BRO Academy

# Manual

CSS+

# PSEUDO CLASSES

**pseudo-classes** (or "fake" classes) are a way to style elements based on their state or position within the document, even though these states or positions are not explicitly represented in the HTML structure (DOM). Unlike classes, which are added manually in the HTML, pseudo-classes are used to target elements in specific states, such as when a user interacts with them or based on their position in the document.

# pseudo-classes

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

# Link Pseudo-classes for `<a>`

`:link` – elements that have not been visited by the user. By default, browsers typically style unvisited links in blue, but with the :link pseudo-class, you can customize this appearance to match the design of your website.

`:visited` – applies styles to links that the user has already clicked on and visited. This is useful for providing visual feedback, so users can easily distinguish between the links they have visited and those they haven't. Browsers typically style visited links in purple, but you can change this using the :visited pseudo-class to fit your design.
In practice, it is also used very rarely.

**pseudo-classes** allow you to style elements based on the way that users interact with these elements.
Start by explaining that user-action pseudo-classes are a group of CSS pseudo-classes that allow you to apply styles based on user interactions with elements on the webpage. These interactions include actions like clicking on an element, focusing on an input field, or hovering over a button or link. These pseudo-classes are essential for creating interactive and responsive designs that react to user behavior.

# User-action pseudo-classes

## :active

The :active pseudo-class applies styles to an element while it is being clicked or activated. This means that the styles defined under :active are visible only during the short moment when the user is pressing down on the element.
It's commonly used to give feedback to users when they click on buttons or links, often by changing the element's appearance to indicate that it is being clicked.

## :focus

The :focus pseudo-class targets elements that are currently in focus. This typically applies to form elements like input fields, text areas, or buttons when they are selected by the user, either by clicking on them or navigating to them using the keyboard (e.g., using the Tab key).
Styling elements with :focus is crucial for accessibility, ensuring that users who navigate your site via keyboard or assistive devices can easily see which element is currently active.

## :hover

The :hover pseudo-class is triggered when the user hovers the mouse pointer over an element. This is widely used to provide visual feedback, such as changing the color of a button or link when the user hovers over it.

# The `lang` pseudo-class

targets elements that have a specific language code in their lang attribute. This enables you to create language-specific styling rules, ensuring that content looks appropriate and consistent across different languages.

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


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

# The `negation` pseudo-class

takes a simple selector as an argument and applies styles to all elements that do not match that selector. It's especially useful when you want to apply a style broadly but need to exclude certain elements from that rule.

`: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`

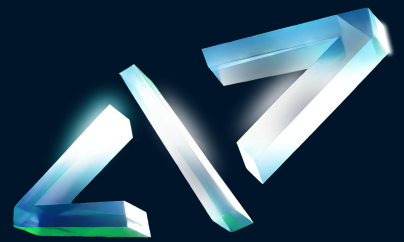
# Structural pseudo-classes

are selectors that allow you to apply styles to elements based on their position within the parent element or the document structure as a whole. These pseudo-classes are particularly useful for targeting elements in complex layouts where the order or structure of elements matters.

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

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

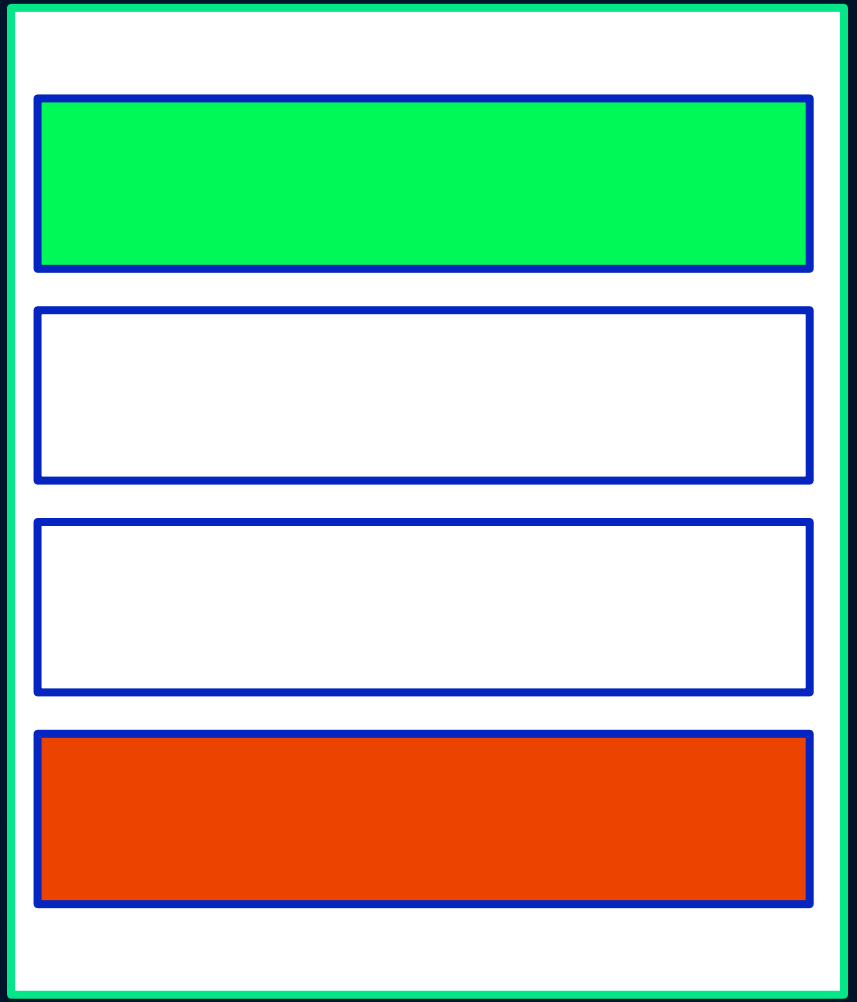
It can sometimes be difficult to determine exactly which element the styles will be applied to.

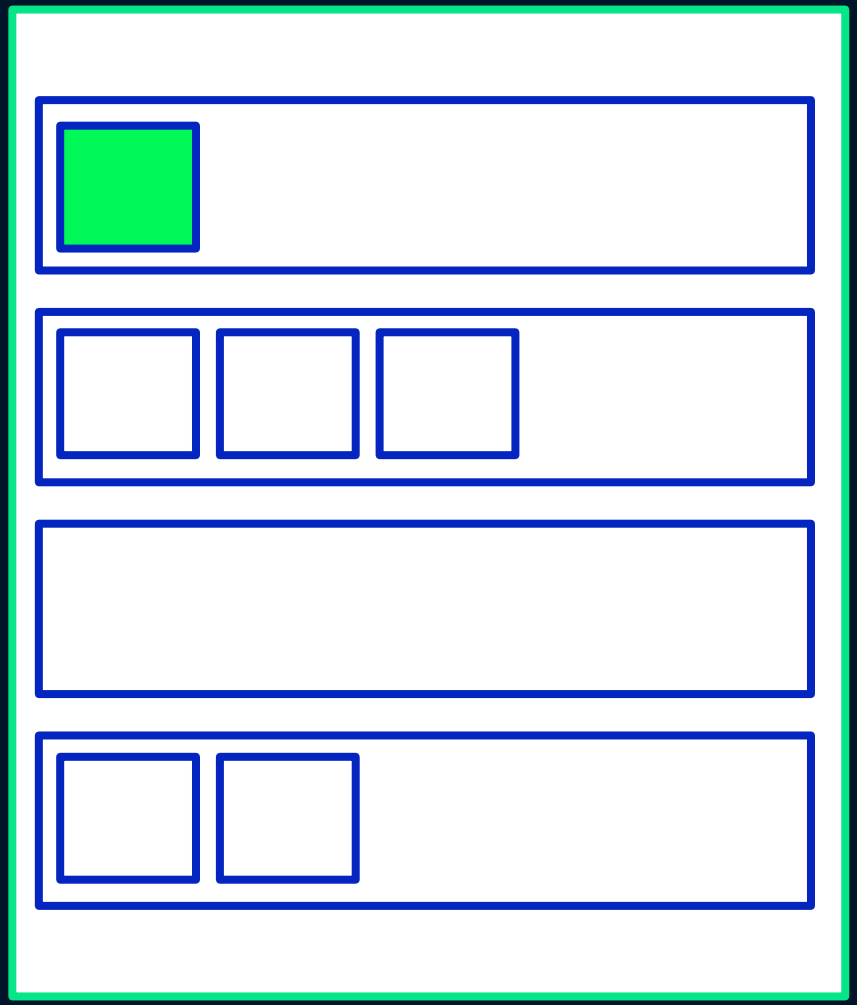


# :first-child
# :last-child

– 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>
```



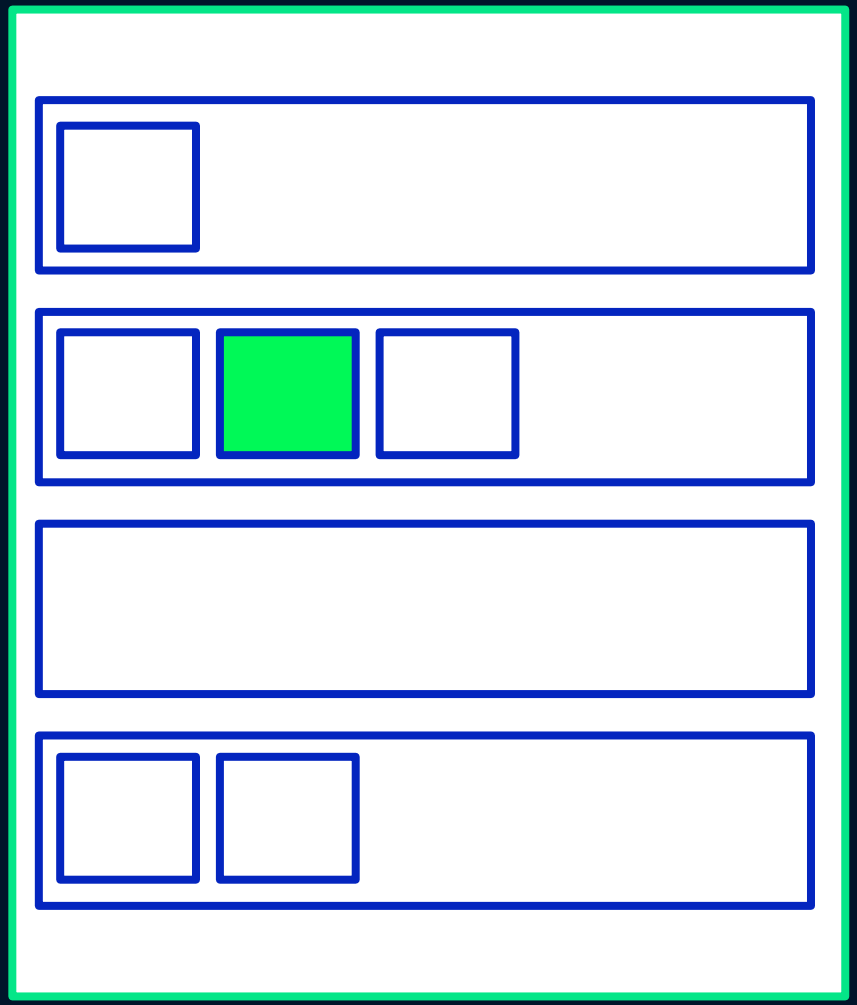


pseudo-class

# :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

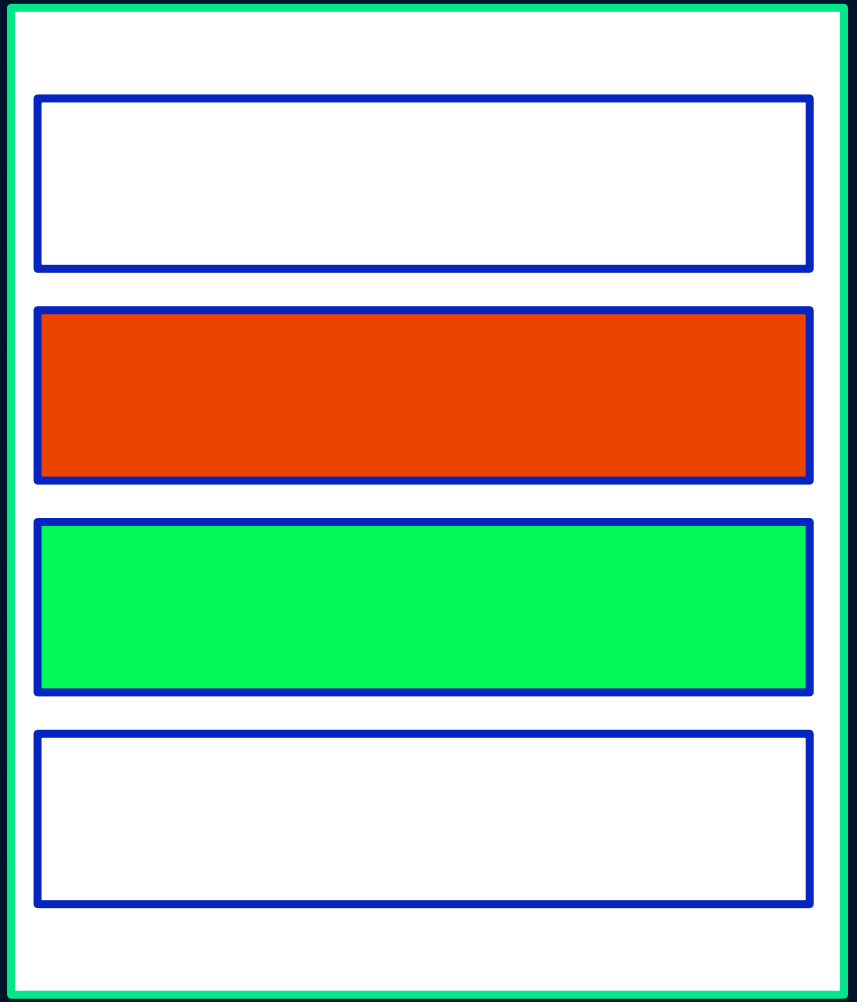
– Selects an element that is the **only** element of its **type** within its parent.

```
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>
```

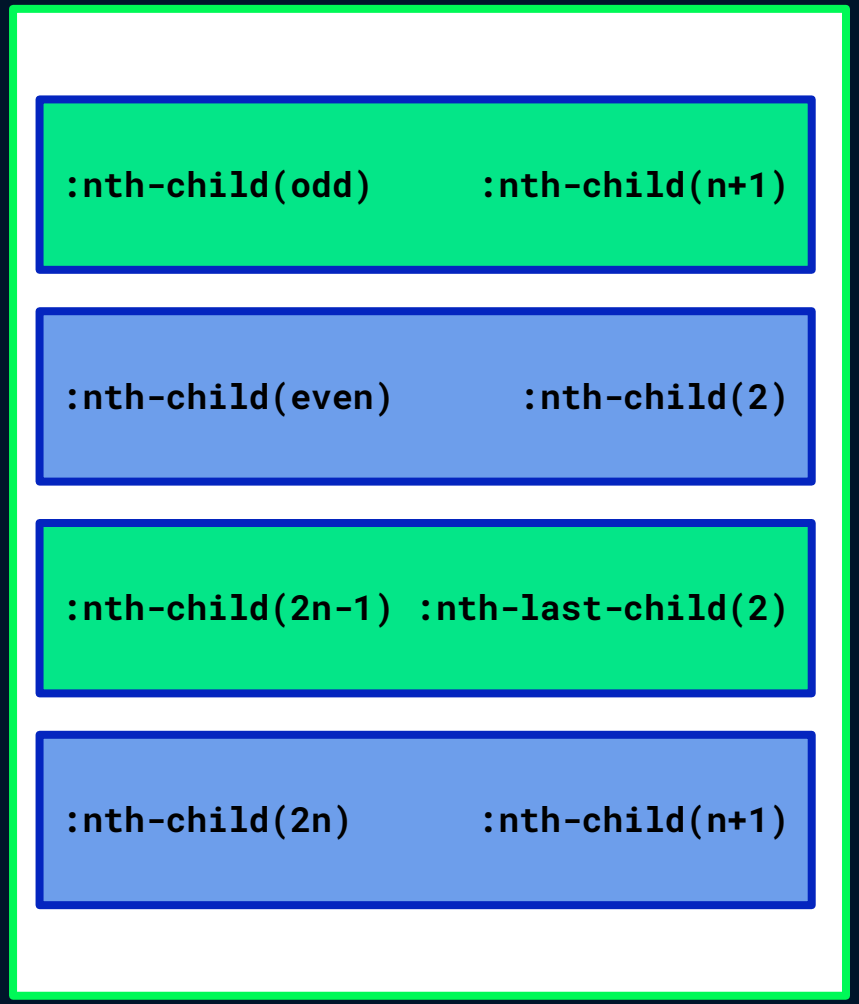


# :nth-child(n)
# :nth-last-child(n)

– 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)

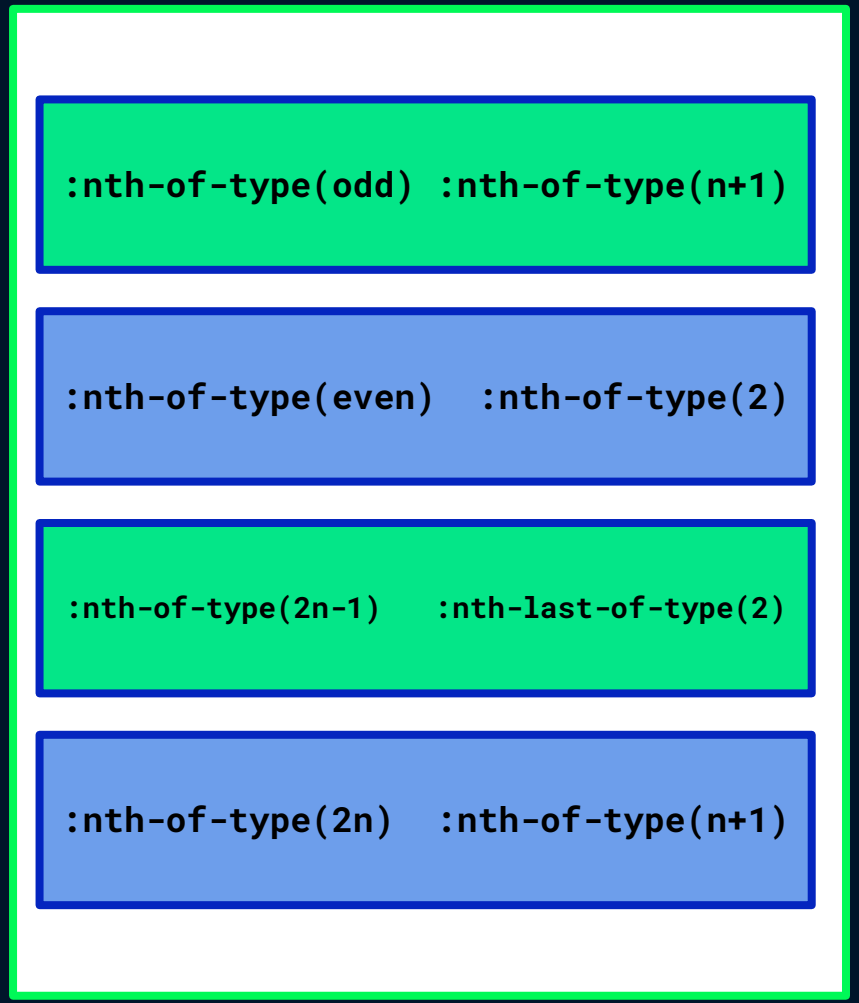


# :nth-of-type(n)
# :nth-last-of-type(n)

– 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

# 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 that Create a New Element

`::before` – A new element is created at the beginning of the element, before the content

`::after` – A new element is created at the end of the element, after the content

- An element can have only one `::before` and one `::after`
- These pseudo-elements are inserted into the document flow and occupy space
- They are visible in the inspector but cannot be accessed via JavaScript
- Any CSS properties can be applied to them.
- They can only be applied to elements with closing tags
- Often used to add decorative elements

# content

– replaces content with a generated value.
This is a **required** property for the `before` and `after` pseudo-elements

```
p::before {
  content: “”;
}
```

| | |
|---|---|
| **Text** | `“hello”;` |
| **Image** | `url(pic.png);` |
| **Attribute** (displays the value of an attribute as text) | `attr(cite);` |
| **Counter** | `counter(list-order);` |
| **Nothing** | `" ";` |
| **Special characters** | `"\21E6";` |
| **Emoji** | `"🎮";` |
| **Gradient** | `linear-gradient(#e66465, #9198e5);` |

# content

In the **content** property, you can combine different values:

```
p:before {
  content: "class: " attr(class);
}
```

The **text** "**class: **" will be placed before the paragraph content, followed by the list of **all class attribute** values for that paragraph.

---

Cannot contain HTML tags:

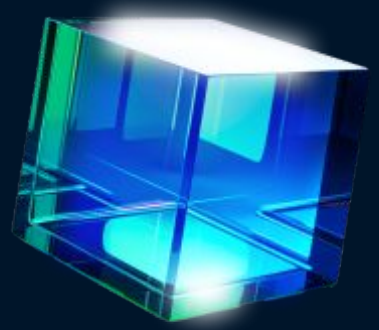
```
p:before {
  content: <p>test<p>;
}
```

# Pseudo-elements for Lists

- Usage of counters in lists
- Styling list markers

`::marker` – Styling list markers.

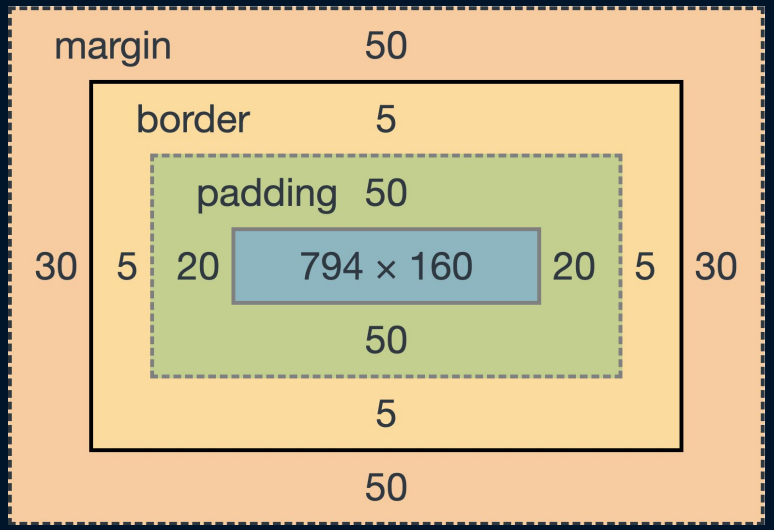
```
ol {
  counter-reset: section;
}
li::before {
  counter-increment: section;
  content: counter(section);
}
li::marker { }
```



# BOX MODEL

# Box model – is essentially a box that wraps around every HTML element.

- content
- padding – The space inside the element, adding space within the element
- border
- margin – The space outside the element, pushing neighboring elements away.



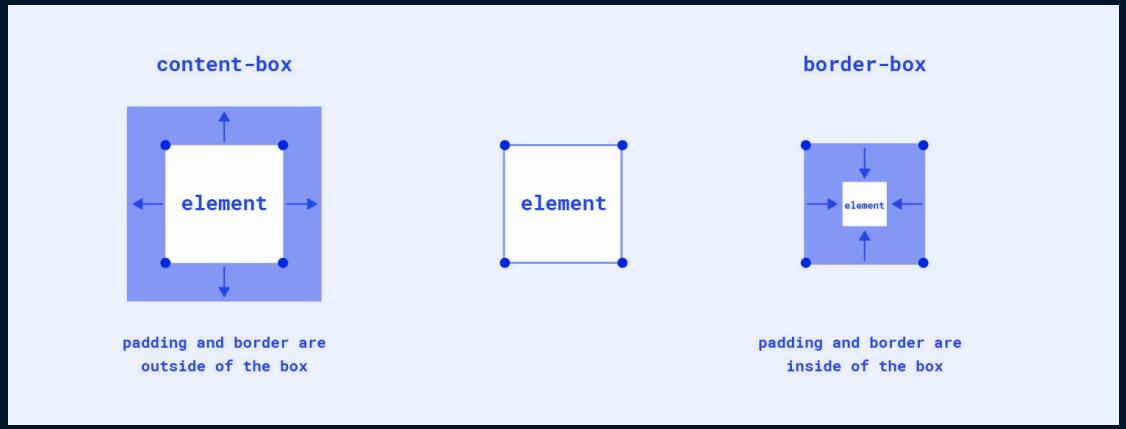
Demo

# box-sizing property

controls how the box model is handled for the element it applies to.

Default:
`content-box` – width and height properties include the `content`, but **does not include** the `padding`, `border`, or `margin`.

`border-box` – width and height properties **include** the `content`, `padding`, and `border`, but **do not include** the `margin`.



# padding

The padding CSS property is commonly used in CSS to add space in the inner side of an element.

**Remember:**

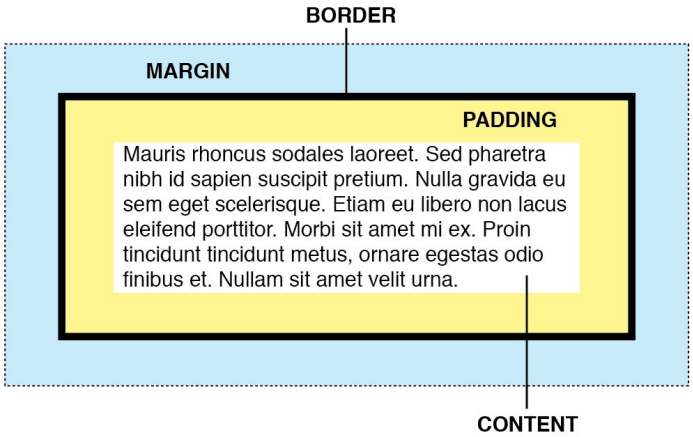
- margin adds space outside an element border
- padding adds space inside an element border

padding has 4 related properties that alter the padding of a single edge at once:

- `padding-top`
- `padding-right`
- `padding-bottom`
- `padding-left`

Units of measurement:

- `px`
- `%`




**Example**

```
padding-left: 30px;
padding-right: 3em;
```

`padding` is a shorthand to specify multiple padding values at the same time, and depending on the number of values entered, it behaves differently:

# `padding` shorthands

**1 value**
Using a single value applies that to **all** the paddings:
**top, right, bottom, left.**

```
padding: 30px;
```

**2 values**
the first to **bottom & top**,
and the second to **left & right**.

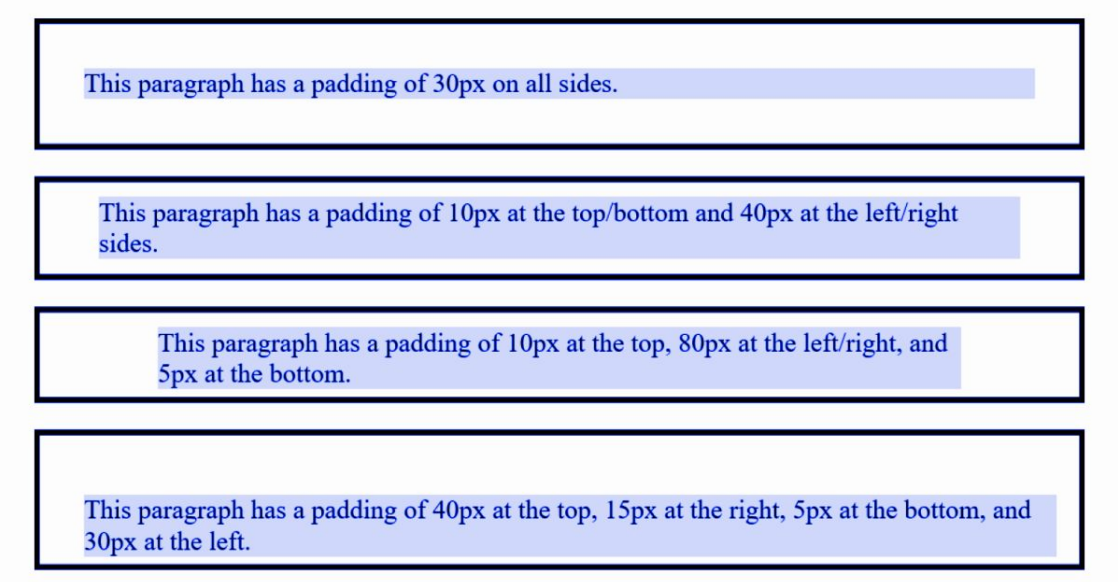
```
padding: 10px 40px;
```

**3 values**
the first to **top**,
the second to **left & right**,
the third to **bottom**.

```
padding: 10px 80px 5px;
```

**4 values**
the first to **top**,
the second to **right**,
the third to **bottom**,
the fourth to **left**.

```
padding: 40px 15px 5px 30px;
```



# margin

The margin CSS property is commonly used in CSS to add space around an element.
**Remember:**

- margin adds space outside an element border
- padding adds space inside an element border

`margin` has 4 related properties that alter the margin of a single edge at once:

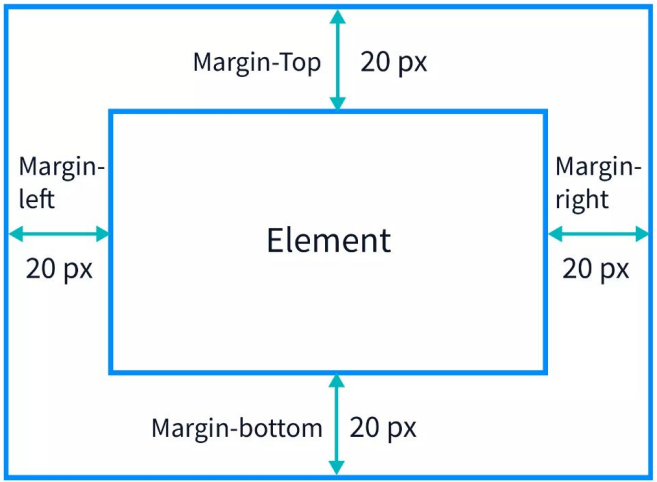
- `margin-top`
- `margin-right`
- `margin-bottom`
- `margin-left`

Units of measurement:

- **px**
- **%**
- **auto**

Can have negative values.

`margin-top` and `margin-bottom` do not work on inline-box




**Example**

```
margin-left: 30px;
margin-right: 3em;
```

# margin

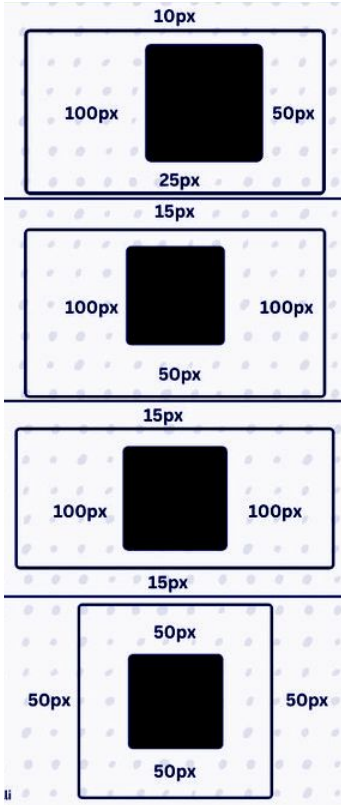
`margin` is a shorthand to specify multiple margins at the same time, and depending on the number of values entered, it behaves differently:

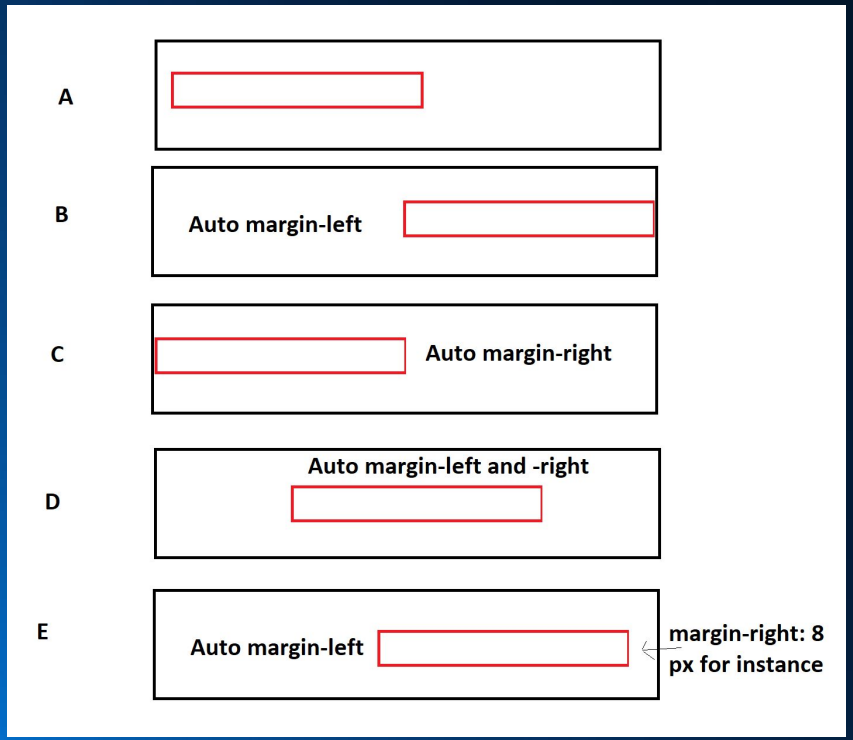
**1 value**
Using a single value applies that to **all** the paddings:
**top, right, bottom, left.**
```
margin: 50px;
```

**2 values**
the first to **bottom & top**,
and the second to **left & right**.
```
margin: 15px 100px;
```

**3 values**
the first to **top**,
the second to **left & right**,
the third to **bottom**.
```
margin: 15px 100px 50px;
```

**4 values**
the first to **top**,
the second to **right**,
the third to **bottom**,
the fourth to **left**.
```
margin: 10px 50px 25px 100px;
```






# margin: auto

```
margin: auto;
margin-right: auto:
margin: 0 auto;
```

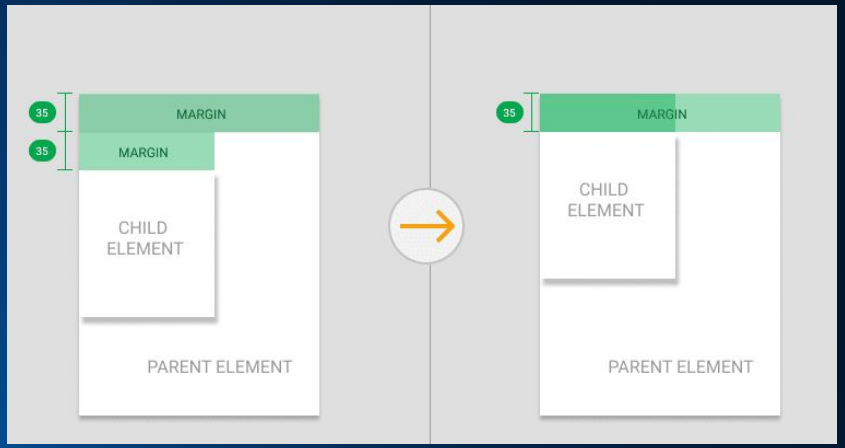
The width of the margin is automatically calculated by the browser and occupies all available space between elements and margins within the container.

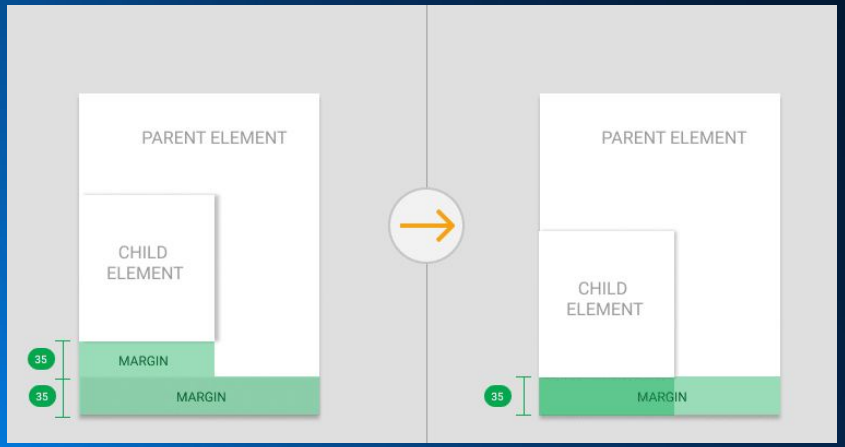
# Collapsing Margins

- The concept of margin collapsing comes from graphic design. There, you have margins to title and subtitles, but when a subtitle comes just after the title you should not double the margins. This is why they developed the concept of collapsed margins and this is why it happens just on vertical margins.

- **When?**
  If there is one thing to remember when you leave this page is that collapsing only happens on vertical margins.

- In elements with `position: absolute` and `position: fixed`, `display: flex`, `display: grid`; `float` margins do not collapse or cause margin drop.



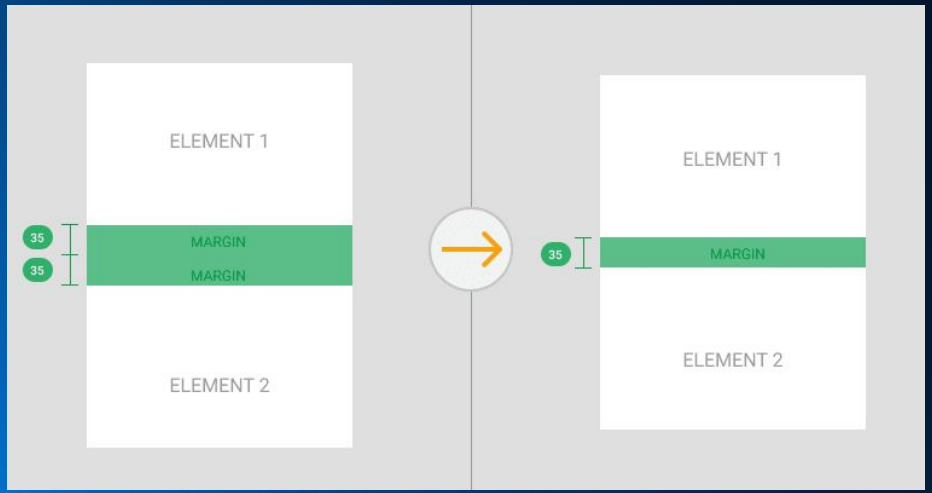




# Collapsing Margins

- If **no** border, padding, inline part or clearance separate the margin-top of a parent to his first child, then it is going to collapse.
- The same happens with the last child of the parent.
- Works for the `margin-top` of the first child inside a parent and for the `margin-bottom` of the last child inside a parent.
- The child's `margin` becomes the parent's `margin` or the larger of the two applies.

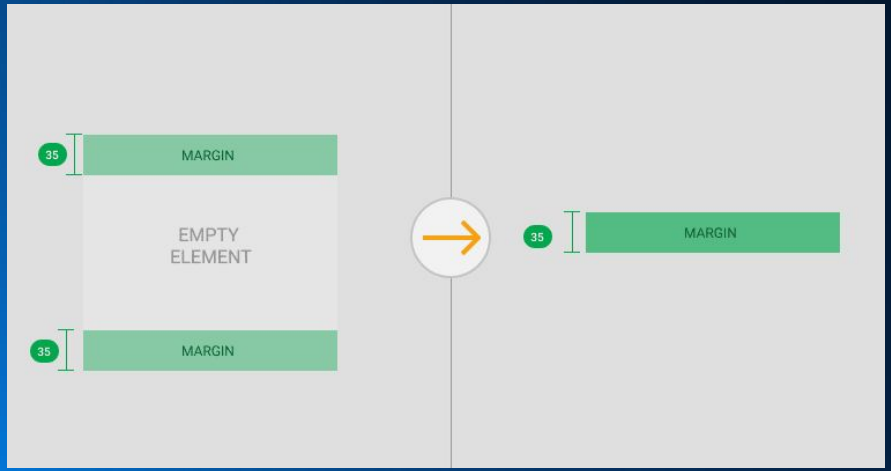
# Collapsing Margins: Adjacent Siblings

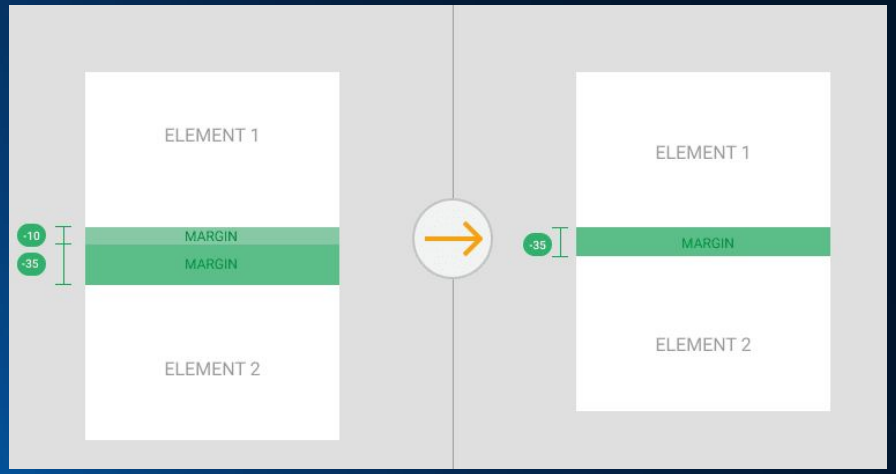
- When there are adjacent siblings, the margin-bottom of the top element collapses with the margin-top of the bottom element. The resulting size is the biggest between the two.

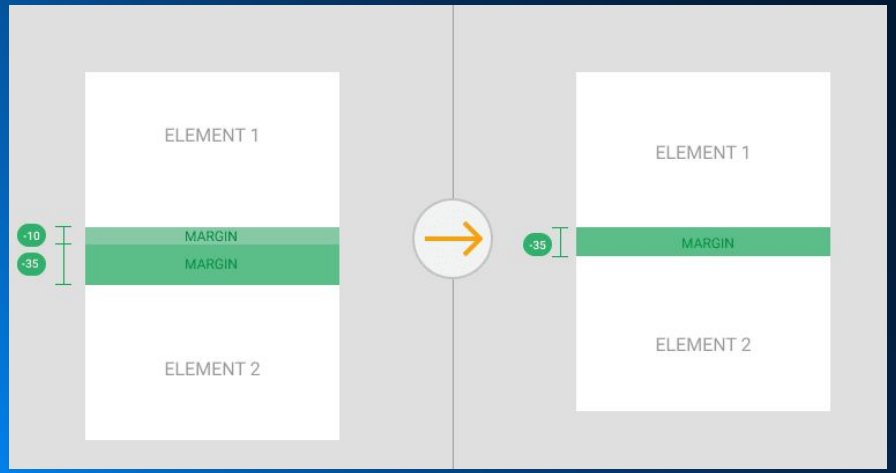


# Collapsing Margins: Empty Blocks

- The margin-top and margin-bottom of an empty block collapse if there are no padding, border, inline content avoiding it. Also, it does not collapse if the block does not have a height or min-height attribute.






# Collapsing Margins:

## Negative Margins

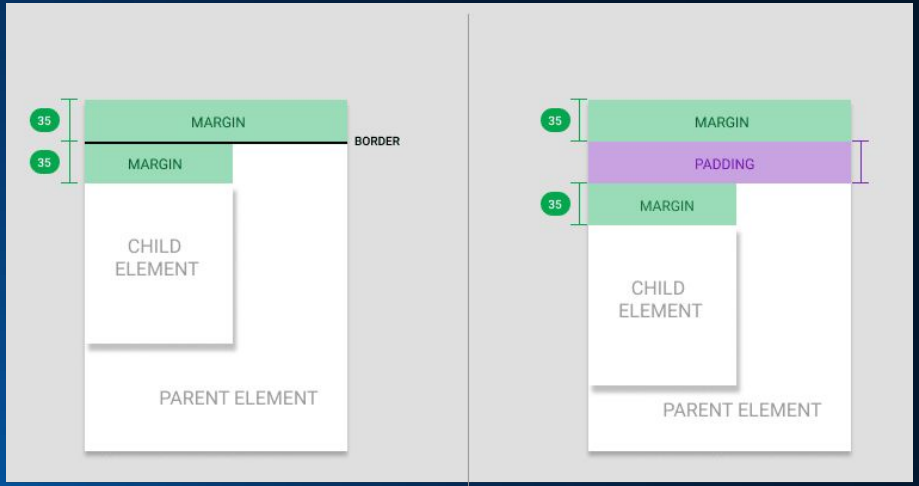
When it comes to negative margins, the resulting size depends on the signals involved.

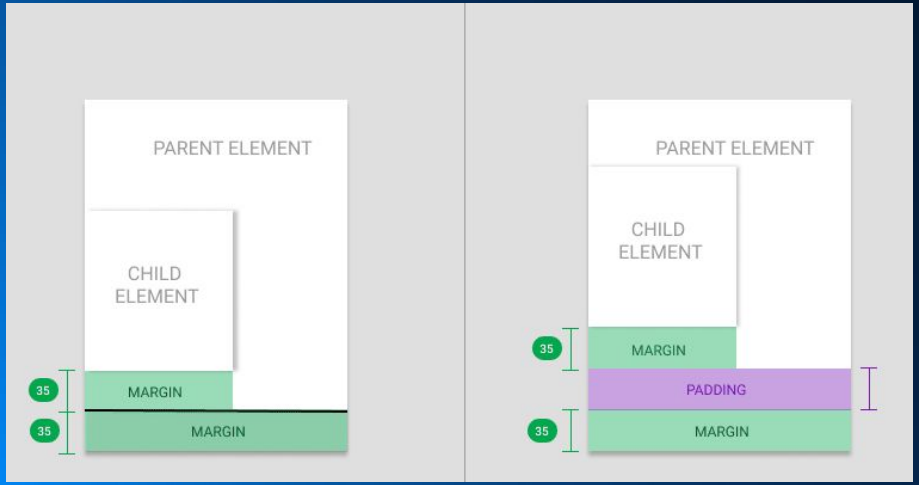
### Negative + Positive

The resulting size of a negative margin and a positive margin is their algebric sum.

### Negative + Negative

When both margins are negative, the resulting size is similar when there are positive + positive. In this, the most negative one wins.




# Collapsing Margins:

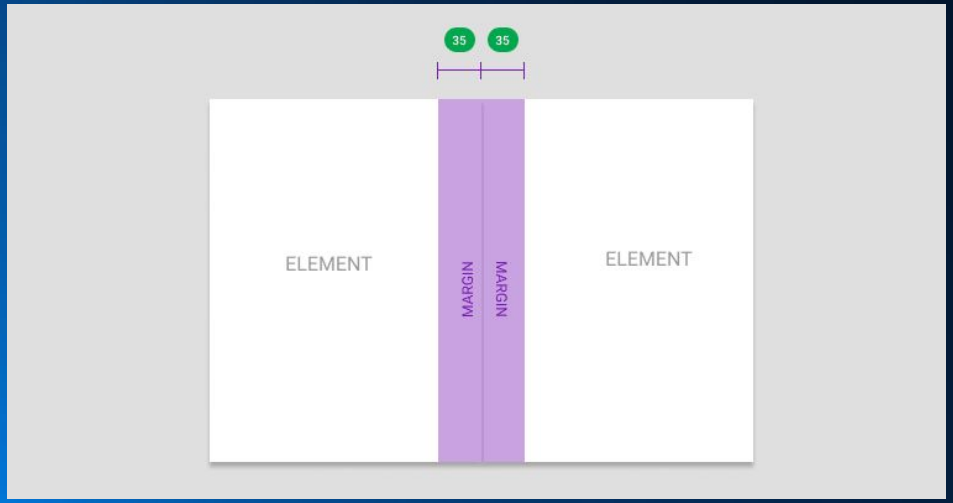
## Border or Padding Property on the Parent Element

If there is a **border-top** or **padding-top** between the parent and the first-child.

The same happens if there is a **border-bottom** or **padding-bottom** between the parent and its last-child.

# Collapsing Margins: horizontal margins

The horizontal margins never collapse.



# logical properties

The CSS logical properties and values module defines logical properties and values that can control layout through logical rather than physical direction and dimension mappings. Logical properties define direction-relative equivalents to their corresponding physical properties. This means content translated into languages with different writing modes will be rendered as intended.

The start of a line is not always the left side of a line. Different writing systems operate in various directions.

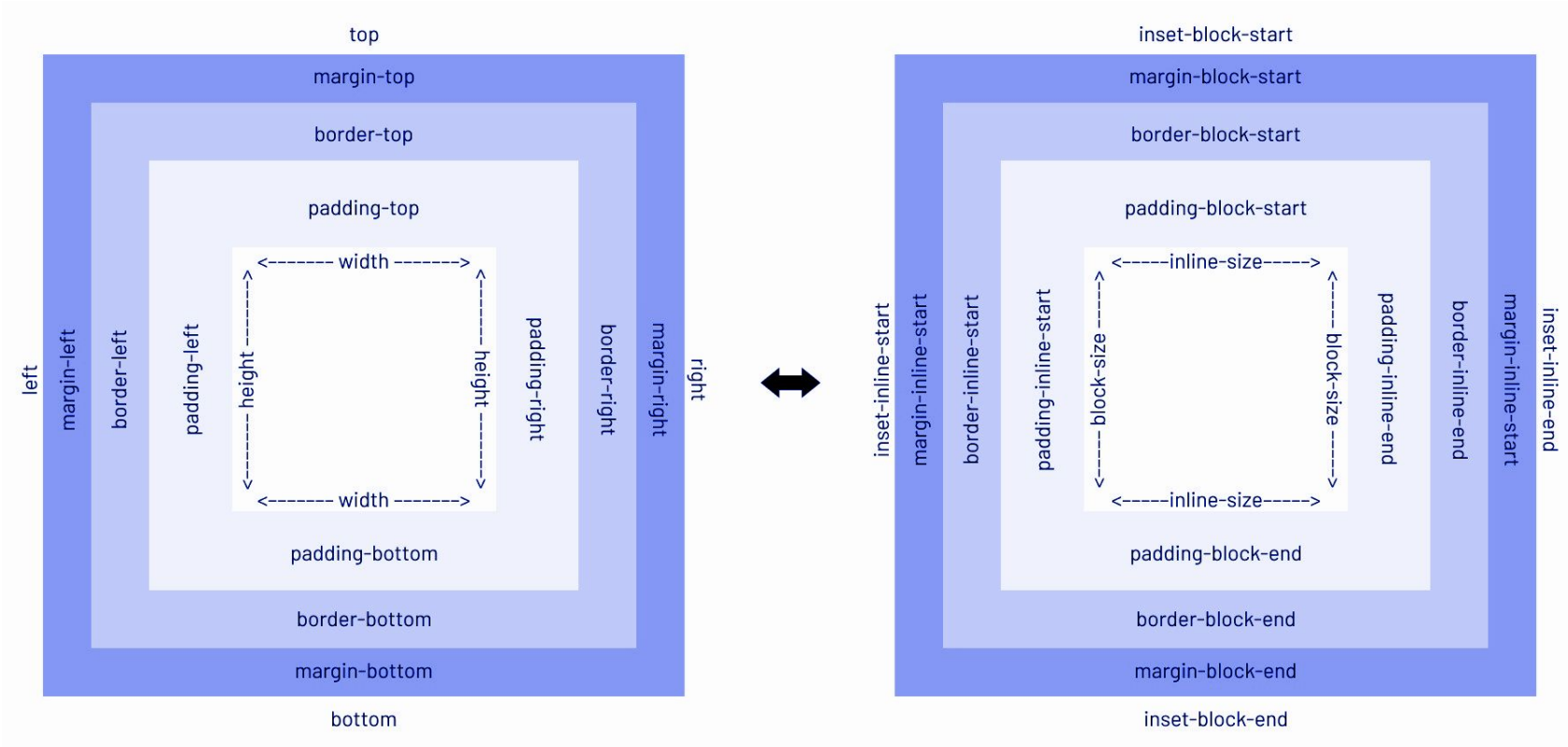
- English and Portuguese are written from left to right with new lines added below the previous ones.
- Hebrew and Arabic are right-to-left languages with new lines again being added below the previous ones.
- In some writing modes, the text lines are vertical, written from top to bottom. Chinese, Vietnamese, Korean, and Japanese are traditionally written vertically, from top to bottom, with each new vertical line added to the left of the previous one.
- Traditional Mongolian is also a top-to-bottom language, but new lines are to the right of previous ones.

The logical properties defined in this module enable defining properties relative to the content's writing direction, rather than a physical direction. This means content translated into languages with different writing modes will be rendered as intended.

**The block** dimension is perpendicular to the flow of text within a line, i.e., the vertical dimension in horizontal writing modes, and the horizontal dimension in vertical writing modes. For standard English text, it is the vertical dimension.

**The inline** dimension is parallel to the flow of text within a line, i.e., the horizontal dimension in horizontal writing modes, and the vertical dimension in vertical writing modes. For standard English text, it is the horizontal dimension.

# physical / logical



# logical properties shorthands

**Vertical**
Using a single or double value applies that to **vertical** paddings/margins:
**top + bottom**

```
padding-block: 30px;
margin-block: 30px 10px;
```

**Horizontal**
Using a single or double value applies that to **horizontal** paddings/margins:
**left + right**.

```
margin-inline: 10px;
padding-inline: 10px 40px;
```