p width="50">width="50" ✓</p>
<p width="100">width="100" X</p>
```

Contains (*=) - ""

```
/* [width*="5"] */
[width*="5"] {
    background-color: lightgreen;
}
```

HTML:

```
<p width="50">Contains '5' ✓</p>
<p width="55">Contains '5' ✓</p>
<p width="150">Contains '5' ✓</p>
```

```
<p width="105">Contains '5' ✓</p>
<p width="200">No '5' ✗</p>
```

Starts With (^=) - ""

```
/* [width^="5"] */
[width^="5"] {
    background-color: lightgreen;
}
```

HTML:

```
<p width="50">Starts with '5' ✓</p>
<p width="55">Starts with '5' ✓</p>
<p width="150">Starts with '1' ✗</p>
<p width="500">Starts with '5' ✓</p>
```

Ends With ($=) - ""

```
/* [width$="5"] */
[width$="5"] {
    background-color: lightgreen;
}
```

HTML:

```
<p width="50">Ends with '0' ✗</p>
<p width="55">Ends with '5' ✓</p>
<p width="105">Ends with '5' ✓</p>
<p width="200">Ends with '0' ✗</p>
```

Other Attribute Selectors

Whitespace-separated list (|=):

```
[lang|="en"] {
    color: blue;
}
```

Matches `en`, `en-US`, `en-GB`, etc.

Space-separated list (~=):

```
[class~="highlight"] {
    background-color: yellow;
}
```

Matches any element with "highlight" in class list.

Grouping Selectors

```
h1, h2, h3 {
    color: blue;
}

.class1, .class2, #id1 {
    font-size: 20px;
}
```

---

09. Pseudo-Classes

Pseudo-Class List

```
/*
  pseudo-class: (:)
      - hover
      - active
      - visited
      - link
      - empty
      - disabled
      - enabled
      - focus
      - first-child
      - last-child
      - nth-child
      - only-child
      - only-of-type
      - nth-last-child
      - first-of-type
      - last-of-type
      - nth-of-type
      - nth-last-of-type
```

```
      - not
*/
```

Interactive Pseudo-Classes

:hover

```
div:hover {
    transform: scale(1.5);
}
```

Applies when user hovers over element.

```
button:hover {
    background-color: darkblue;
}

a:hover {
    text-decoration: underline;
}
```

:active**

```
div:active {
    background-color: lightgreen;
    border: 2px solid #000;
    padding: 20px;
}
```

Applies while element is being clicked/activated.

```
button:active {
    transform: scale(0.95);  /* Pressed effect */
}
```

:focus

```
input:focus {
    border: 4px solid blue;
}
```

Applies when element has focus (clicked/tabbed to).

```
input:focus {
    outline: 2px solid blue;
    box-shadow: 0 0 5px rgba(0,0,255,0.5);
}

textarea:focus {
    border-color: green;
}
```

Link Pseudo-Classes

```
a:visited {
    color: lightcoral;
}

a:link {
    color: lightgreen;
}
```

All link states:

```
/* Unvisited links */
a:link {
    color: blue;
}

/* Visited links */
a:visited {
    color: purple;
}

/* Hovered links */
a:hover {
    color: red;
}

/* Active (being clicked) */
a:active {
    color: orange;
}
```

IMPORTANT: Order matters (LVHA):

```
/* Correct order: */
a:link { }
a:visited { }
a:hover { }
a:active { }

/* Wrong order causes issues! */
```

Form Pseudo-Classes

:disabled / :enabled

```
input:disabled {
    opacity: 0.5;
    cursor: not-allowed;
}

input:enabled {
    cursor: text;
}
```

HTML:

```
<input type="text" disabled>  <!-- :disabled applies -->
<input type="text">           <!-- :enabled applies -->
```

:checked

```
input[type="checkbox"]:checked {
    background-color: green;
}

input[type="radio"]:checked + label {
    font-weight: bold;
}
```

:required / :optional

```
input:required {
    border: 2px solid red;
}
```

```
input:optional {
    border: 2px solid gray;
}
```

:valid / :invalid

```
input:valid {
    border-color: green;
}

input:invalid {
    border-color: red;
}
```

Structural Pseudo-Classes

:first-child

```
/* ul :first-child */
ul :first-child {
    color: lightgreen;
}

/* vs */

/* ul:first-child */
ul:first-child {
    color: lightgreen;
}
```

Difference:

```
<ul>
    <li>Item 1</li> ← ul :first-child ✓ (first child of ul)
    <li>Item 2</li>
</ul>
<ul>              ← ul:first-child ✗ (ul is not first child)
    <li>Item 1</li>
</ul>
```

:last-child

```css
li:last-child {
    border-bottom: none;
}
```

HTML:

```html
<ul>
    <li>Item 1</li>
    <li>Item 2</li>
    <li>Last item ✓</li>
</ul>
```

:nth-child()

```css
/* ul :nth-child(2n+4) */
ul :nth-child(2n+4) {
    color: lightgreen;
}
```

Formula: an + b

- `a` = step size
- `n` = counter (0, 1, 2, 3, ...)
- `b` = offset

Examples:

```css
/* Even children (2, 4, 6, 8...) */
:nth-child(2n) {
    background-color: lightblue;
}

/* Odd children (1, 3, 5, 7...) */
:nth-child(2n+1) {
    background-color: lightpink;
}

/* Every third child starting from 1 (1, 4, 7, 10...) */
:nth-child(3n+1) {
    font-weight: bold;
}

/* First 3 children */
```

```
:nth-child(-n+3) {
    color: red;
}

/* Keywords */
:nth-child(odd)   /* Same as 2n+1 */
:nth-child(even)  /* Same as 2n */
```

Visual:

```
<ul>
    <li>1</li>  ← 2n+4: 2(0)+4 = 4 ✗
    <li>2</li>  ← 2n+4: 2(0)+4 = 4 ✗
    <li>3</li>  ← 2n+4: 2(0)+4 = 4 ✗
    <li>4</li>  ← 2n+4: 2(0)+4 = 4 ✓
    <li>5</li>  ← 2n+4: 2(1)+4 = 6 ✗
    <li>6</li>  ← 2n+4: 2(1)+4 = 6 ✓
    <li>7</li>
    <li>8</li>  ← 2n+4: 2(2)+4 = 8 ✓
    <li>9</li>
</ul>
```

:nth-last-child()

Same as nth-child but counts from the end.

```
li:nth-last-child(2) {
    color: red;  /* Second from last */
}
```

:only-child

```
p:only-child {
    font-weight: bold;
}
```

HTML:

```
<div>
    <p>Only child ✓</p>
</div>

<div>
```

```
    <p>Not only child X</p>
    <p>Not only child X</p>
</div>
```

Type-Specific Structural Pseudo-Classes

:first-of-type

```
/* div :first-child */
div :first-child {
    color: lightgreen;
}

/* div p:first-of-type */
div p:first-of-type {
    color: lightgreen;
}
```

Difference:

```
<div>
    <span>Span 1</span> ← :first-child ✓
    <p>P 1</p>          ← :first-child X, p:first-of-type ✓
    <p>P 2</p>
</div>
```

:last-of-type

```
p:last-of-type {
    margin-bottom: 0;
}
```

:nth-of-type()

```
/* Every second paragraph */
p:nth-of-type(2n) {
    background-color: lightblue;
}
```

:only-of-type

```css
/* div p:only-of-type */
div p:only-of-type {
    color: lightgreen;
}
```

HTML:

```html
<div>
    <span>Span 1</span>
    <p>Only paragraph ✓</p>
    <span>Span 2</span>
</div>

<div>
    <p>Not only ✗</p>
    <p>Not only ✗</p>
</div>
```

:not() - ""

```css
/* div :not(span) */
div :not(span) {
    color: lightgreen;
}
```

HTML:

```html
<div>
    <span>Span (excluded) ✗</span>
    <p>P (selected) ✓</p>
    <div>Div (selected) ✓</div>
</div>
```

Multiple exclusions:

```css
/* Select all except span and div */
:not(span):not(div) {
    color: blue;
}

/* CSS4 (newer browsers) */
:not(span, div) {
```

```
    color: blue;
}
```

:empty

```
div:empty {
    display: none;
}
```

HTML:

```
<div></div>               ← :empty ✓
<div>Text</div>           ← :empty ✗
<div> </div>              ← :empty ✗ (whitespace counts!)
<div><!-- comment --></div> ← :empty ✓ (comments don't count)
```

---

10. Pseudo-Elements

"" - Pseudo-Element List

```
/*
  pseudo-elements (::)
      - before
      - after
      - first-line
      - first-letter
      - selection
      - placeholder
*/
```

::before and ::after

"":

```
p::before {
    content: "";
    width: 0px;
    height: 3px;
    display: block;
    background-color: lightgreen;
```

```
    transition: width 1s;
}

p:hover::before {
    width: 120px;
}
```

Creates underline that grows on hover!

How it works:

```
<!-- HTML -->
<p>This is a paragraph</p>

<!-- Browser renders as: -->
<p>
    ::before (underline)
    This is a paragraph
</p>
```

Requirements:

- Must have `content` property (even if empty)
- Must have `display` property (inline by default)

Common uses:

1. Decorative elements:

```
h1::before {
    content: "★ ";
    color: gold;
}
```

2. Icons:

```
.external-link::after {
    content: " 🔗";
}
```

3. Quotes:

```
blockquote::before {
    content: open-quote;
}

blockquote::after {
    content: close-quote;
}
```

4. Clearfix:

```
.container::after {
    content: "";
    display: table;
    clear: both;
}
```

::first-line

"":

```
/* p::first-line */
p::first-line {
    color: lightgreen;
}
```

Styles only the first line of text (adjusts if window resizes).

```
p::first-line {
    font-weight: bold;
    font-size: 120%;
    color: blue;
}
```

Allowed properties:

- Font properties
- Color properties
- Background properties
- `text-decoration`, `text-transform`, `line-height`

::first-letter

"":

```
/* p::first-letter */
p::first-letter {
    font-size: 30px;
}
```

Creates drop cap effect.

```
p::first-letter {
    font-size: 3em;
    font-weight: bold;
    float: left;
    margin-right: 10px;
    line-height: 0.9;
}
```

Result:

```
This is a paragraph with a
  large first letter called
  a drop cap.
```

::selection

"":

```
/* p::selection */
p::selection {
    background-color: yellow;
    color: black;
}
```

Styles text when selected by user.

```
::selection {
    background-color: #ffb7b7;
    color: #fff;
}

/* Firefox */
::-moz-selection {
    background-color: #ffb7b7;
```

```
    color: #fff;
}
```

Allowed properties:

- `color`
- `background-color`
- `text-decoration`
- `text-shadow`

::placeholder

"":

```
/* input::placeholder */
input::placeholder {
    color: lightgreen;
}
```

Styles placeholder text in inputs.

```
input::placeholder {
    color: #999;
    font-style: italic;
    opacity: 0.7;
}

/* Browser prefixes */
input::-webkit-input-placeholder { /* Chrome, Safari */
    color: #999;
}

input::-moz-placeholder { /* Firefox 19+ */
    color: #999;
}

input:-ms-input-placeholder { /* IE 10-11 */
    color: #999;
}
```

Single vs Double Colon

CSS2 (single colon):

```
:before
:after
:first-line
:first-letter
```

CSS3 (double colon):

```
::before
::after
::first-line
::first-letter
::selection
::placeholder
```

Both work, but double colon is recommended:

```
/* Old style (still works) */
p:before { content: "→"; }

/* New style (recommended) */
p::before { content: "→"; }
```

---

11. Vendor Prefixes

""

```
/*
  Vendor Prefixes:

    -webkit- : chrome, safari
    -moz-    : firefox
    -ms-     : IE
    -o-      : opera

    -webkit-border-radius: 5px;
    -moz-border-radius: 5px;
    -ms-border-radius: 5px;
    -o-border-radius: 5px;
    border-radius: 5px;
*/
```

What Are Vendor Prefixes?

Browser-specific prefixes used for **experimental or new CSS features** before they're fully standardized.

Why Were They Needed?

Before standardization:

```
/* Different browsers implemented features differently */
-webkit-transform: rotate(45deg);  /* Chrome, Safari */
-moz-transform: rotate(45deg);     /* Firefox */
-ms-transform: rotate(45deg);      /* IE */
-o-transform: rotate(45deg);       /* Opera */
transform: rotate(45deg);          /* Standard */
```

Common Vendor Prefixes

-webkit- (WebKit engine)

- Chrome
- Safari
- Edge (Chromium-based)
- Opera (newer versions)

-moz- (Mozilla)

- Firefox

-ms- (Microsoft)

- Internet Explorer
- Old Edge

-o- (Opera)

- Opera (older versions)

Properties That Needed Prefixes

Border radius:

```
-webkit-border-radius: 10px;
-moz-border-radius: 10px;
```

```
border-radius: 10px;
```

Box shadow:

```
-webkit-box-shadow: 5px 5px 10px rgba(0,0,0,0.5);
-moz-box-shadow: 5px 5px 10px rgba(0,0,0,0.5);
box-shadow: 5px 5px 10px rgba(0,0,0,0.5);
```

Transform:

```
-webkit-transform: scale(1.5);
-moz-transform: scale(1.5);
-ms-transform: scale(1.5);
-o-transform: scale(1.5);
transform: scale(1.5);
```

Transition:

```
-webkit-transition: all 0.3s ease;
-moz-transition: all 0.3s ease;
-o-transition: all 0.3s ease;
transition: all 0.3s ease;
```

Animation:

```
-webkit-animation: spin 2s linear infinite;
-moz-animation: spin 2s linear infinite;
animation: spin 2s linear infinite;
```

Keyframes:

```
@-webkit-keyframes spin {
    from { -webkit-transform: rotate(0deg); }
    to { -webkit-transform: rotate(360deg); }
}

@-moz-keyframes spin {
    from { -moz-transform: rotate(0deg); }
    to { -moz-transform: rotate(360deg); }
}

@keyframes spin {
```

```
    from { transform: rotate(0deg); }
    to { transform: rotate(360deg); }
}
```

Order Matters!

Always put standard property last:

```
/* ✓ Correct */
-webkit-border-radius: 5px;
-moz-border-radius: 5px;
border-radius: 5px;  /* Standard last - overrides prefixes in modern browsers
*/

/* ✗ Wrong */
border-radius: 5px;
-webkit-border-radius: 5px;  /* This overrides standard! */
```

Modern Day (2024+)

Most prefixes are no longer needed:

```
/* Old way (no longer needed) */
-webkit-border-radius: 10px;
-moz-border-radius: 10px;
border-radius: 10px;

/* Modern way (sufficient) */
border-radius: 10px;
```

Still need prefixes for:

- Very new/experimental features
- Supporting very old browsers
- Some WebKit-specific properties

Example (still needs prefix):

```
/* Backdrop filter (still somewhat new) */
-webkit-backdrop-filter: blur(10px);
backdrop-filter: blur(10px);

/* Appearance (WebKit specific) */
-webkit-appearance: none;
```

```
-moz-appearance: none;
appearance: none;
```

Tools for Auto-Prefixing

1. Autoprefixer (recommended)

```
/* You write: */
.box {
    transform: scale(1.5);
}

/* Autoprefixer adds: */
.box {
    -webkit-transform: scale(1.5);
    -ms-transform: scale(1.5);
    transform: scale(1.5);
}
```

2. CSS preprocessors (SASS/LESS)

```
@mixin transform($value) {
    -webkit-transform: $value;
    -moz-transform: $value;
    -ms-transform: $value;
    transform: $value;
}

.box {
    @include transform(scale(1.5));
}
```

Browser Feature Detection

"" (HTML):

```
<script src="modernizr-custom.js"></script>

<script>
    if (!Modernizr.inputtypes.date) {
        alert("Please change your browser")
    }
</script>
```

Modernizr detects browser features without using vendor prefixes.

How it works:

```
// Check if feature is supported
if (Modernizr.cssanimations) {
    // Use animations
} else {
    // Fallback
}

if (Modernizr.borderradius) {
    // Use border-radius
} else {
    // Use images or other fallback
}
```

CSS classes added by Modernizr:

```
<!-- Modern browser -->
<html class="js flexbox canvas canvastext">

<!-- Old browser -->
<html class="no-js no-flexbox no-canvas no-canvastext">
```

```
/* Style for modern browsers */
.flexbox .container {
    display: flex;
}

/* Fallback for old browsers */
.no-flexbox .container {
    display: block;
    float: left;
}
```

---

12. Summary

CSS3 Key Features Recap

Visual Effects:

- `border-radius`: Rounded corners
- `box-shadow`: Element shadows
- `text-shadow`: Text shadows

Layout:

- `box-sizing`: Control box model calculation

Movement:

- `transform`: 2D/3D transformations
- `transition`: Smooth property changes
- `animation`: Complex keyframe animations

Selection:

- Advanced selectors (attribute, combinators)
- Pseudo-classes (`:hover` , `:nth-child` , etc.)
- Pseudo-elements (`::before` , `::after` , etc.)

Cross-Browser:

- Vendor prefixes (mostly legacy now)
- Feature detection (Modernizr)

Best Practices

1. **Always use** `box-sizing: border-box`
2. **Prefer** `transform` **and** `opacity` **for animations** (performance)
3. **Use transitions for simple state changes**
4. **Use animations for complex effects**
5. **Specify exact properties in transitions** (not `all`)
6. **Most vendor prefixes are no longer needed** (use Autoprefixer if needed)
7. **Test across browsers** for new features

This comprehensive guide covers all the CSS3 features "" with detailed explanations, examples, and best practices!

Abdullah Ali

Contact : +201012613453