BRO Academy

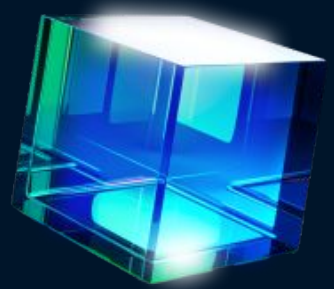
Manual

Lesson 9

Pseudo-classes

Help style specific attributes or states that are not reflected in the DOM.

- **User-action** pseudo-classes
- **The lang** pseudo-class
- **The negation** pseudo-class
- **Structural** pseudo-classes
- **User interface** pseudo-class selectors



Link Pseudo-classes for <a>

`:link` – styles unvisited links

`:visited` – styles visited links

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

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

User-action pseudo-classes

`:active` – element is being clicked

`:focus` – element is in focus

`:hover` – mouse is hovering over the element

```
a, button, input
```

The `lang` pseudo-class

Applied to elements with the `lang` attribute

```
p:lang(fr) { font-style: italic; }


<p lang="fr">
   Adieu
</p>
<p lang="jw">
   Sugeng rawuh
</p>
```

The negation pseudo-class

Styles are applied to all elements except those matching the selector

`:not(p) { }` – all elements except paragraphs **tags**

`:not(.intro) { }` – all elements except those **with class** .intro

`:not(#news) { }` – all elements except those **with id** #news

`:not(:lang(fr)) { }` – all elements **except** those with the French **language**

`:not([disabled]) { }` – all elements except those **without** the disabled **attribute**

`p:not(.intro) { }` – all **paragraphs** **except** those with the class `.intro`

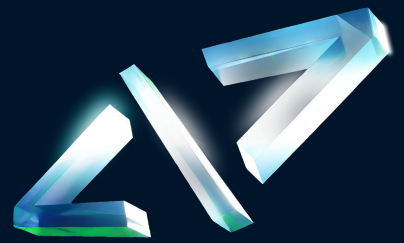
Structural pseudo-classes

Allow you to select elements based on their position in the document structure.

! If the document structure changes, the structural pseudo-class might apply to a different element or potentially to no element at all.

It can sometimes be difficult to determine exactly which element the styles will be applied to.

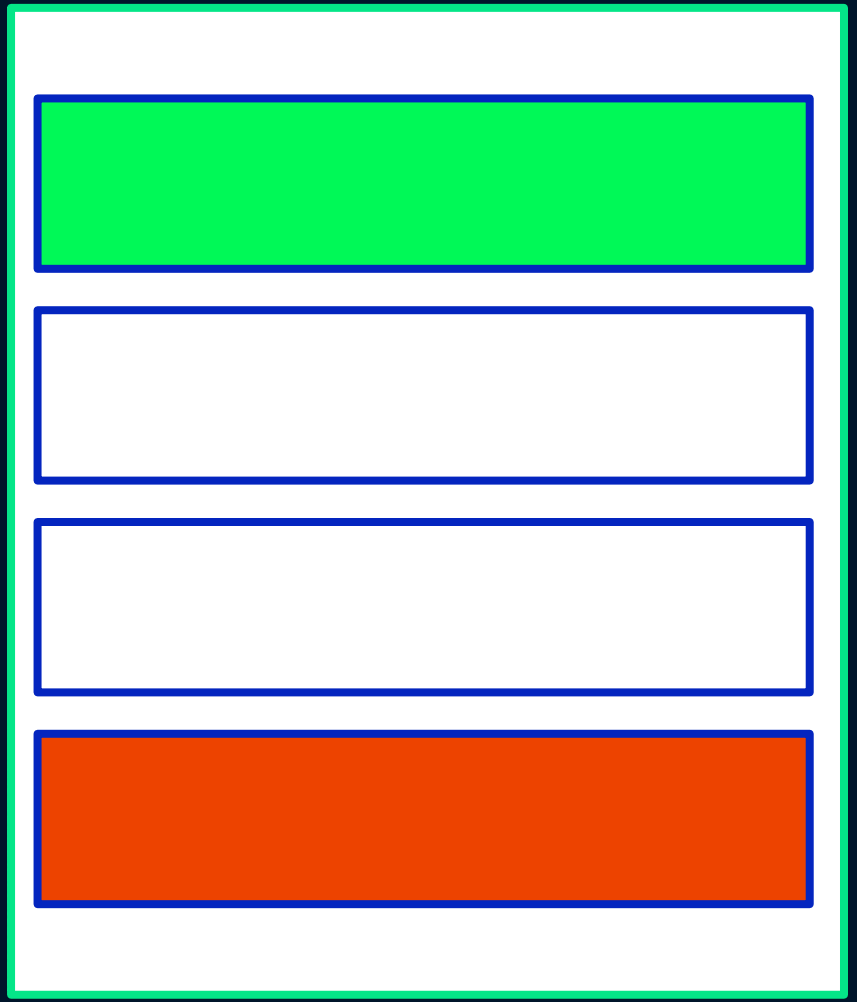
```
:first-child { }
:only-child { }
:nth-child(3n) { }
```

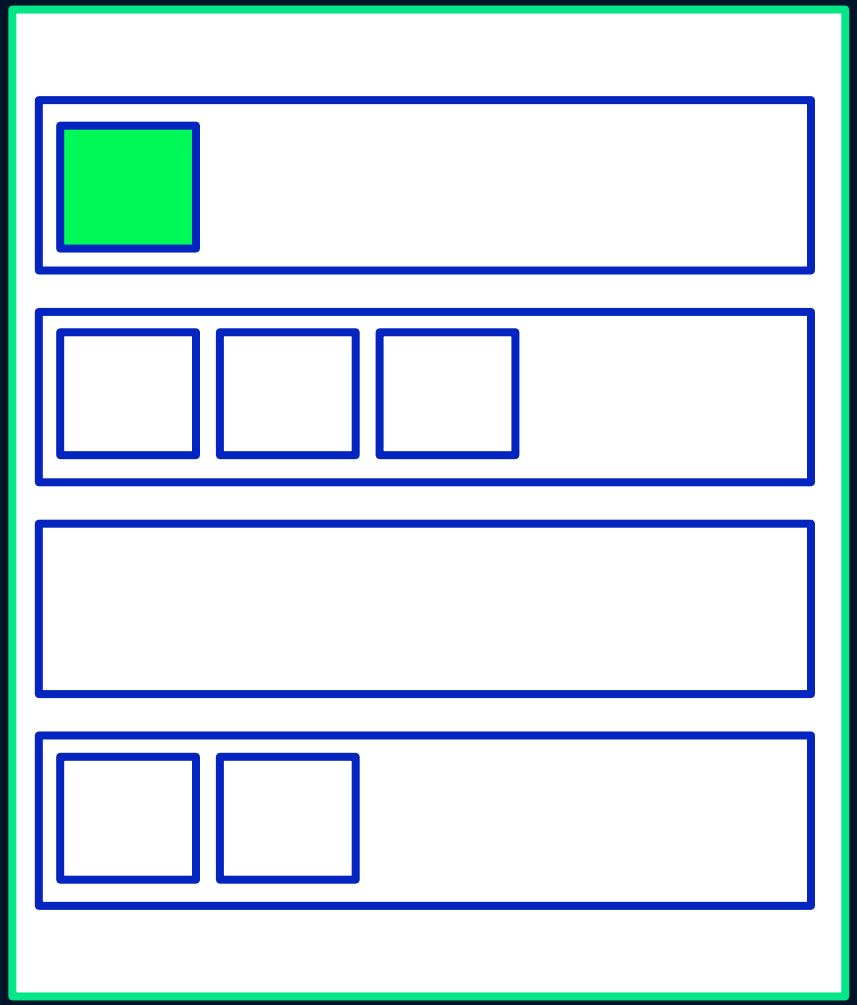


:first-child
:last-child

– Selects the element that is the **first/last child** of another element.

```
article:first-child { }
article:last-child { }

<section>
   <article> 1 </article>
   <article> 2 </article>
   <article> 3 </article>
   <article> 4 </article>
</section>
```



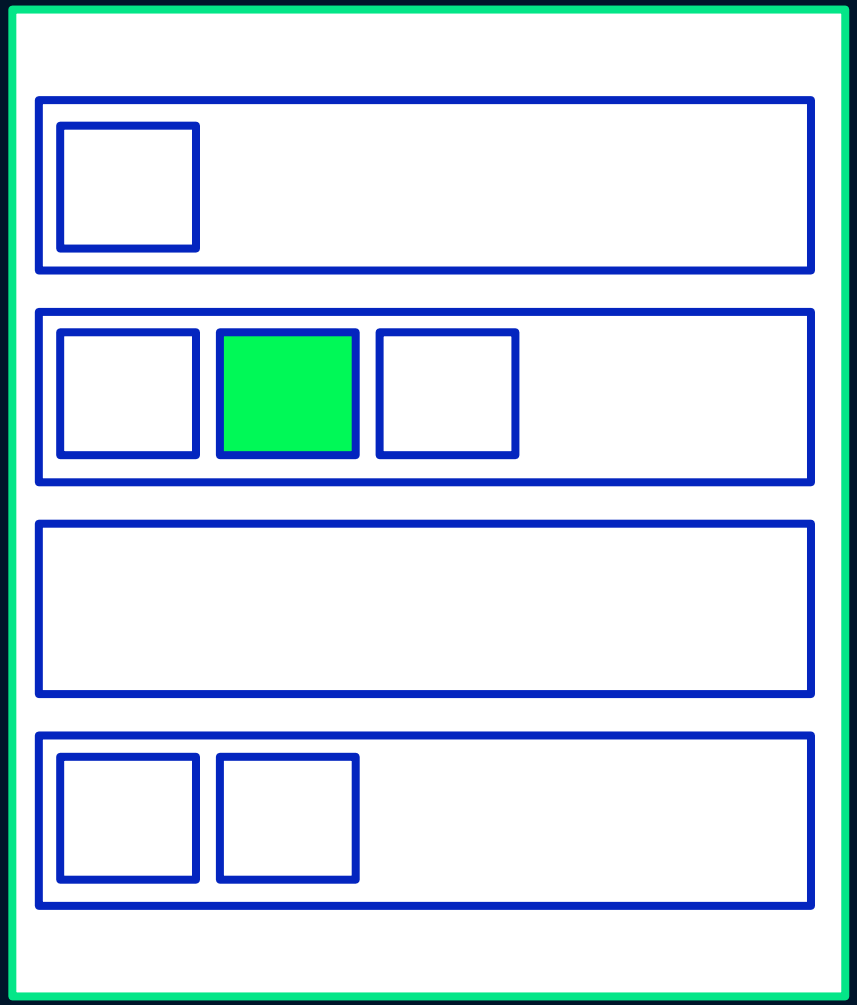


pseudo-class

:only-child

– Selects an element that is the only child of another element.

```
div:only-child { }

<article>
   <div> 1 </div>
</article>
```



pseudo-class

:only-of-type

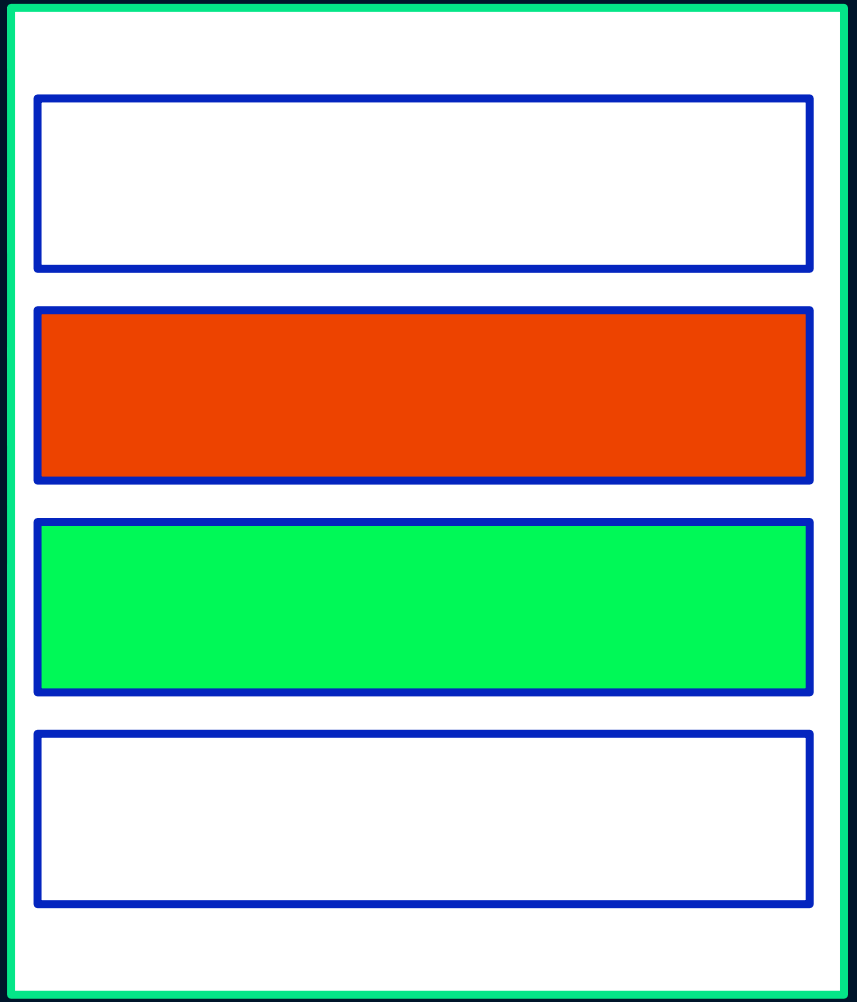
– Selects an element that is the **only** element of its **type** within its parent.

```
p:only-of-type { }

<article>
   <div> 1 </div>
   <p> 1 </p>
   <div> 1 </div>
</article>
```

:first-of-type
:last-of-type

– Selects an element that is the **first/last** of its **type** within its parent element.

```
p:first-of-type { }
p:last-of-type { }

<section>
   <article> 1 </article>
   <p> 2 </p>
   <p> 3 </p>
   <article> 4 </article>
</section>
```

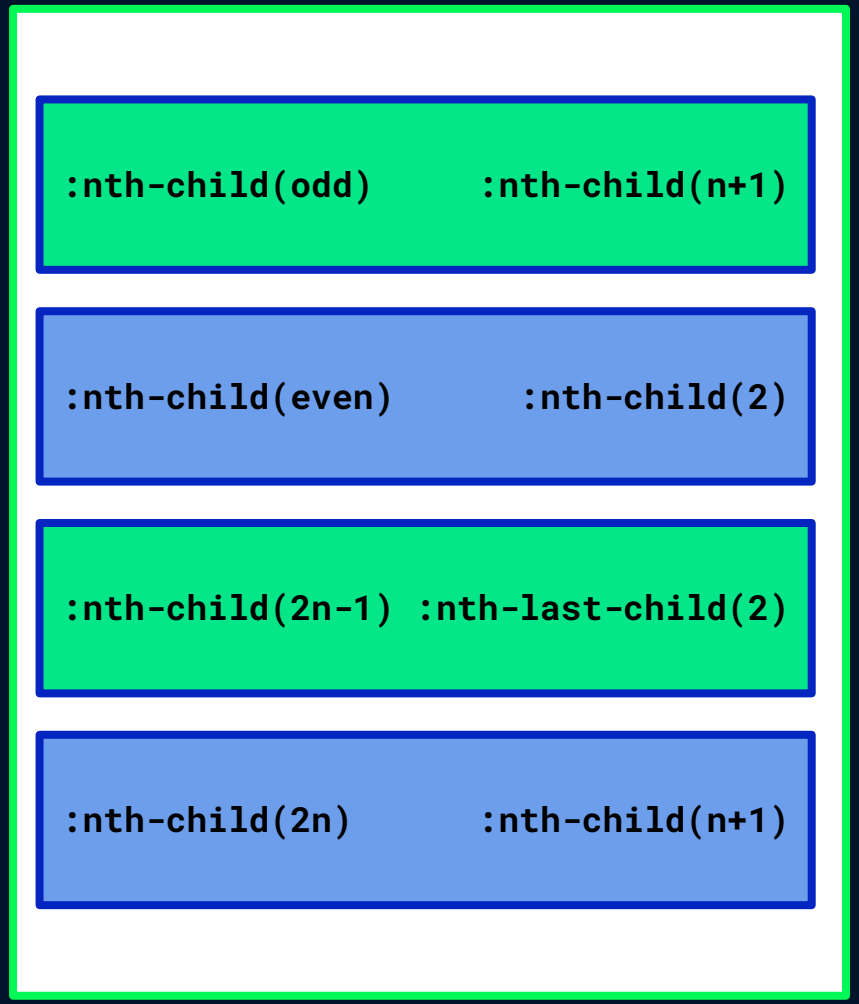


:nth-child(n)
:nth-last-child(n)

– Selects **specific child elements** in a parent element starting from the beginning or the end.

n:

- **number**
- **number + n** (selects every **n-th** element)
- expression with **+/-** (allows starting from an element other than the first)
- **even** (all even elements)
- **odd** (all odd elements)

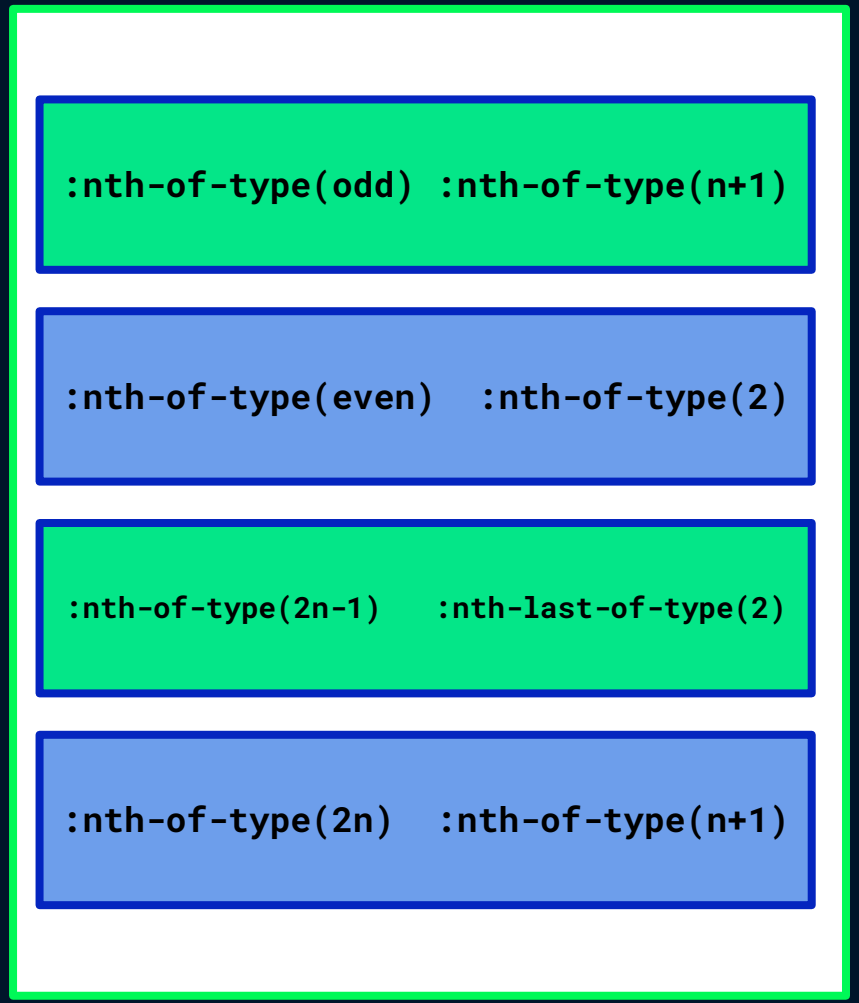


:nth-of-type(n)
:nth-last-of-type(n)

– Selects elements of a **specific type in the parent** element starting from the beginning or the end.

n:

- **number**
- **number + n** (selects every **n-th** element)
- expression with **+/-** (allows starting from an element other than the first)
- **even** (all even elements)
- **odd** (all odd elements)



:root

– Selects the root element of the document (тег **<html>**).

:empty

– Selects an element that has no content or child elements (an empty element).

A space is already a character, so the tag is no longer considered empty.
It also applies to input elements where no value has been entered.

```
:root { }

<html>
    <head> 1 </head>
    <body> 1 </body>
</html>

p:empty { }

<article>
    <p> 1 </p>
    <p> </p>
    <p></p>
    <p><span></span></p>
</article>
```

Pseudo-elements

– **(fake elements)** Allow styling elements that are not in the document tree.

`::-webkit-scrollbar` – styles the scrollbar

+ Other pseudo-elements of the form `::-webkit-scrollbar-*,` are used only with prefixes and only in `webkit` browsers

```
.invisible-scrollbar::-webkit-scrollbar {
  display: none;
}
```

Pseudo-elements for text

`::first-line` – styles the first line of text

`::first-letter` – styles the first letter of text

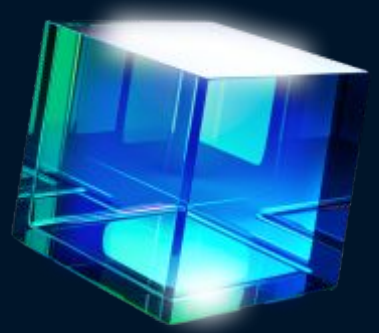
```
p::first-line { }
p:first-letter { }
<p>
   This is the first line
   of a paragraph of text
</p>
```

Pseudo-elements for Lists

- Usage of counters in lists
- Styling list markers

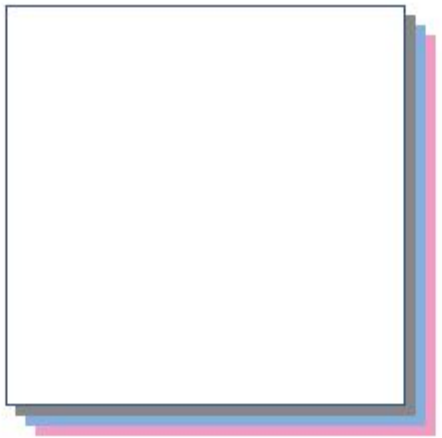
`::marker` – Styling list markers.

```
ol {
  counter-reset: section;
}
li::before {
  counter-increment: section;
  content: counter(section);
}
li::marker { }
```



box-shadow

box-shadow property adds shadow effects around an element. A box shadow is described by X and Y offsets relative to the element, blur and spread radii, and color.
If a border-radius is specified on the element with a box shadow, the box shadow takes on the same rounded corners.



You can specify a single box-shadow using 2–4 <length> values, <color> and inset:

- If only 2 values are given, they are interpreted as <offset-x><offset-y> values
- If a 3rd value is given, it is interpreted as a <blur-radius>
- If a 4th value is given, it is interpreted as a <spread-radius>

To specify multiple shadows, provide a comma-separated list of shadows (each one having bigger <length> values than the previous ones), for example:

Example

```
box-shadow: 5px 5px #888,
            10px 10px #7eb4e2,
            15px 15px #f69ec4;
```

box-shadow: inset 5px 8px 15px 18px red

offset-x, offset-y, blur-radius, spread-radius – order matters

inset
internal shadow

changes the shadow to one inside the box — inside the border, above the background, but below content

optional
order does not matter

offset-x
horizontal offset

negative values place the shadow to the left of the element;
positive ones put the shadow on the right of the box
0 = no offset

required

offset-y
vertical offset

negative values place the shadow above the element;
positive ones put the shadow below the box
0 = no offset

required

blur-radius

the larger this value, the bigger the blur;
negative values are not allowed
If not specified, it will be set at 0 and the shadow's edge will be sharp

optional

spread-radius

positive values will cause the shadow to expand and grow bigger,
negative values will cause the shadow to shrink
If not specified, it will be 0 and the shadow will be the same size as the element

optional

color

if not specified, it inherits the element's color

optional
order does not matter

Multiple box-shadow

can be applied to a single element by separating each shadow with a comma:

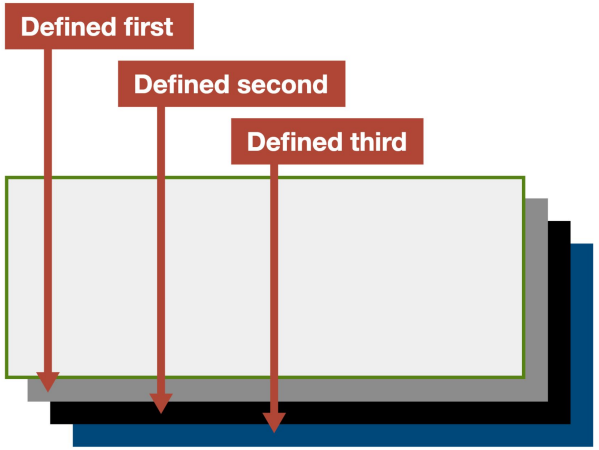
```
p { box-shadow: inset 10px 10px 10px 10px red,     /* all possible values specified */
                red 10px 10px 10px 10px inset,     /* different order */
                10px 10px 10px 10px red,           /* no inset specified */
                10px 10px 10px 10px,               /* no color specified */
                10px 10px 10px,                    /* no spread-radius specified */
                10px 10px; }                       /* no blur-radius specified */
```

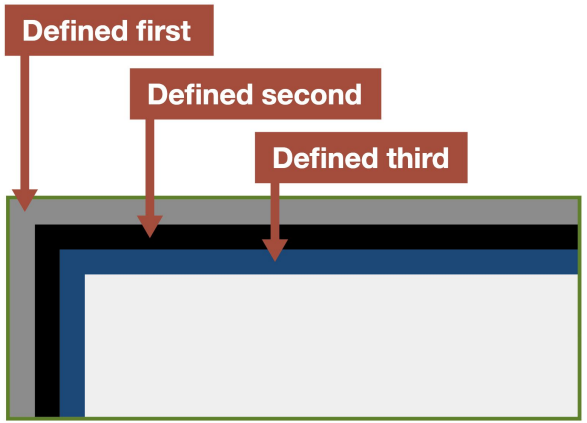
order in box-shadow

The order of the shadows matters

```
p { box-shadow: first, second, third; }
```

the first shadow in the list will be on top, and subsequent shadows will be layered below it.





```
p { box-shadow: inset offset-x, offset-y, blur-radius, spread-radius color; }
```

examples

box-shadow

text-shadow

The **`text-shadow`** property is useful to add a shadow to the text. Shadows can be single-layered or multi-layered, blurred, colored or transparent. By applying a shadow to an element, you can specify only one length and color value, thus creating a color copy of a single character or word. Also, with the help of the shadow, you can make the text more readable if the contrast between the text color and the background is not sufficient.

Each shadow is applied both to the text itself and to the text-decoration property if there is one. You can set several shadows at the same time, separated by commas. Shadows overlap but do not overlap the text itself. The first shadow is always located above the rest of the shadows.

`text-shadow: x-offset | y-offset | blur | color`

- **x-offset**: specifies the horizontal offset of the shadow; a positive value creates a shadow offset to the right of the text, and a negative length creates a shadow to the left
- **y-offset**: specifies the vertical offset of the shadow; a positive value shifts the shadow down, a negative value shifts it up
- **blur**: sets the blur radius; negative values are not allowed; if the blur value is 0, the edge of the shadow is clear; otherwise, the larger the value, the more blurred is the edge of the shadow

Example

```
text-shadow: 1px 1px #32557f,
             1px -1px #32557f,
             -1px 1px #32557f,
             -1px -1px #32557f,
             3px 3px 6px rgba(0,0,0,.5);
```

filter

– Applies visual effects to elements
(like in instagram)

Filters are commonly used to adjust the rendering of images, backgrounds, and borders.

The filter property is specified as none or one or more of the functions listed below. If the parameter for any function is invalid, the function returns none. Except where noted, the functions that take a value expressed with a percent sign (as in 34%) also accept the value expressed as decimal (as in 0.34).

Values:

- **Keyword –** `blur`
- **url –** `url("filters.svg#filter-id")`

```
filter: blur(5px);
filter: contrast(175%) brightness(3%);
```

filter

`blur(px)`

`brightness(0-1)`

`contrast(%)`

`drop-shadow(x y blur color)` – inner shadow

`grayscale(%)` – black and white

`hue-rotate(deg)` – shifts the color palette around the color wheel

`invert(%)`

`opacity(%)`

`saturate(%)`

`sepia(%)` – like vintage photo

No Filter Applied

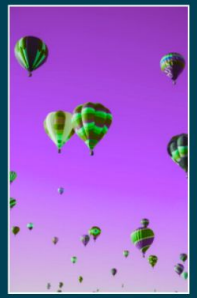
filter: blur(2px);

filter: brightness(0.4);

filter: contrast(200%);

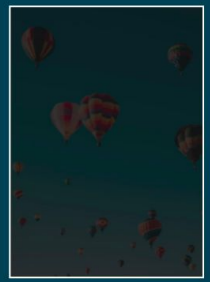
filter: drop-shadow(16px red);

filter: grayscale(80%);



filter: hue-rotate(90deg);

filter: invert(85%);



filter: opacity(15%);

filter: saturate(400%);

filter: sepia(560%);

CSS mask

Imagine cutting out a circle in a sheet of paper and placing it over a picture; you've applied a mask.

Think about masking as a way to apply complex, detailed, and shapes with varying opacity over another element. This can lead to really beautiful visual effects and performant alternatives to other techniques. For instance, animating gradients can be really CPU-intensive. But in the graphic below, we're animating the mask instead of the gradient to the same visual effect, and it's a lot less heavy.

Masks operate based on the alpha channel:

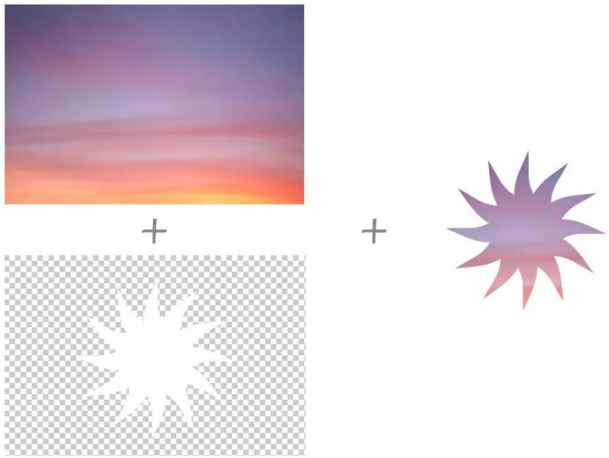
- ➔ **Black –** full invisibility
- ➔ **White –** full visibility
- ➔ **Gray –** partial transparency

Image:

The mask-image property works in a similar way to the background-image property. Use a url() value to pass in an image. Your mask image needs to have a transparent or semi-transparent area.

A fully transparent area will cause the part of the image under that area to be invisible. Using an area which is semi-transparent however will allow some of the original image to show through.

```
mask: url(mask.png);
```



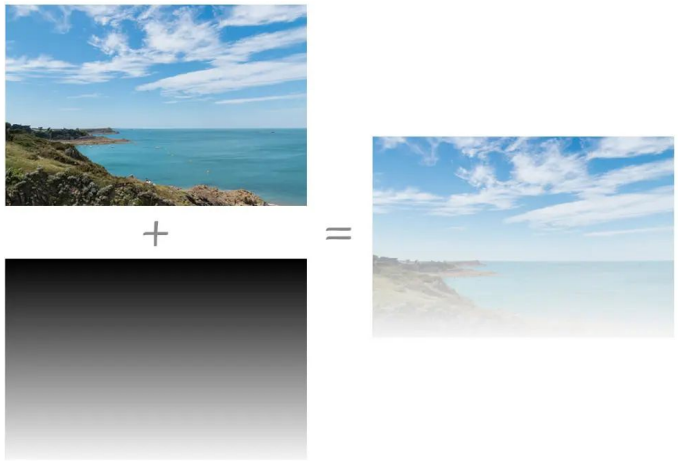
Gradient mask:

Using a CSS gradient as your mask is an elegant way of achieving a masked area without needing to go to the trouble of creating an image or SVG.

A simple linear gradient used as a mask could ensure that the bottom part of an image will not be too dark underneath a caption, for example.

You can use any of the supported gradient types, and get as creative as you like.

```
mask: linear-gradient(from, to);
```



CSS mask properties

mask-image – the image used as the mask

mask-mode – chooses the mask based on transparent or opaque areas

mask-position – mask position relative to the element

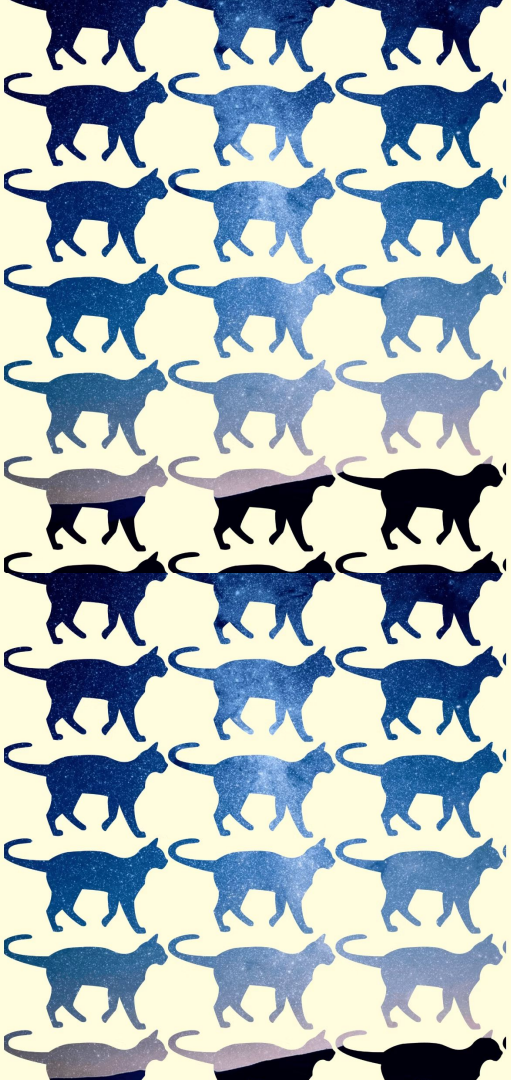
mask-size

mask-repeat – You can also repeat your mask just as you might repeat a background image, in order to use a small image as a repeating pattern.

mask-origin – defines the starting point of the mask **– border, padding, content**

mask-clip – the area to which the mask is applied

mask-composite – allows combining mask layers

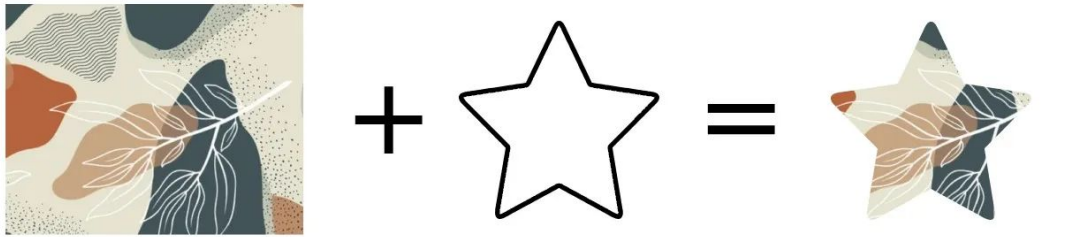


As with background images you can specify multiple mask sources, combining them to get the effect that you want. This is particularly useful if you want to use a pattern generated with CSS gradients as your mask. These typically will use multiple background images and so can be translated easily into a mask.

```
mask-image:
  linear-gradient(45deg, #000000 25%, rgba(0,0,0,0.2) 25%),
  linear-gradient(-45deg, #000000 25%, rgba(0,0,0,0.2) 25%),
  linear-gradient(45deg, rgba(0,0,0,0.2) 75%, #000000 75%),
  linear-gradient(-45deg, rgba(0,0,0,0.2) 75%, #000000 75%);
mask-size: 20px 20px;
mask-position: 0 0, 0 10px, 10px -10px, -10px 0px;
```

CSS clip-path

– Defines the area to show or hide



Think about clip-path in CSS as a way to cut a shape out of another shape. There's no concept of opacity, or alpha channel, to gray area here. Parts of the element with a clipping path applied are literally *visible* or *not visible*. **Clipping just uses the geometry of the shape. Because of this, certain visual elements won't be applied.** This includes, but is not limited to: stroke and stroke styles, gradients, and fill colors.

Another thing to keep in mind is that the pieces that are clipped away won't accept pointer events, so events can only be captured on the parts that you can visually see.

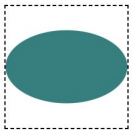
Generator clip-path – **Clippy**

CSS clip-path shapes

```
clip-path: inset(width height);
```

```
clip-path: inset(width height round border-radius);
```

```
clip-path: circle(radius at x, y);
```



```
clip-path: ellipse(radius-x, radius-y at x, y);
```

```
clip-path: polygon(vertex, vertex ...);
```

inset – rectangle

circle

ellipse

polygon – any shape with any number of corners

path – SVG path with coordinates

```
clip-path: path("M0.5,1 C0.5 ... “)
```

MASK VS CLIP-PATH

Raster	Vector
Partial Transparency	Opacity Only
Pre-drawn Images	Custom Shapes
More Complex Settings (mask-* properties)	Limits Element Shape (no additional properties)
Static Shape	Animatable Shape Changes
Text Wraps Shape Perimeter	Text Wraps Around Original Rectangle

shape-outside

Text Wrapping Around a Shape.

Shapes:

- **circle()** – Creates a circular shape for the text to wrap around.
- **ellipse()**
- **inset()** – Defines a rectangular area.
- **polygon()** – Creates any shape with three or more corners.
- **url()** – Uses an image as the shape for text wrapping.

Applied to an element that the text should wrap around.

shape-outside

```
clip-path: circle(70% at 0% 50%)
shape-outside: circle(70% at 0% 50%)
+
float: left/right
+
margin-left/right
```