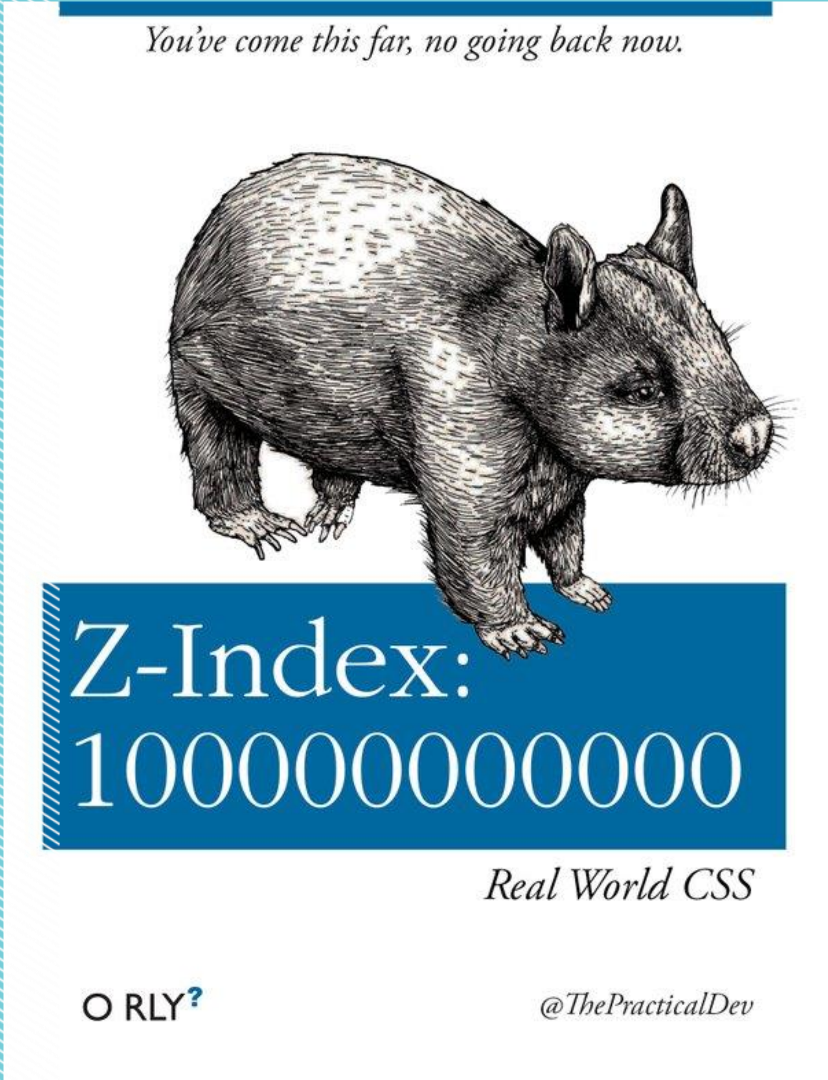
CSS Layouts

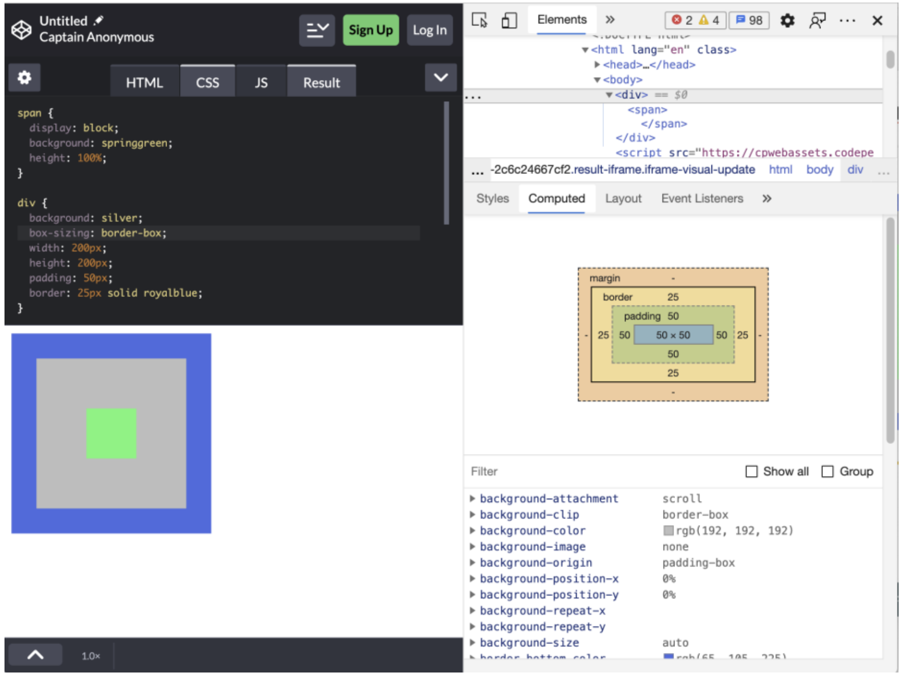
(how to fail centering things)



Frontend Junior Program - 2022

Agenda

Prerequisites



```
span {
  display: block;
  background: springgreen;
  height: 100%;
}

div {
  background: silver;
  box-sizing: border-box;
  width: 200px;
  height: 200px;
  padding: 50px;
  border: 25px solid royalblue;
}
```

```
▼<html lang="en" class>
  ▶<head>…</head>
  ▼<body>
    ▼<div> == $0
        <span>
          </span>
      </div>
      <script src="https://cpwebassets.codepe
```

```
… -2c6c24667cf2.result-iframe.iframe-visual-update   html   body   div
```

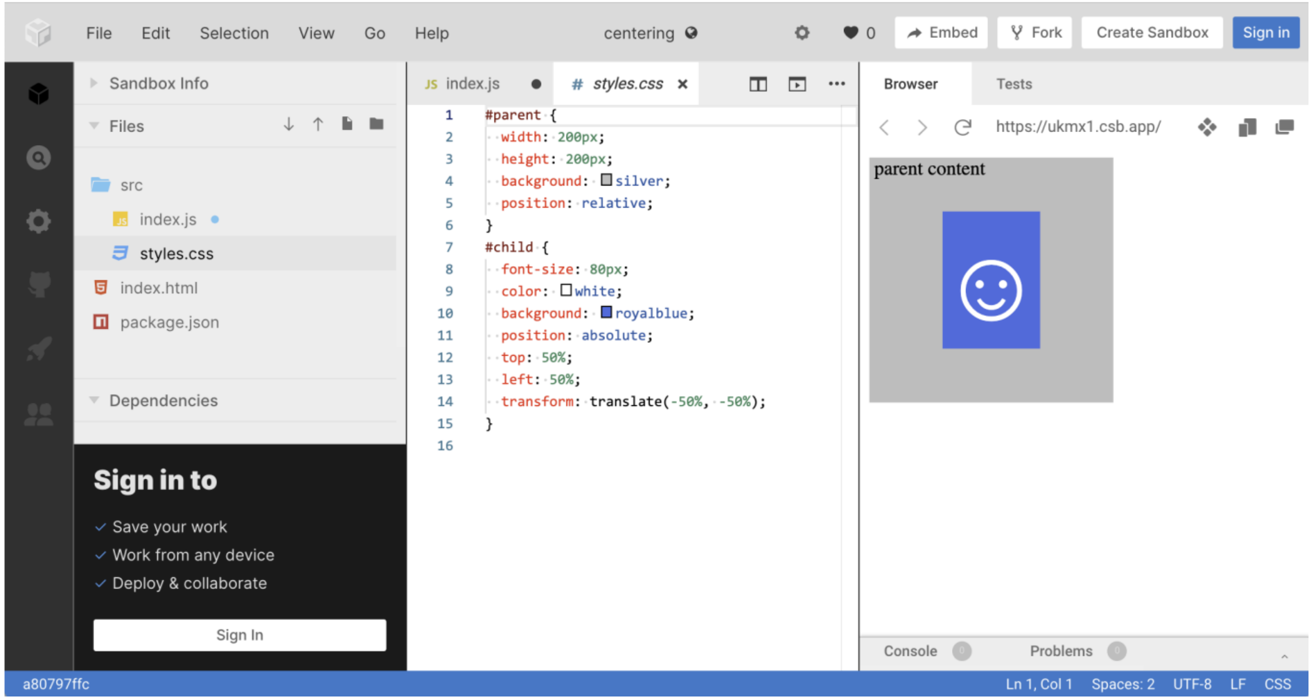
```
Filter                                  ☐ Show all   ☐ Group
▶ background-attachment       scroll
▶ background-clip             border-box
▶ background-color            rgb(192, 192, 192)
▶ background-image            none
▶ background-origin           padding-box
▶ background-position-x       0%
▶ background-position-y       0%
▶ background-repeat-x
▶ background-repeat-y
▶ background-size             auto
```

While working with CSS / layouts the inspector is always our friend.

Mac: Option (⌥) + Command (⌘) + C
Windows: CTRL + SHIFT + C

Chrome DevTools

Prerequisites



Sign in to
- Save your work
- Work from any device
- Deploy & collaborate

Sign In

```css
#parent {
  width: 200px;
  height: 200px;
  background: silver;
  position: relative;
}
#child {
  font-size: 80px;
  color: white;
  background: royalblue;
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

Online code editors also helps in practicing – explore different ones!

Prerequisites

Every text in this document written in *italic* is a reference to the standard, or excerpt from MDN Web Docs.

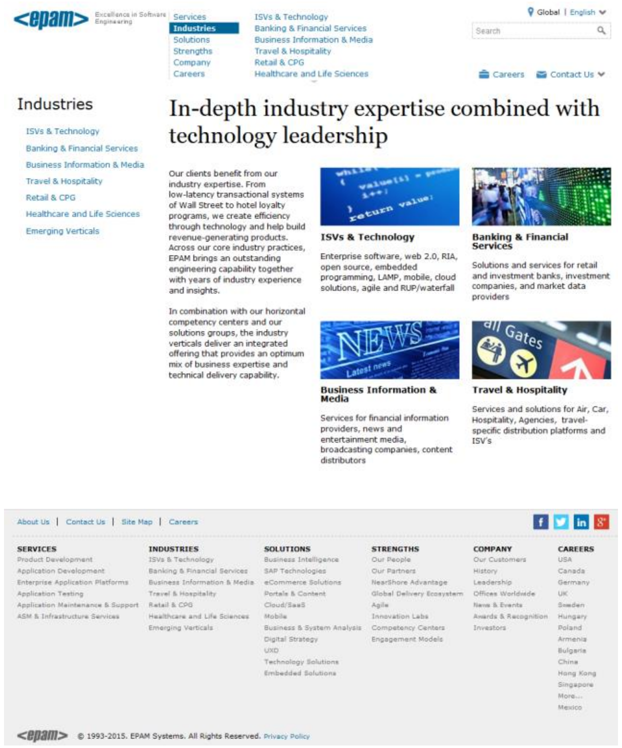
The goal of this document is to serve as a starting point of your exploration.

Also, we try to share our project experiences and to provide best practices.

Please always check the links provided, and follow this path:

- first try to have a quick overview
- practice
- have a deep dive
- goto 10 (if needed)

CSS layout



Excellence in Software Engineering

Services
Industries
Solutions
Strengths
Company
Careers

ISVs & Technology
Banking & Financial Services
Business Information & Media
Travel & Hospitality
Retail & CPG
Healthcare and Life Sciences

Global | English

Search

Careers Contact Us

Industries

- ISVs & Technology
- Banking & Financial Services
- Business Information & Media
- Travel & Hospitality
- Retail & CPG
- Healthcare and Life Sciences
- Emerging Verticals

In-depth industry expertise combined with technology leadership

Our clients benefit from our industry expertise. From low-latency transactional systems of Wall Street to hotel loyalty programs, we create efficiency through technology and help build revenue-generating products. Across our core industry practices, EPAM brings an outstanding engineering capability together with years of industry experience and insights.

In combination with our horizontal competency centers and our solutions groups, the industry verticals deliver an integrated offering that provides an optimum mix of business expertise and technical delivery capability.

ISVs & Technology

Enterprise software, web 2.0, RIA, open source, embedded programming, LAMP, mobile, cloud solutions, agile and RUP/waterfall

Banking & Financial Services

Solutions and services for retail and investment banks, investment companies, and market data providers

Business Information & Media

Services for financial information providers, news and entertainment media, broadcasting companies, content distributors

Travel & Hospitality

Services and solutions for Air, Car, Hospitality, Agencies, travel-specific distribution platforms and ISV's

About Us | Contact Us | Site Map | Careers

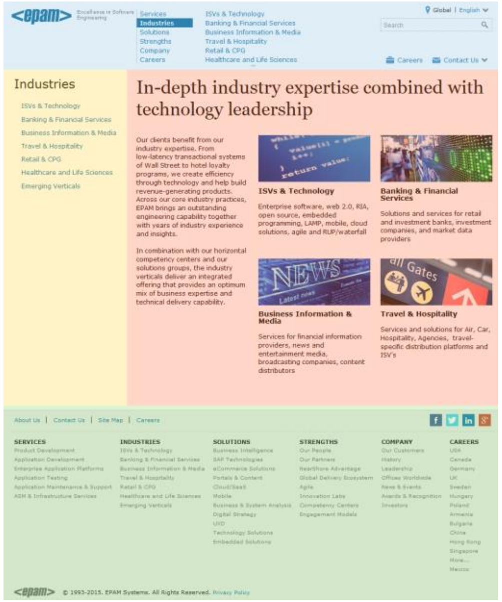
SERVICES	INDUSTRIES	SOLUTIONS	STRENGTHS	COMPANY	CAREERS
Product Development	ISVs & Technology	Business Intelligence	Our People	Our Customers	USA
Application Development	Banking & Financial Services	SAP Technologies	Our Partners	History	Canada
Enterprise Application Platforms	Business Information & Media	eCommerce Solutions	NearShore Advantage	Leadership	Germany
Application Testing	Travel & Hospitality	Portals & Content	Global Delivery Ecosystem	Offices Worldwide	UK
Application Maintenance & Support	Retail & CPG	Cloud/SaaS	Agile	News & Events	Sweden
ASM & Infrastructure Services	Healthcare and Life Sciences	Mobile	Innovation Labs	Awards & Recognition	Hungary
	Emerging Verticals	Business & System Analysis	Competency Centers	Investors	Poland
		Digital Strategy	Engagement Models		Armenia
		UXD			Bulgaria
		Technology Solutions			China
		Embedded Solutions			Hong Kong
					Singapore
					More...
					Mexico

© 1993-2015. EPAM Systems. All Rights Reserved. Privacy Policy

CSS page layout techniques allow us to control the positions of the elements on the page.

MDN: Introduction to CSS layout

Design analysis



Excellence in Software Engineering

Services
Industries
Solutions
Strengths
Company
Careers

ISVs & Technology
Banking & Financial Services
Business Information & Media
Travel & Hospitality
Retail & CPG
Healthcare and Life Sciences

Global | English
Search
Careers Contact Us

Industries

ISVs & Technology
Banking & Financial Services
Business Information & Media
Travel & Hospitality
Retail & CPG
Healthcare and Life Sciences
Emerging Verticals

In-depth industry expertise combined with technology leadership

Our clients benefit from our industry expertise. From low-latency transactional systems of Wall Street to hotel loyalty programs, we create efficiency through technology and help build revenue-generating products. Across our core industry practices, EPAM brings an outstanding engineering capability together with years of industry experience and insights.

In combination with our horizontal competency centers and our solutions groups, the industry verticals deliver an integrated offering that provides an optimum mix of business expertise and technical delivery capability.

ISVs & Technology

Enterprise software, web 2.0, RIA, open source, embedded programming, LAMP, mobile, cloud solutions, agile and RUP/waterfall

Banking & Financial Services

Solutions and services for retail and investment banks, investment companies, and market data providers

Business Information & Media

Services for financial information providers, news and entertainment media, broadcasting companies, content distributors

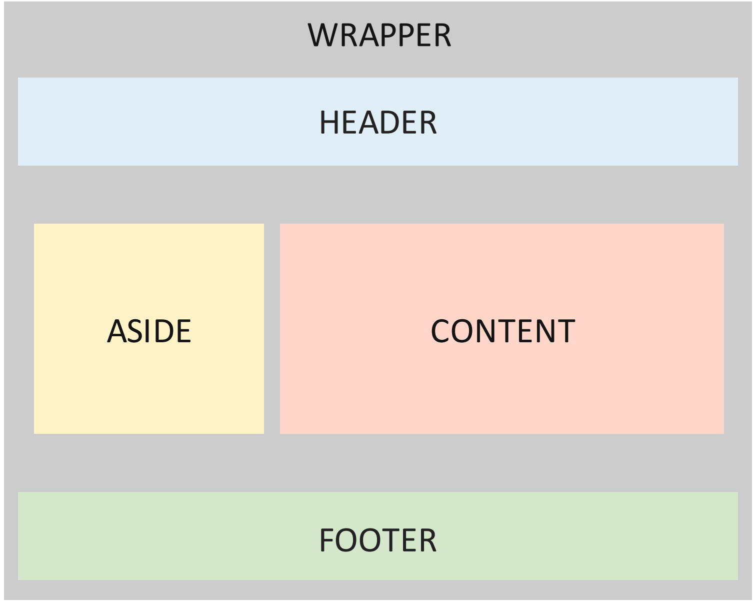
Travel & Hospitality

Services and solutions for Air, Car, Hospitality, Agencies, travel-specific distribution platforms and ISV's

About Us | Contact Us | Site Map | Careers

SERVICES	INDUSTRIES	SOLUTIONS	STRENGTHS	COMPANY	CAREERS
Product Development	ISVs & Technology	Business Intelligence	Our People	Our Customers	USA
Application Development	Banking & Financial Services	SAP Technologies	Our Partners	History	Canada
Enterprise Application Platforms	Business Information & Media	eCommerce Solutions	NearShore Advantage	Leadership	Germany
Application Testing	Travel & Hospitality	Portals & Content	Global Delivery Ecosystem	Offices Worldwide	UK
Application Maintenance & Support	Retail & CPG	Cloud/SaaS	Agile	News & Events	Sweden
ASM & Infrastructure Services	Healthcare and Life Sciences	Mobile	Innovation Labs	Awards & Recognition	Hungary
	Emerging Verticals	Business & System Analysis	Competency Centers	Investors	Poland
		Digital Strategy	Engagement Models		Armenia
		UXD			Bulgaria
		Technology Solutions			China
		Embedded Solutions			Hong Kong
					Singapore
					More...
					Mexico

© 1993-2015. EPAM Systems. All Rights Reserved. Privacy Policy



[MDN: Introduction to CSS layout]

A basic template

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <title>Document title</title>
    <link rel="stylesheet" href="css/main.css" />
   </head>

<body>
  <div id="wrapper">
    <header id="header">
      <h1 class="logo"><a href="#"></a></h1>
      <nav id="nav"></nav>
    </header>
    <main id="main">
      <section id="content"></section>
      <aside id="sidebar"></aside>
    </main>
    <footer id="footer"></footer>
  </div>
</body>
</html>
```

Positioning techniques

Techniques of building layouts and positioning content

Float Property

Floats allow elements to appear next to, or apart from, one another

Position Property

The position property provides different ways to uniquely position an element

Display Property

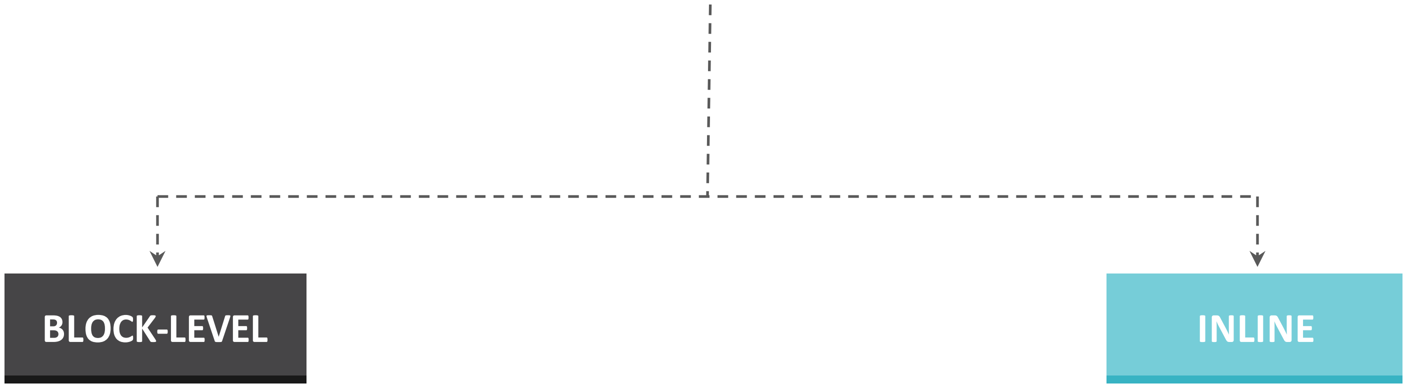
The display property specifies the type of box used for an HTML element

Z-Index Property

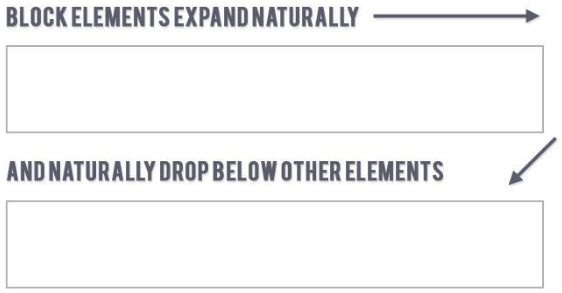
To change the order of how elements are stacked, also known as the z-axis, the z-index property is to be used

BLOCK-LEVEL VS INLINE ELEMENTS

Element types



Block-level elements


BLOCK ELEMENTS EXPAND NATURALLY
AND NATURALLY DROP BELOW OTHER ELEMENTS

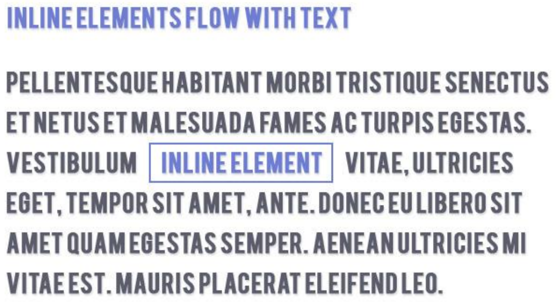
```css
#navigation a {
  display: block;
  padding: 20px 10px;
}
```

- If no width is set, will expand naturally to fill its parent container
- If no height is set, will expand naturally to fit its child elements (assuming they are not floated or positioned)
- Can have margins and/or padding
- Ignores the vertical-align property
- By default, will be placed below previous elements in the markup (assuming no floats or positioning on surrounding elements)

`<p>, <div>, <form>, <header>, <nav>, <ul>, <li>, <h1>,…`

MDN: Block-level elements

Inline elements


INLINE ELEMENTS FLOW WITH TEXT
PELLENTESQUE HABITANT MORBI TRISTIQUE SENECTUS ET NETUS ET MALESUADA FAMES AC TURPIS EGESTAS. VESTIBULUM INLINE ELEMENT VITAE, ULTRICIES EGET, TEMPOR SIT AMET, ANTE. DONEC EU LIBERO SIT AMET QUAM EGESTAS SEMPER. AENEAN ULTRICIES MI VITAE EST. MAURIS PLACERAT ELEIFEND LEO.

```
li {
    display: inline
}
```

- Flows along with text content
- Will ignore top and bottom margin settings, but will apply left and right margins, and any padding
- If floated left or right, will automatically become a block-level element, subject to all block characteristics
- Is subject to white-space settings in CSS
- Will ignore the width and height properties
- Is subject to the vertical-align property

`<a>, <span>, <b>, <em>, <i>, <cite>, <mark>, <code>,…`

MDN: Inline elements

Important elements

block-level

<div>	Document division
<p>	Paragraph
<ul>	Unordered list
<ol>	Ordered list
<li>	List item
<table>	Table
<form>	Input form
<fieldset>	Field set label
<nav>	Contains navigation links
<header>	Section or page header
<h1> - <h6>	Headings levels 1-6
<aside>	Aside content
<main>	Contains the central content
<section>	Section of a web page
<footer>	Section or page footer

inline

<a>	Anchor
<br>	Line break
<button>	Button
<em>	Emphasis
<iframe>	Inline Frame
<img>	Image Embed
<input>	Input (Form Input)
<label>	Represents a caption
<select>	Select
<span>	Generic inline container
<textarea>	Represents a multi-line plain-text editing control

MDN: HTML elements reference

MDN: Content categories

Block-level elements

- <address>
- <article>
- <aside>
- <blockquote>
- <details>
- <dialog>
- <dd>
- <div>
- <dl>
- <dt>
- <fieldset>
- <figcaption>
- <figure>
- <footer>
- <form>
- <h1>, <h2>, <h3>, <h4>, <h5>, <h6>
- <header>
- <hr>
- <li>
- <main>
- <nav>
- <ol>
- <p>
- <pre>
- <section>
- <table>
- <ul>

MDN: Block-level elements

Inline elements

<a>
<abbr>
<audio>
<b>
<bdi>
<bdo>
<br>
<button>
<canvas>
<cite>
<code>
<data>
<datalist>
<del>
<dfn>
<em>
<embed>
<i>
<iframe>
<img>
<input>
<ins>
<kbd>
<label>
<map>
<mark>
<meter>
<noscript>
<object>
<output>
<picture>
<progress>
<q>
<ruby>
<s>
<samp>
<script>
<select>
<slot>
<small>
<span>
<strong>
<sub>
<sup>
<svg>
<template>
<textarea>
<time>
<u>
<var>
<video>
<wbr>

MDN: Inline elements

Do we need to use <h1> - <h6> elements always?

Production projects here many times target a web application (such as Gmail), not a document-based site (Wikipedia).

In a web application, heading elements can be considered as non-semantic, because a single component rarely contains sections in different levels, hence there is no room for headings in a different level.

Using headings here could lead to issues when trying to relocate that component in DOM.

There are document parts, however, even in web apps, such as FAQ, Privacy Policy, Terms and Conditions. There, we do need to utilize these elements.

Also, use headings and semantic elements in document-based web sites as it helps the SEO and screen-readers.

The display property

*The **display** CSS property sets whether an element is treated as a block or inline element and the layout used for its children, such as flow layout, grid or flex.*

```
span {
    display: block;
}
```

The element generates a block element box, generating line breaks both before and after the element when in the normal flow.

```
div {
    display: inline;
}
```

The element generates one or more inline element boxes that do not generate line breaks before or after themselves. In normal flow, the next element will be on the same line if there is space.

```
a {
    display: inline-block;
}
```

Generates a block element box that will be flowed with surrounding content as if it were a single inline box.

```
section {
    display: table;
}
```

These elements behave like HTML <table> elements. It defines a block-level box. Its children can be styled using properties like

{display: table-cell}, {display: table-row} etc.

MDN: display

The display property

*Formally, the **display** property sets an element's inner and outer display types.*
The outer type sets an element's participation in flow layout; the inner type sets the layout of children.

```
p {
    display: none;
}
```

Turns off the display of an element so that it has no effect on layout (the document is rendered as though the element did not exist).
All descendant elements also have their display turned off.

```
div {
    display: flex;
}
```

The element behaves like a block element and lays out its content according to the flexbox model.

```
.container {
    display: grid;
}
```

The element behaves like a block element and lays out its content according to the grid model.

Flexbox and grid are complex areas and outside of the scope of this document.
They do have a separate modules (flexbox, grid) in the CSS standard.
They are not in the state of *Recommendation* yet, but a *Candidate Recommendation* and a *Candidate Recommendation Draft.* Still, these can be considered as stable and supported by modern browsers.

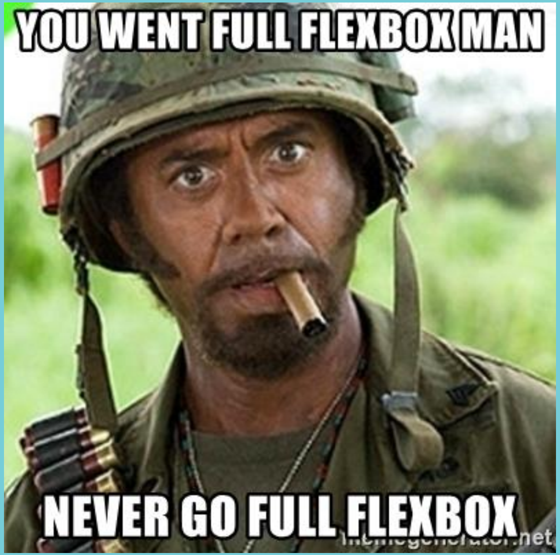
MDN: display

Do we use flexbox in projects?

Yes, flexbox is a powerful tool, however, many times it leads to more trouble, than it really helps.

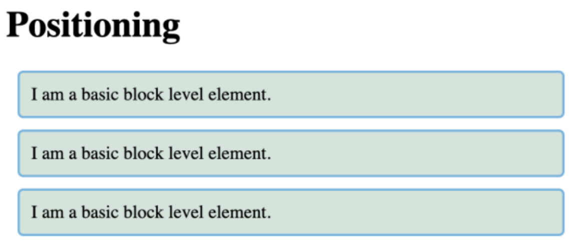
We suggest using flexbox only in the below cases:

- when it is the simplest solution for the problem
- as a last resort



POSITIONING

Basic positioning

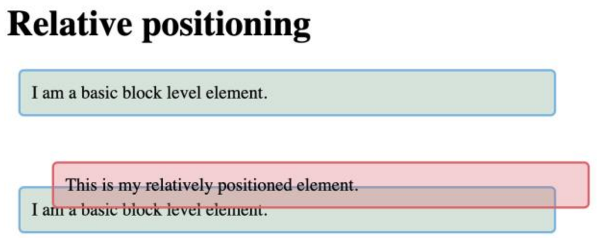


Positioning

I am a basic block level element.

I am a basic block level element.

I am a basic block level element.

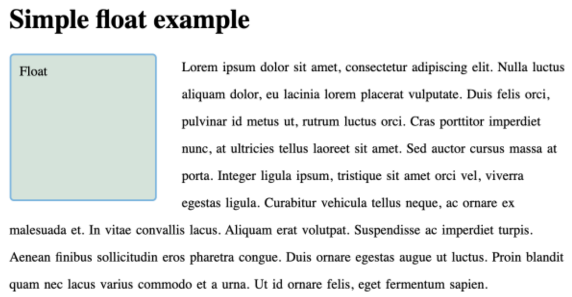


Relative positioning

I am a basic block level element.

This is my relatively positioned element.

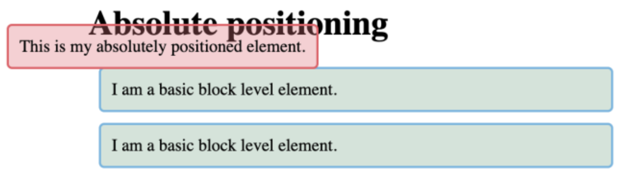
I am a basic block level element.



Simple float example

Float

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla luctus aliquam dolor, eu lacinia lorem placerat vulputate. Duis felis orci, pulvinar id metus ut, rutrum luctus orci. Cras porttitor imperdiet nunc, at ultricies tellus laoreet sit amet. Sed auctor cursus massa at porta. Integer ligula ipsum, tristique sit amet orci vel, viverra egestas ligula. Curabitur vehicula tellus neque, ac ornare ex malesuada et. In vitae convallis lacus. Aliquam erat volutpat. Suspendisse ac imperdiet turpis. Aenean finibus sollicitudin eros pharetra congue. Duis ornare egestas augue ut luctus. Proin blandit quam nec lacus varius commodo et a urna. Ut id ornare felis, eget fermentum sapien.



Absolute positioning

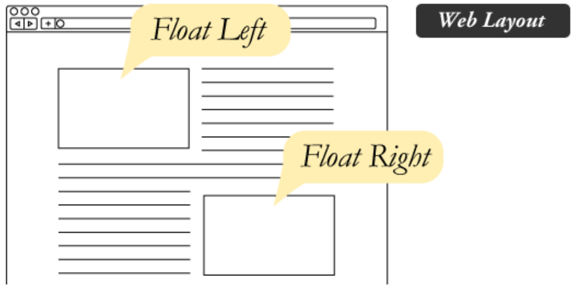
This is my absolutely positioned element.

I am a basic block level element.

I am a basic block level element.

MDN: Introduction to CSS layout

The float property

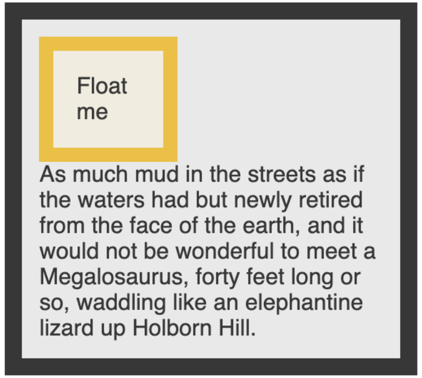


The float CSS property places an element on the left or right side of its container, allowing text and inline elements to wrap around it. The element is removed from the normal flow of the page, though still remaining a part of the flow (in contrast to absolute positioning).

As float implies the use of the block layout, it modifies the computed value of the display values

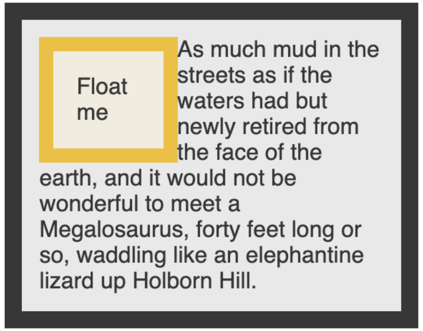
MDN: float

How it looks


Float me

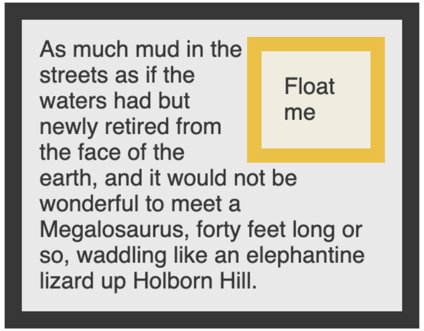
As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
div {
  float: none;
}
```


Float me

As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
div {
  float: left;
}
```


As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

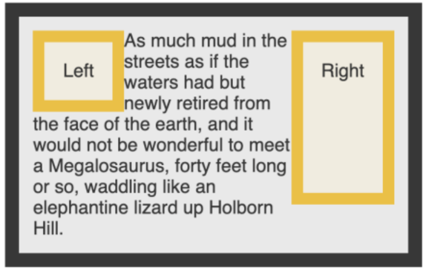
Float me

```
div {
  float: right;
}
```

[MDN: float]

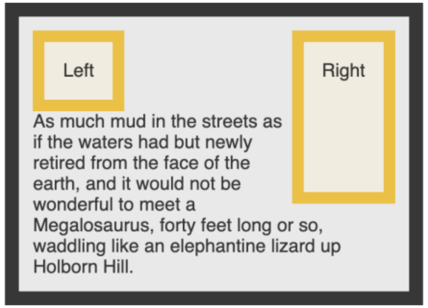
The clear property

Sometimes you may want to force an item to move below any floated elements. For instance, you may want paragraphs to remain adjacent to floats, but force headings to be on their own line.



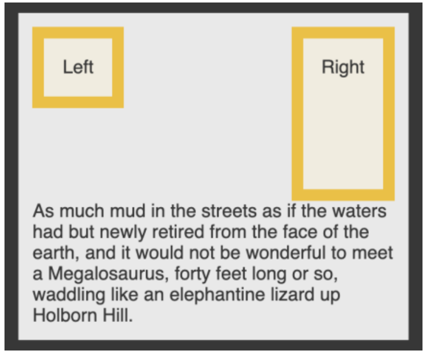
Left
Right
As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
div {
  clear: none;
}
```



Left
Right
As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
div {
  clear: left;
}
```



Left
Right
As much mud in the streets as if the waters had but newly retired from the face of the earth, and it would not be wonderful to meet a Megalosaurus, forty feet long or so, waddling like an elephantine lizard up Holborn Hill.

```
div {
   clear: right;
}
```

note: here clear:both provides the same result

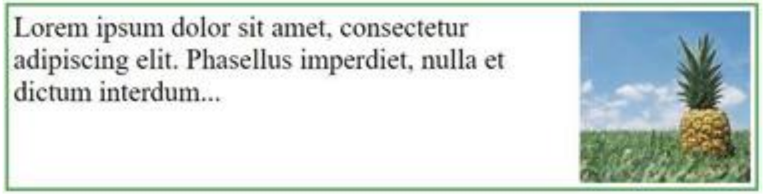
[MDN: clear]

Clearfix

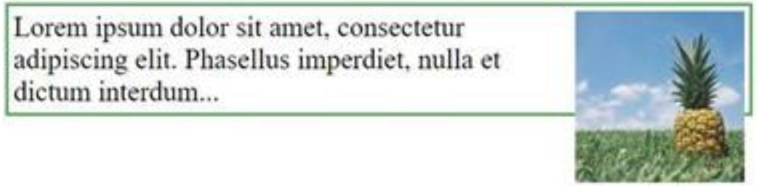
If an element contains only floated elements, its height collapses to nothing. If you want it to always be able to resize, so that it contains floating elements inside it, you need to self-clear its children.

This is called clearfix, and one way to do it is to add clear to a replaced ::after pseudo-element on it.

```
#container::after {
  content: "";
  display: block;
  clear: both;
}
```


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...

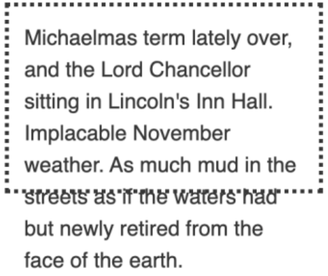
with clearfix


Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus imperdiet, nulla et dictum interdum...

without clearfix

Overflow

Content is not clipped and may be rendered outside the padding box.

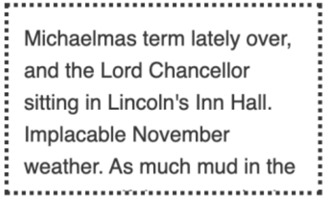


Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November weather. As much mud in the streets as if the waters had but newly retired from the face of the earth.

```
div {
  overflow: visible;
}
```

Content is clipped if necessary to fit the padding box.

No scrollbars are provided, and no support for allowing the user to scroll (such as by dragging or using a scroll wheel) is allowed.

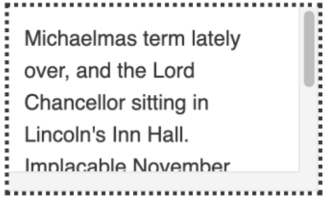


Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November weather. As much mud in the

```
div {
  overflow: hidden;
}
```

Content is clipped if necessary to fit the padding box.

Browsers always display scrollbars whether or not any content is actually clipped, preventing scrollbars from appearing or disappearing as content changes.

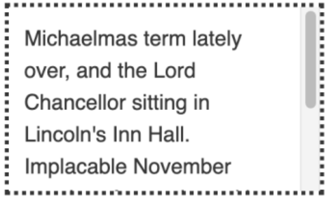


Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November

```
div {
  overflow: scroll;
}
```

Depends on the user agent. If content fits inside the padding box, it looks the same as visible, but still establishes a new block formatting context.

Desktop browsers provide scrollbars if content overflows.



Michaelmas term lately over, and the Lord Chancellor sitting in Lincoln's Inn Hall. Implacable November

```
div {
  overflow: auto;
}
```

MDN: overflow

Position property

static
The default value

absolute
Positioned relative to the first parent element that has a position other than static.

fixed
Positioned relative to the browser window

relative
Positioned relative to its normal position

sticky
Normal + offset relative to its *nearest scrolling ancestor*

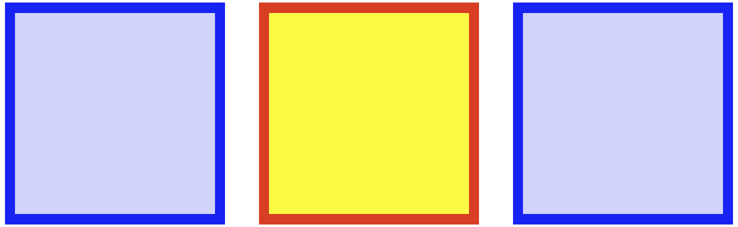
MDN: position

position: static

The element is positioned according to the normal flow of the document. The top, right, bottom, left, and z-index properties have no effect.

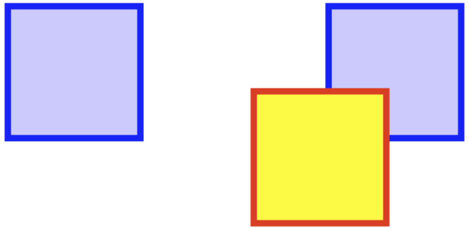
This is the default value.

```
div {
  position: static;
}
```



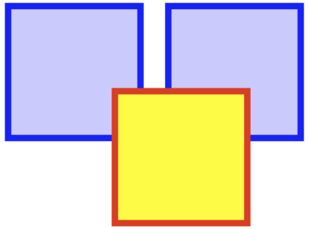
position: relative and absolute

The element is positioned according to the normal flow of the document, and then offset relative to itself based on the values of top, right, bottom, and left. The offset does not affect the position of any other elements; thus, the space given for the element in the page layout is the same as if position were static.



```
div {
  position: relative;
  top: 40px;
  left: 40px;
}
```

The element is removed from the normal document flow, and no space is created for the element in the page layout. It is positioned relative to its closest positioned ancestor, if any; otherwise, it is placed relative to the initial containing block. Its final position is determined by the values of top, right, bottom, and left.



```
div {
  position: absolute;
  top: 40px;
  left: 40px;
}
```

The chance that a question about

position: relative vs absolute

will come up in an interview is almost 100%!

(… and you need to know the answer precisely. Build many examples in different layouts until you have a strong grasp on it!)

position: fixed and sticky

```
div {
    position: fixed;
    bottom: 0;
    right: 0;
}
```

An element with position: fixed; is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled.

The top, right, bottom, and left properties are used to position the element.

A fixed element does not leave a gap in the page where it would normally have been located.

```
div {
    position: sticky;
    bottom: 0;
    right: 0;
}
```

The element is positioned according to the normal flow of the document, and then offset relative to its nearest scrolling ancestor and containing block (nearest block-level ancestor) based on the values of top, right, bottom, and left.

The offset does not affect the position of any other elements.

It's treated as relatively positioned until its containing block crosses a specified threshold (such as setting top to value other than auto) within its flow root (or the container it scrolls within), at which point it is treated as "stuck" until meeting the opposite edge of its containing block.

Remember!

An *absolute* element is positioned relative to its closest positioned ancestor.

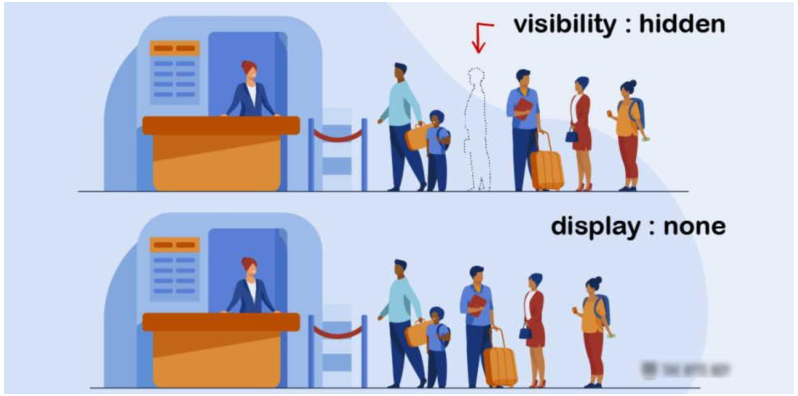
(A positioned ancestor could mean any parent, whose position is anything, except static.)

A common mistake is to believe that ~~only position: relative works there.~~

Hiding elements - display:none vs. visibility:hidden

```
.passenger {
    visibility: hidden;
}
```

```
.passenger {
    display: none;
}
```



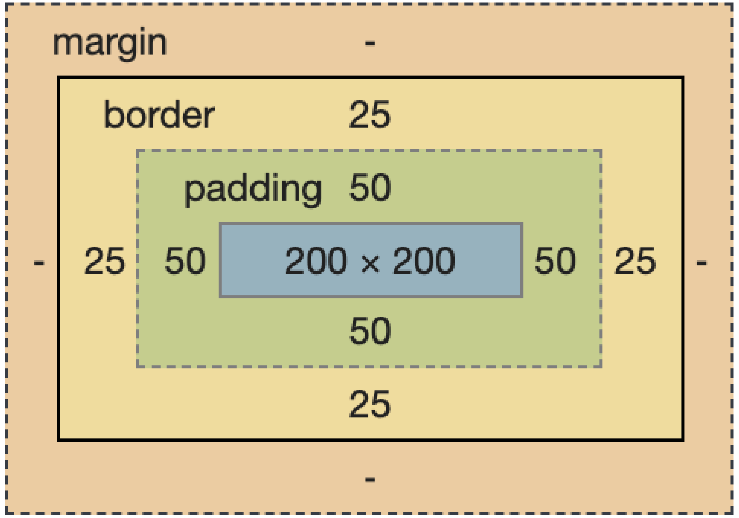
The element will be hidden, but still affects the layout.

The document is rendered as though the element doesn't exist in the document tree.

[MDN: visibility]

BOX MODEL

Box Model



Content - The content of the box, where text and images appear

Padding - Clears an area around the content. The padding is transparent

Border - A border that goes around the padding and content

Margin - Clears an area outside the border. The margin is transparent

Margin and padding: top, right, bottom, left

```
margin-top: <length> | <percentage> | auto
margin-right: <length> | <percentage> | auto
margin-bottom: <length> | <percentage> | auto
margin-left: <length> | <percentage> | auto
```

```
padding-top: <length> | <percentage>
padding-right: <length> | <percentage>
padding-bottom: <length> | <percentage>
padding-left: <length> | <percentage>
```

margins: a positive value places it farther from its neighbours, while a negative value places it closer.

Margin and padding: shorthand

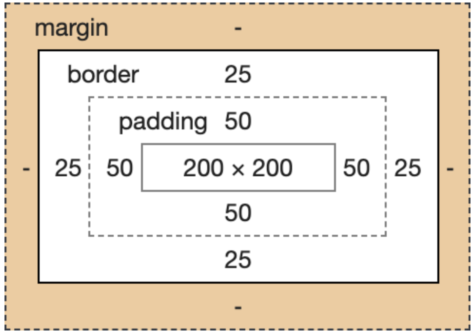
```
margin: [ <length> | <percentage> | auto ]{1,4} inherit
margin: all
margin: vertical horizontal
margin: top horizontal bottom
margin: top right bottom left
```

```
padding: [ <length> | <percentage> ]{1,4}
padding : all
padding : vertical horizontal
padding : top right bottom left
padding : top horizontal bottom
```

vertical is first!

Always starts with top.

Even when only 3 values are provided. In this case left = right.



MDN: margin

Margin and padding: shorthand

Remember: vertical is first, starts with top, and goes clockwise!

```
padding : vertical horizontal
padding : top right bottom left
```

vertical is first!

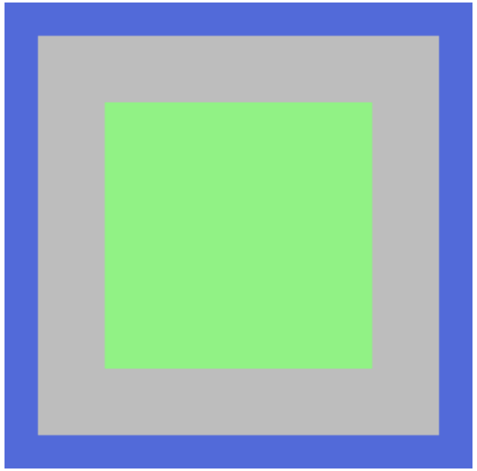
starts with top.

MDN: padding

BOX-SIZING

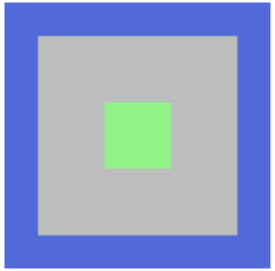
Box-sizing: different models

Same HTML, same CSS
except the box-sizing



```
div {
  box-sizing: content-box;
}
```

The default

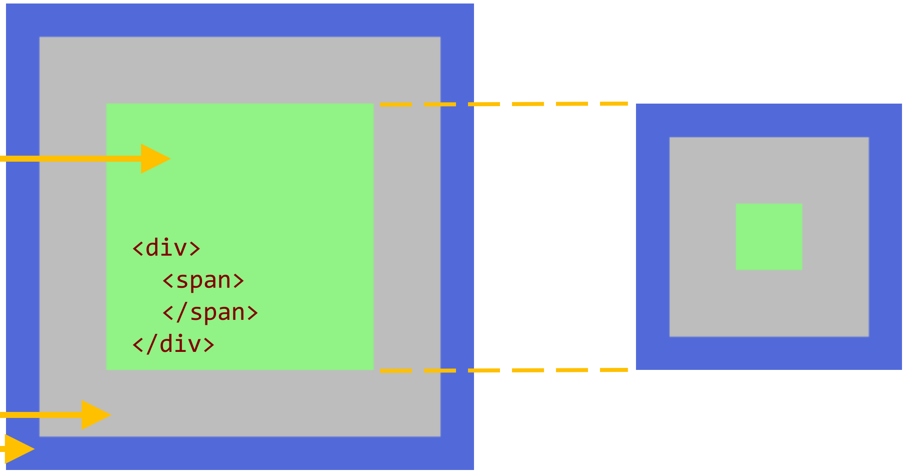


```
div {
  box-sizing: border-box;
}
```

The traditional (IE)

Box-sizing: comparison

```
span {
  display: block;
  background: springgreen;
  height: 100%;
}
div {
  box-sizing: content-box;
  background: silver;
  width: 200px;
  height: 200px;
  padding: 50px;
  border: 25px solid royalblue;
}
```



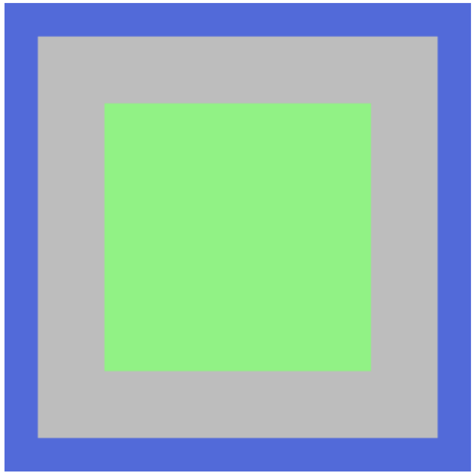
```
div {
  box-sizing: content-box;
}
```

The width and height properties includes only the content. Border and **padding are not included**.

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

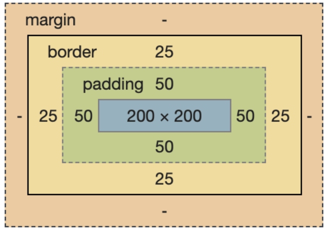
The width and height properties **includes** content, **padding and border**.

Box-sizing: content-box



```
  200px         // width
+ 2 * 50px.     // padding
+ 2 * 25px      // border
= 350px         // total width
```

```
div {
  box-sizing: content-box;
  background: silver;
  width: 200px;
  height: 200px;
  padding: 50px;
  border: 25px solid royalblue;
}
```



This is the initial and default value as specified by the CSS standard. The width and height properties include the content, but does not include the padding, border, or margin.

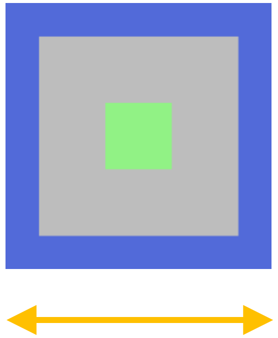
Here, the dimensions of the element are calculated as:

- *width = width of the content*
- *height = height of the content.*

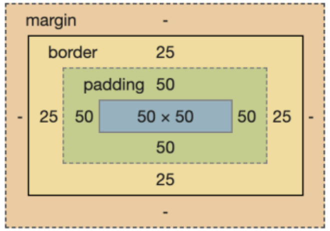
***Borders and padding are not included** in the calculation.*

MDN: box-sizing

Box-sizing: border-box



```
  200px      // width
= 200px      // total width
```



```
div {
  box-sizing: border-box;
  background: silver;
  width: 200px;
  height: 200px;
  padding: 50px;
  border: 25px solid royalblue;
}
```

The width and height properties include the content, padding, and border, but do not include the margin. Note that padding and border will be inside of the box. The content box can't be negative and is floored to 0, making it impossible to use border-box to make the element disappear.

Here the dimensions of the element are calculated as:

- *width = border + padding + width of the content*
- *height = border + padding + height of the content.*

MDN: box-sizing

Box-sizing: what to use in general?

It is often useful to set box-sizing to border-box to layout elements.

***This makes dealing with the sizes of elements much easier, and generally eliminates a number of pitfalls** you can stumble on while laying out your content.*

On the other hand, when using position: relative or position: absolute, use of box-sizing: content-box allows the positioning values to be relative to the content, and independent of changes to border and padding sizes, which is sometimes desirable.

MDN: box-sizing

What to use in projects?

In production projects always follow the project’s conventions.

MARGIN COLLAPSING

Margin collapsing

*The top and bottom margins of blocks are sometimes combined (collapsed) into a single margin whose size is the largest of the individual margins (or just one of them, if they are equal), a behavior known as **margin collapsing**.*

Note that the margins of floating and absolutely positioned elements never collapse.

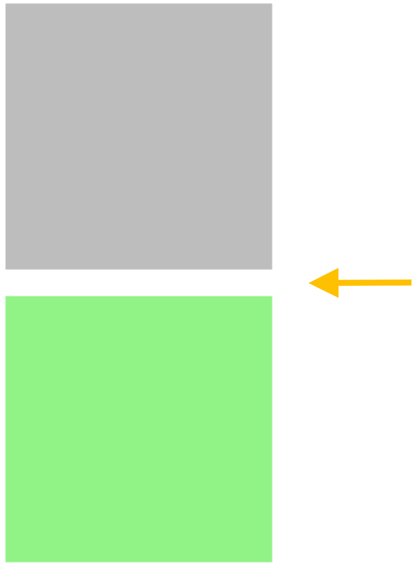
Margin collapsing occurs in three basic cases:

- *Adjacent siblings*
- *No content separating parent and descendants*
- *Empty blocks*

MDN: Mastering margin collapsing

Margin collapsing, case 1: Adjacent siblings

```
<div id="block">
</div>
<div id="adjacent">
</div>

#block {
  background: silver;
  margin-bottom: 10px;
}

#adjacent {
  background: springgreen;
  margin-bottom: 10px;
}

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



Adjacent siblings almost always collapse

You can get a rid of issues about margin collapsing by defining only top *or* bottom margins.

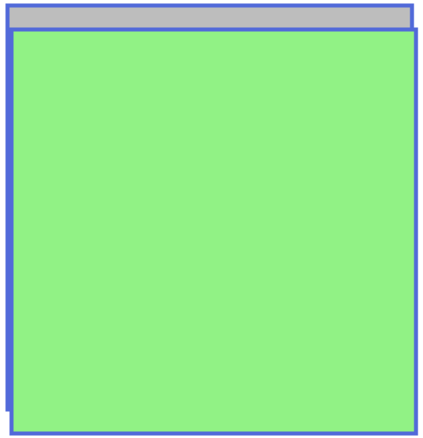
No margins to collapse, no cry ¯_(ツ)_/¯

[MDN: Mastering margin collapsing](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Mastering_margin_collapsing)

Margin collapsing, case 2: no content separating parent and descendants

```
<div id="parent">
  <div id="child">
  </div>
</div>

#parent {
  background: silver;
}

#child {
  background: springgreen;
}

div {
  width: 200px;
  height: 200px;
  margin-top: 10px;
  border: 2px solid royalblue;
}
```

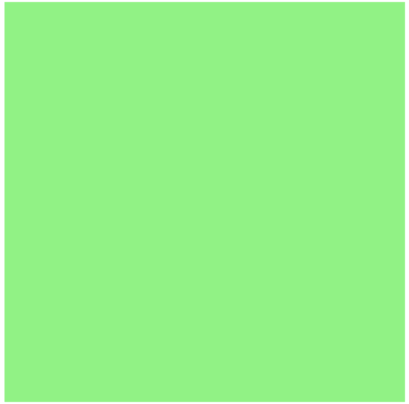
← we have a border = no collapsing



MDN: Mastering margin collapsing

Margin collapsing, case 2: no content separating parent and descendants

```
<div id="parent">
  <div id="child">
  </div>
</div>

#parent {
  background: silver;
}

#child {
  background: springgreen;
}

div {
  width: 200px;
  height: 200px;
  margin-top: 10px;
}
```



Nothing separates the parent and the descendant = margins collapsing

MDN: Mastering margin collapsing

Margin collapsing, case 2: no content separating parent and descendants

Rules are pretty complex, yet basically if something (border, padding, etc.) separates the blocks = no collapsing

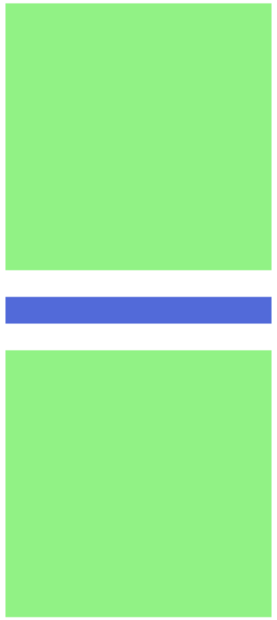
If there is no border, padding, inline part, block formatting context created, or clearance to separate the margin-top of a block from the margin-top of one or more of its descendant blocks; or no border, padding, inline content, height, min-height, or max-height to separate the margin-bottom of a block from the margin-bottom of one or more of its descendant blocks, then those margins collapse. The collapsed margin ends up outside the parent.

RTFM!

MDN: Mastering margin collapsing

Margin collapsing, case 3: Empty blocks

```
<div id="first"></div>
<div id="middle"></div>
<div id="last"></div>



#first, #last {
  width: 100px;
  height: 100px;
  background: springgreen;
}

#middle {
  width: 100px;
  height: 10px;
  margin-top: 10px;
  margin-bottom: 10px;
  background: royalblue;
}
```

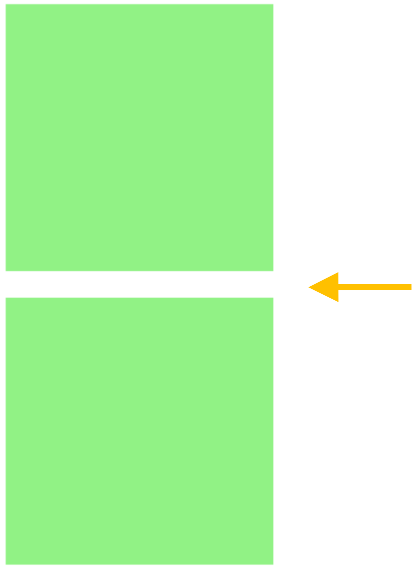


the #middle div does have height =
no collapsing

MDN: Mastering margin collapsing

Margin collapsing, case 3: Empty blocks

```
<div id="first"></div>
<div id="middle"></div>
<div id="last"></div>



#first, #last {
  width: 100px;
  height: 100px;
  background: springgreen;
}

#middle {
  width: 100px;
  margin-top: 10px;
  margin-bottom: 10px;
  background: royalblue;
}
```



empty #middle div, no height =
margins collapsing

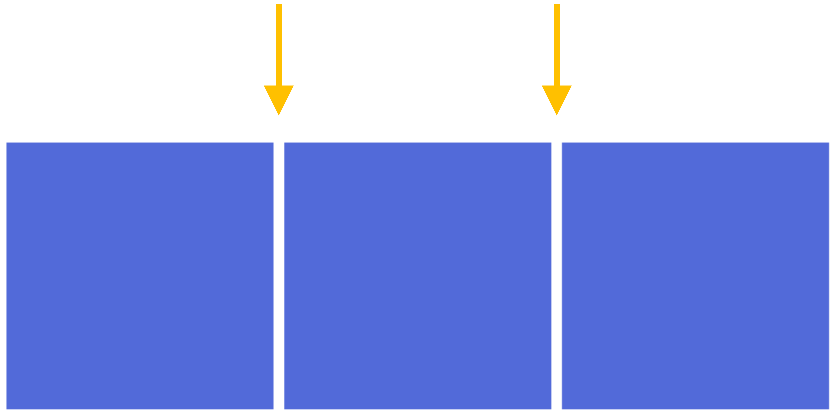
[MDN: Mastering margin collapsing]

FIGHTING WITH WHITESPACES

Inherent whitespaces

```html
<span></span>
<span></span>
<span></span>
```

```css
span {
  display: inline-block;
  background: royalblue;
  width: 100px;
  height: 100px;
}
```



MDN: Whitespace

Remove whitespaces between elements

```html
<nav>
  <a href="#">
    One</a><a href="#">
    Two</a><a href="#">
    Three</a>
</nav>

<nav>
  <a href="#">One</a
  ><a href="#">Two</a
  ><a href="#">Three</a>
</nav>

<nav>
  <a href="#">One</a><!--
  --><a href="#">Two</a><!--
  --><a href="#">Three</a>
</nav>

<nav>
  <a href="#">One
  <a href="#">Two
  <a href="#">Three
</nav>
```

Never ever do this!

Usually, we don't have a control on how DOM is assembled, also, relying on this makes the code extremely brittle.

Any changes in content could lead to bugs in design.

Remove visible whitespaces with CSS

```html
<div>
  <span></span>
  <span></span>
  <span></span>
</div>
```

```css
div {
  font-size: 0;
}

span {
  display: inline-block;
  background: royalblue;
  width: 100px;
  height: 100px;
}
```

MDN: Whitespace

No worries!

While it seems problematic to handle, it is usually a non-issue.

Either these whitespaces are needed, or elements are positioned directly (e.g., with absolute positioning)

Still, you can run into this here and there, so good to know the source of those mysterious margins.

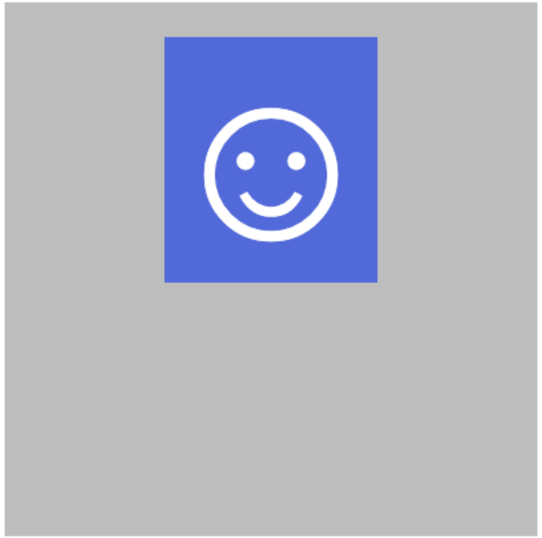
CENTERING

Centering – text-align: center and display: inline-block

```
<div id="parent">
  <div id="child"></div>
</div>



#parent {
  width: 200px;
  height: 200px;
  background: silver;

  text-align: center;
}

#child {
  font-size: 80px;
  color: white;
  background: royalblue;

  display: inline;
}
```

centers just horizontally

display: inline or inline-block

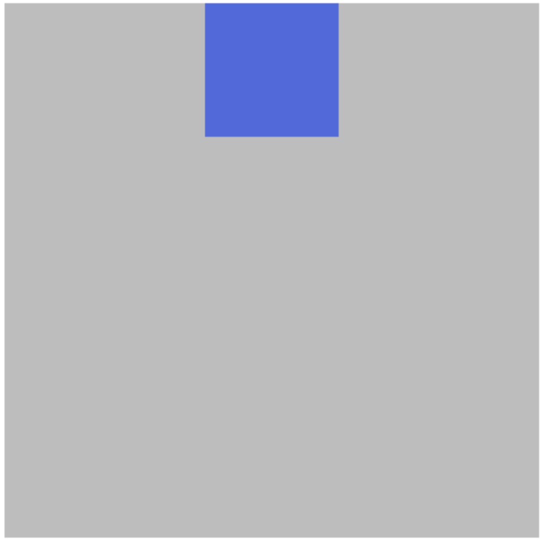


Centering – margin: auto

```
<div id="parent">
  <div id="child"></div>
</div>



#parent {
  width: 200px;
  height: 200px;
  background: silver;
}

#child {
  width: 50px;
  height: 50px;
  background: royalblue;

  margin: 0 auto;
  margin: auto;
}
```

← centers just horizontally

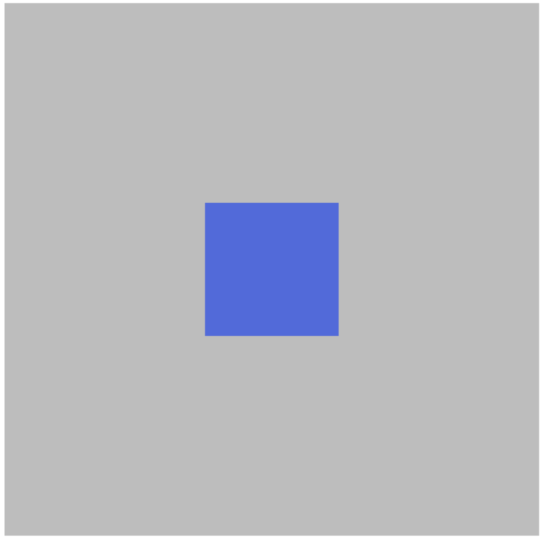
← does not center vertically, either



Centering – margin: auto

```
<div id="parent">
  <div id="child"></div>
</div>

#parent {
  width: 200px;
  height: 200px;
  background: silver;
}

#child {
  width: 50px;
  height: 50px;
  background: royalblue;

  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
}
```

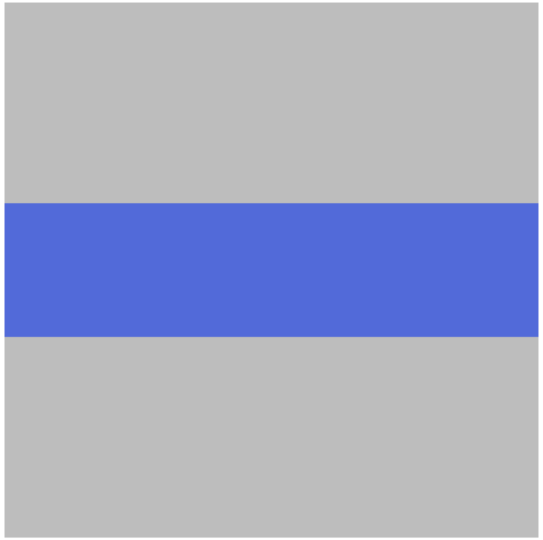
this way works in both direction,
but there are side effects!



Centering – margin: auto

```
<div id="parent">
  <div id="child"></div>
</div>

#parent {
  width: 200px;
  height: 200px;
  background: silver;
}

#child {

  height: 50px;
  background: royalblue;

  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  margin: auto;
}
```

whoops, missing width, fixed size is needed to work properly!

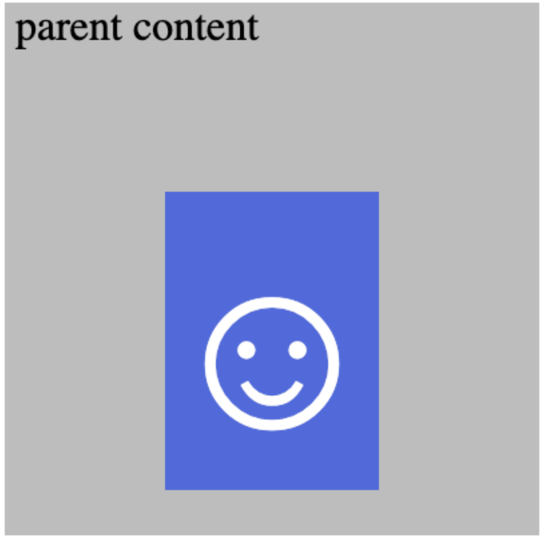


Centering – margin: auto and display: grid

```
<div id="parent">
  &nbsp;parent content<div id="child">&#9786;</div>
</div>

#parent {
  width: 200px;
  height: 200px;
  background: silver;

  display: grid;
}

#child {
  font-size: 80px;
  color: white;
  background: royalblue;

  margin: auto;
}
```

display: grid

margin: auto;



works, in a way…

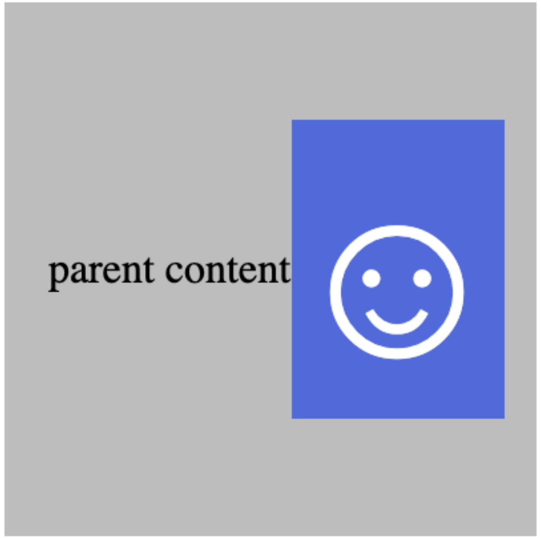
Centering – display:flex

```
<div id="parent">
  &nbsp;parent content<div id="child">&#9786;</div>
</div>

#parent {
  width: 200px;
  height: 200px;
  background: silver;

  display: flex;
  justify-content: center;
  align-items: center;
}

#child {
  font-size: 80px;
  color: white;
  background: royalblue;
}
```

horizontally

vertically

no position: absolute



works, but…

Centering – position: absolute with negative margins

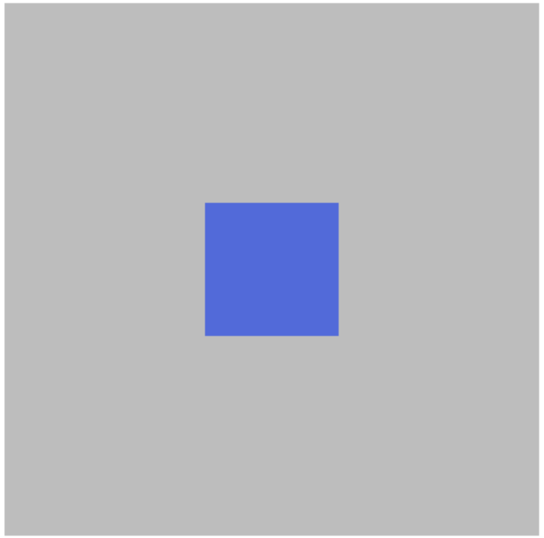
```
<div id="parent">
  <div id="child"></div>
</div>


#parent {
  width: 200px;
  height: 200px;
  background: silver;
  position: relative;
}

#child {
  width: 50px;
  height: 50px;
  background: royalblue;

  position: absolute;
  top: 50%;
  left: 50%;
  margin-top: -25px;
  margin-left: -25px;
}
```

anything but static, not just relative

absolute

50%

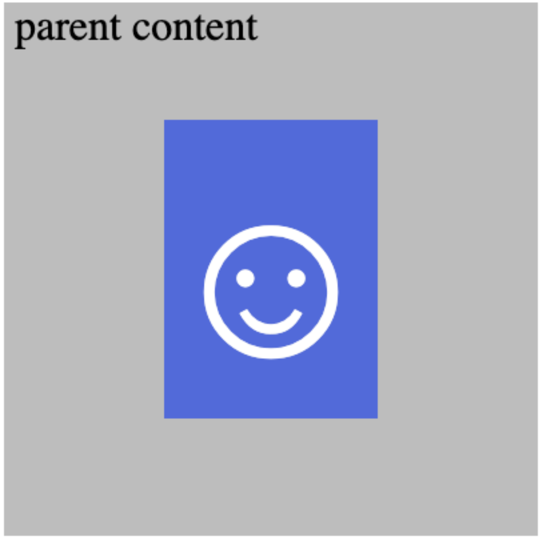
half of the sizes, works only with fixed sized elements!



Centering – position: absolute with transform

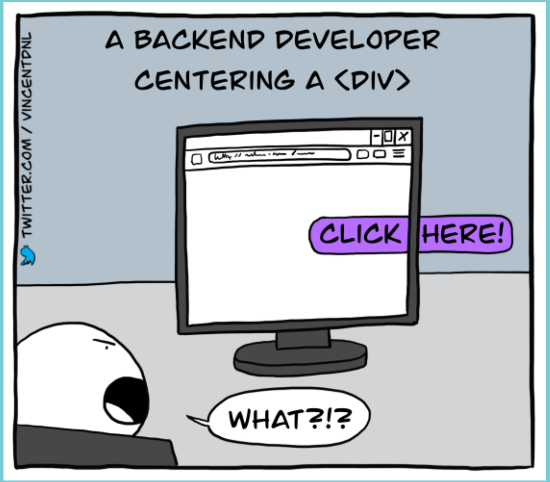
```
<div id="parent">
  &nbsp;parent content<div id="child">&#9786;</div>
</div>


#parent {
  width: 200px;
  height: 200px;
  background: silver;
  position: relative;
}

#child {
  font-size: 80px;
  color: white;
  background: royalblue;

  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}
```

always works, regardless of the size



you can use safely now

[Can I use: transform]



What to use in projects?

```
position: absolute;
top: 50%;
left: 50%;
transform: translate(-50%, -50%);
```

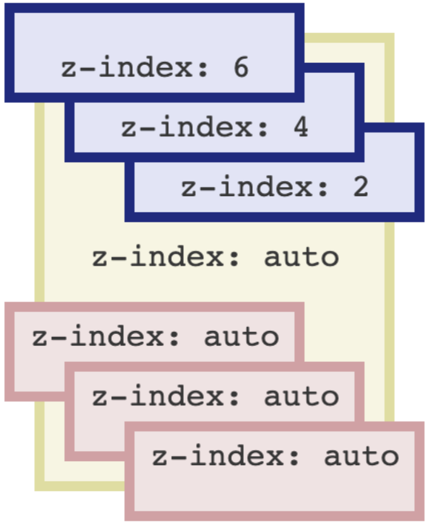
Usually in a @mixin, though

Z-index and the stacking order

z-index

The z-index CSS property sets the z-order of a positioned element and its descendants or flex items. Overlapping elements with a larger z-index cover those with a smaller one.

```
/* Keyword value */
z-index: auto;

/* <integer> values */
z-index: 0;
z-index: 3;
z-index: 289;
z-index: -1; /* Negative values to lower the priority */

/* Global values */
z-index: inherit;
z-index: initial;
z-index: unset;
```



MDN: z-index

Stacking context

The stacking context is a three-dimensional conceptualization of HTML elements along an imaginary z-axis relative to the user, who is assumed to be facing the viewport or the webpage. HTML elements occupy this space in priority order based on element attributes.

In summary:

- *Stacking contexts can be contained in other stacking contexts, and together create a hierarchy of stacking contexts.*
- ***Each stacking context is completely independent of its siblings: only descendant elements are considered when stacking is processed.***
- *Each stacking context is self-contained: after the element's contents are stacked, the whole element is considered in the stacking order of the parent stacking context.*

A stacking context is formed, anywhere in the document, by any element in the following scenarios:

- *Root element of the document (<html>).*
- *Element with a position value absolute or relative and z-index value other than auto.*
- *Element with a position value fixed or sticky.*
- *(there are many more)…*

MDN: The stacking context

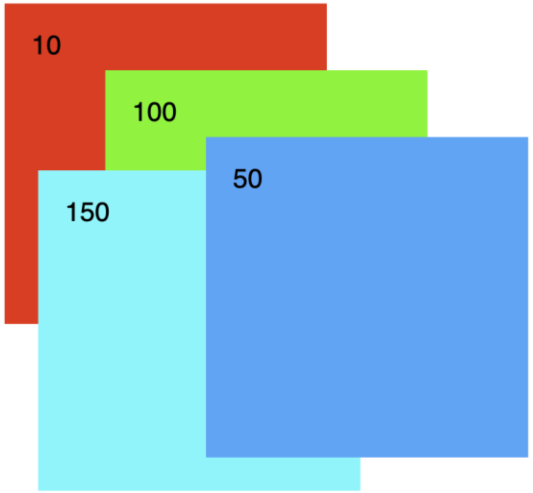
Stacking context

```
.one {
  background: #f00;
  top: 100px;
  left: 200px;
  z-index: 10;
}

.two {
  background: #0f0;
  top: 50px;
  left: 75px;
  z-index: 100;
}

.three {
  background: #0ff;
  top: 125px;
  left: 25px;
  z-index: 150;
}

.four {
  background: #0af;
  top: 200px;
  left: 350px;
  z-index: 50;
}
```

```
<div class="one">10
  <div class="two">100</div>
  <div class="three">150</div>
</div>

<div class="four">50</div>
```

Different stacking contexts

```
div {
  width: 200px;
  height: 200px;
  padding: 20px;
  position: absolute;
}
```



the .four div is on the top, despite having z-index: 50!

please don't…

When your z-index does not seem to work…

Always check the stacking context!

Maybe they are on **different ones**.

Q&A

edu_hu@epam.com