



Workshop

HTML & CSS

Topics - HTML

- [DevTips](#)
- [History of the Web](#)
- [HTML Syntax](#)
- [Basic HTML Structure](#)
- [Semantics](#)
- [Document Structure](#)

- [Text](#)
- [Anchors](#)
- [Images](#)
- [Lists](#)
- [Forms](#)

Topics

→ [HTML](#)

→ [CSS](#)

















DevTips

You don't have to know how it works,
but where to look it up.

devdocs.io

- Help for HTML, CSS and JavaScript
- Based on MDN
- PWA (available offline 🇩🇪)
- devdocs.io



Search...

- ▶  CSS
- ▶  DOM
- ▶  DOM Events
- ▶  HTML
- ▶  HTTP
- ▶  JavaScript
- ▼ DISABLED (368)
- ▶  Angular
- ▶  Angular.js
- ▶  Ansible
- ▶  Apache HTTP Server 2.4.37
- ▶  Apache Pig
- ▶  Async 3.0.1
- ▶  Babel 6.26.1
- ▶  Backbone.js 1.4.0
- ▶  Bash 5.0
- ▶  Bluebird 3.5.1

Welcome!

[Stop showing this message](#)

DevDocs combines multiple API documentations in a fast, organized, and searchable interface. Here's what you should know before you start:

1. Open the [Preferences](#) to enable more docs and customize the UI.
2. You don't have to use your mouse — see the list of [keyboard shortcuts](#).
3. The search supports fuzzy matching (e.g. "bgcp" brings up "background-clip").
4. To search a specific documentation, type its name (or an abbr.), then Tab.
5. You can search using your browser's address bar — [learn how](#).
6. DevDocs works [offline](#), on mobile, and can be installed on [Chrome](#).
7. For the latest news, follow [@DevDocs](#).
8. DevDocs is free and [open source](#).  Stars  22k
9. And if you're new to coding, check out [freeCodeCamp's open source curriculum](#).

Happy coding!

Emmet

- Popular extension for most online and offline editors
- Boosts productivity with shortcuts and abbreviations
- [Emmet cheat sheet](#)

Online editors

- Make use of online editors as playground for small and isolated problems
- Share code examples with your colleagues
- I can recommend
 - codepen.io
 - jsbin.com
 - [Codesandbox.io](https://codesandbox.io)
 - [stackblitz](https://stackblitz.com)

caniuse

- Sometimes it's hard to know which HTML or CSS feature is available in which browser
- Gives you additional hints about known issues or resources
- [caniuse](#)

DevTools in Firefox and Chrome

- DevTools have tons of features, some are obvious, but some are hidden
- Time used to get familiar with DevTools is well invested
- [Chrome DevTools](#)
- [Firefox DevTools](#)

CSS color names

<code>

No demo without fancy color names, see [list](#).

```
.dont-do-that {  
  color: salmon;  
  background-color: papayawhip;  
  border-color: lemonchiffon;  
}
```

Code Katas

- Small coding challenges that focus on a specific topic
 - Can be JavaScript, CSS or framework-related
- Can be done within a team or as a learning group
- Coding is timeboxed
- All solutions get reviewed together
 - see how your colleagues think and code
 - learn new stuff

HTML

HTML is the most basic building block of the Web.

- [MDN web docs](#)

Why / What you'll learn



- HTML & CSS are the foundation of the web and every web application
- They can be used to solve a variety of use cases without touching any JavaScript 🎉

History of the Web

... and of HTML

The web started on the computer of Tim
Berners-Lee and look what only happened in 30
years.

The birth of HTML and the Web

- [Tim Berners-Lee](#) worked at [CERN](#)
- He developed the initial version of the web
- First proposal of HTML in 1989
- First implementation of HTML, Browser and server software in 1993
- HTML stands for *HyperText Markup Language*

The idea of HTML in 1990

- Share information (not cat gifs, sorry 🐱)
- Structure information
- Link documents

The first website ever

- [Browser version](#) of the first website
- [Line mode version](#) of the first website

HTML Versions

- July 25, 1993: HTML 1.0
- November 24, 1995: HTML 2.0
- January 14, 1997: HTML 3.2
- December 18, 1997: HTML 4.0
- December 24, 1999: HTML 4.01
- **October 28, 2014: HTML5**
- Since then “HTML Living Standard”

Cool new things in HTML5

- [Audio](#)
- [Video](#)
- [Canvas](#) (Demo [#1](#) [#2](#))
- (more) [Semantic elements](#)
- [Form validations](#) (Demo [#1](#) [#2](#))

Pssst...

HTML is backwards compatible by default.

HTML Syntax

HTML syntax

- HTML is organized with tags
- Most elements have an opening and closing tag
- Some elements only have an opening tag, i.e. images

HTML syntax

<code>

Most HTML elements have an opening and closing tag.



HTML syntax

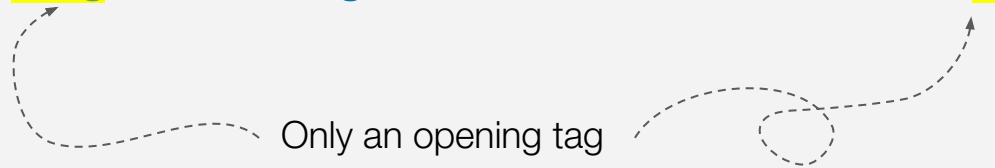
<code>

Some elements only have an opening tag. They are called “empty elements”

```

```

Only an opening tag



HTML syntax

<code>

You can write empty elements with /> at the end (“XHTML-like”)

```

```

```

```

HTML elements

- Often “HTML tags” and “HTML elements” are used interchangeable
- Each element holds content, i.e. text, images or other elements
- The element name should describe its content (more details later)

HTML elements

<code>

Elements can contain text and/or other elements

```
<h1>Welcome to my website</h1>
```

```
<article>
```

```
  <h2>Cats</h2>
```

```
  <p>I like cats 🐱</p>
```

```
  
```

```
</article>
```

Permitted content

- Some elements only allow specific content
- Usually the browser will render things nonetheless, but from a technical point of view its invalid markup
- [W3C markup validator](#)

Permitted content

- For example
 - `<u1>` elements only allow zero or more `` elements as content
 - `<p>` elements only allow phrasing content

Example for permitted content on devdocs.io

``

The `` represents an unordered list of items, typically rendered as a bulleted list.



HTML Demo: ``

Reset

HTML CSS

```
1 <ul>
2   <li>Milk</li>
3   <li>Cheese
4     <ul>
5       <li>Blue cheese</li>
6       <li>Feta</li>
7     </ul>
```

Output

- Milk
- Cheese
 - Blue cheese
 - Feta

Content categories [↗]	Flow content [↗] , and if the <code></code> element's children include at least one <code></code> element, Palpable content [↗] .
Permitted content	zero or more <code></code> elements, which in turn often contain nested <code></code> or <code></code> elements.
Tag omission	None, both the starting and ending tag are mandatory. <small>HTML / ul — DevDocs</small>
Permitted parents	Any element that accepts flow content [↗] .
Permitted ARIA roles	<code>directory</code> [↗] , <code>group</code> [↗] , <code>listbox</code> [↗] , <code>menu</code> [↗] , <code>menubar</code> [↗] , <code>radiogroup</code> [↗] , <code>tablist</code> [↗] , <code>toolbar</code> [↗] , <code>tree</code> [↗] , <code>presentation</code> [↗]
DOM Interface	<code>HTMLUListElement</code> [↗]

HTML comments

- HTML Comments are not visible to the user
- Comments can be used to leave messages for other developers
- Can be used to hide HTML from the user

HTML comments

<code>

```
<article>
  <h2>Cats</h2>
  <!-- Go away cats
  <p>I like cats 🐱</p>
  -->
</article>
```

This element is not visible to the user

```
<!-- Hey there, I'm a comment -->
```

HTML attributes

- HTML elements can have attributes
- Attributes configure or adjust elements to a specific behavior
- Most attributes are optional, some are mandatory
 - E.g. `src` attribute for `img`
- [Attribute reference on MDN](#)

HTML attributes

<code>

Syntax for HTML attributes.



HTML attributes

<code>

HTML attributes for the image element.

Attribute for the image source

```

```

Alternative text, if the image
can't be display and for screen
readers

Basic HTML Structure

Basic HTML Structure

- HTML documents usually contain four basic parts
 - A **doctype** → defines the version of HTML used for this document
 - A **html** element → holds the entire document
 - A **head** element → holds metadata
 - A **body** element → hold actual document content

Basic HTML Structure

<code>

A basic HTML skeleton.

HTML version of this document

```
<!DOCTYPE html>
```

Metadata for this document

```
<html lang="en">
```

```
<head>
```

Title of this document

```
<title></title>
```

```
</head>
```

Visible content to the user

```
<body></body>
```

```
</html>
```

HTML document is enclosed with `html` tags

DOCTYPE

- The doctype defines how the browser will handle this document
- DOCTYPE is not an HTML element
- Is the only allowed content before the `html` start tag

DOCTYPE types

- The doctype for HTML5 is `html`
- Other possible document types are
 - XHTML
 - HTML (other versions than HTML5)
 - SVG


DOCTYPE

<code>

Examples for different doctypes.

 HTML5
<!DOCTYPE html>

<!DOCTYPE html
PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

 XHTML 1.0
Transitional

<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

 SVG

`<html>` element

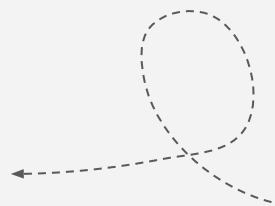
- The `html` element is the top-level element of an HTML document
- The `html` element is also called *root element* or *document element*
- Only one `html` element is allowed within a HTML document
- All other elements must be descendants of the `html` element

<html> element

<code>

The html element is the root element of all websites.

```
<!DOCTYPE html>  
<html lang="en">  
  <head></head>  
  <body></body>  
</html>
```



The lang attribute can be used to specify the language of this document.

<head> element

- The `head` element holds general information about the document
 - Document title
 - Related scripts and/or stylesheets
 - Document author
 - Document description
 - Encoding
 - Viewport

<head> element

<code>

Examples for elements within the `head` element.

```
<head>
  <title>My cat blog</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="author" content="Taylor Swift">
  <meta name="description" content="the best cats on the internet">
  <link rel="stylesheet" href="style.css">
</head>
```

<title> element

- The `title` element holds the documents title
- The title is shown in the browser's title bar and page's tab
- The `title` element can only contain text
 - All other content will be ignored

<title> element

<code>

The `title` element contains the title of the current document.

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>My cat blog</title>
  </head>
  <body></body>
</html>
```

<meta> element

- Pre-defined meta elements for
 - Description, keywords, author, viewport, ...
- Own meta information can be added using the **name** and **content** attributes

<meta> element

<code>

Examples for custom meta elements.

```
<head>  
  <meta name="google-site-verification" content="02jpioandsfvijnase">  
  <meta name="foo" content="bar">  
</head>
```

Meta tag used by Google for site verification

Own (pointless) meta tag

<body> element

- The **body** element holds the documents visible content
- There can only be one **body** element in a document

<body> element

<code>

The `body` element contains the actual content of a website.

```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <h1></h1>
    <p></p>
  </body>
</html>
```

Task

Website Police



Semantics

The meaning of HTML elements

“Semantic HTML elements clearly describe it’s meaning in a human and machine readable way.”

- freecodecamp.org

Why / What you'll learn



- Accessibility
 - WCAG defines accessibility rules and levels
 - Common technical requirement
 - Screen readers
- SEO
 - search engine optimization
 - Google Ranking = 💰💰💰

Semantic elements

- An element should describe its content
- There are around 100 [semantic elements](#)
- Generic elements are also available for when no semantic element matches
 - Before using a semantic element wrong, consider using a generic element instead

Document Structure

Document structure

- *Document structure elements* organize the main parts of a document
- Commonly used elements for document structure are
 - `<header>`, `<footer>`
 - `<main>`
 - `<aside>`
 - `<nav>`
 - `<article>`
 - `<section>`
 - `<search>`
 - `<div>`


<header> element

<code>

The header element represents the page's header.

```
<body>
  <header>
    
    <h1>...</h1>
    <nav>...</nav>
  </header>
</body>
```

A header usually contains a logo, title and navigational links.

The diagram shows a code block with HTML tags. The opening <header> and closing </header> tags are highlighted in yellow. Three dashed arrows originate from the text 'A header usually contains a logo, title and navigational links.' and point to the tag, the <h1> tag, and the <nav> tag respectively.

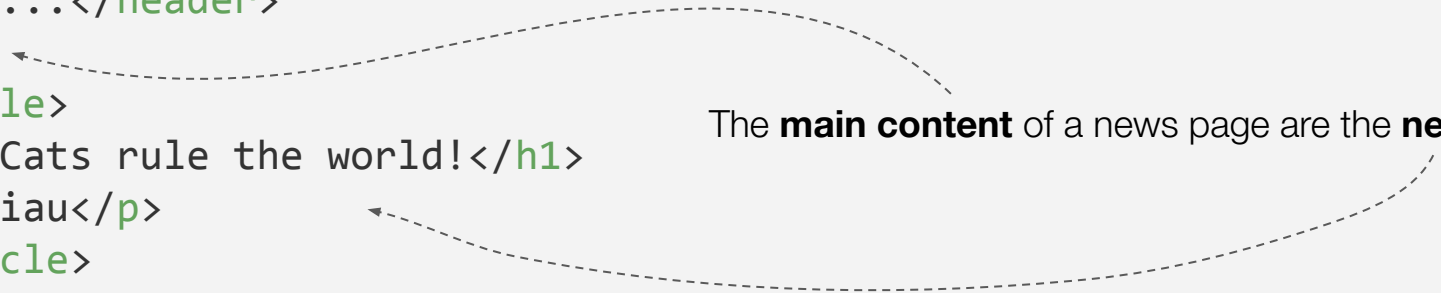
<main> element

<code>

The `main` element contains the primary content of the document.

```
<body>
  <header>...</header>
  <main>
    <article>
      <h1>Cats rule the world!</h1>
      <p>Miau</p>
    </article>
    <article>...</article>
  </main>
</body>
```

The **main content** of a news page are the **news**.



<aside> element

<code>

The `aside` element represents secondary content (that is not required to understand the main content).

```
<body>
  <header>...</header>
  <main>...</main>
  <aside>
    <section>
      <h1>Buy the new vacuum!</h1>
      
    </section>
    <nav>Content Navigation</nav>
  </aside>
</body>
```

The `aside` element can hold **ads** or a **secondary navigation** for the main content.

<footer> element

<code>

The `footer` element represents a footer for the nearest sectional element.

```
<body>
  <main>
    <article>
      ...
      <footer>Written by Ernest Hemingway</footer>
    </article>
  </main>
  <footer>
    <a href="imprint.html">Imprint</a>
    <a href="contact.html">Contact us</a>
    <p>Copyright 2023 - Cat Empire</p>
  </footer>
</body>
```

When placed inside other elements, i.e. articles, it represents the footer of this particular piece of content.

When placed inside body, the footer represents the websites footer area.

<nav> element

<code>

The nav element provides navigational content.

```
<header>
  <nav>
    <a href="famous-cats.html">Famous cats</a>
  </nav>
</header>
<aside>
  <nav>
    <a href="#spacecats">Cats in Space</a>
  </nav>
</aside>
<main>
  <article id="spacecats"></article>
</main>
```

When placed inside the documents header it represents the websites navigation.

When placed inside aside element, it can act as a sub navigation for the current document.

<article> element

<code>

The `article` element represents a piece of content that is independent.

```
<header>...</header>
```

```
<main>
```

```
  <article>
```

```
    <h1>Cats in Space</h1>
```

```
    <section>...</section>
```


```
    <section>...</section>
```

```
    <footer>....</footer>
```

```
  </article>
```

```
</main>
```

```
</footer>...</footer>
```



This article can be moved out of this document and would still make sense on its own.

Heading elements

- Headings are represented by six hierarchical elements
 - `<h1>`, `<h2>`, `<h3>`, `<h4>`, `<h5>`, `<h6>`
- Do not skip heading levels for accessibility reasons ⚠
- Technically spoken there can be multiple **h1** headings
 - From a logical point of view, **there should only be one**

<h*> elements

<code>

The heading element represents the page's header.

```
<header>
```

```
  <h1>The CAT site</h1>
```

```
</header>
```

```
<article>
```

```
  <h2>How to get a fluffy cat?</h2>
```

```
  <section>
```

```
    <h3>Shampoo</h3>
```

```
  </section>
```

```
</article>
```

h1 represents the document title.

An article can contain different lower heading levels to structure the content.

Task

Get familiar with codesandbox



Task

Basic Structure



Text

Text elements

- Text elements are used to structure the content of the website
- Commonly used elements are
 - `<p>`
 - `<blockquote>`, `<q>`
 - ``, ``, `<i>`, `<small>`
 - `<abbr>`, `<address>`
 - `
`

Text elements

- Text inside these text elements will be rendered differently
- Screen readers will read specific words with different emphasis

<p> element

<code>

The p element represents a paragraph of content, i.e. text and images.

```
<article>
  <h2>How to get a fluffy cat?</h2>
  <p>This is the ultimate guide...</p>
  <section>
    <h3>Shampoo</h3>
    <p>The used shampoo...</p>
    <p>We recommend the Fluffy Master 3000 Shampoo
      
    </p>
  </section>
</article>
```

A paragraph can be used as the sub heading.


Paragraphs structure the content of the document into meaningful pieces.

<blockquote> element

<code>

The `blockquote` element represents a extended quotation.

```
<section>
  <h3>Shampoo</h3>
  <p>The used shampoo is the most important...</p>
  <blockquote>
    <p>Fluffy Master 3000 is the best!</p>
    <footer>- Anonymous cat person</footer>
  </blockquote>
</section>
```



Can be used to mark the most important quote in this article / section.

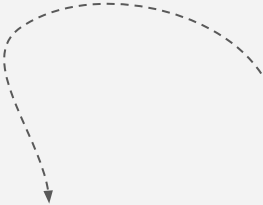
<q> element

<code>

The q element represents a short inline quotation.

```
<section>
  <h3>Shampoo</h3>
  <p>
    When it comes to fluffy cats,
    <q>The used shampoo is the most important thing</q>
    says the famous cat coiffeur Luigi.
  </p>
</section>
```

The q element is used for short inline quotations inside a paragraph of text.



 element

<code>

The `strong` element marks content as strong importance (as in warning or seriousness).

```
<p>
```

```
  The most important rule:
```

```
  <strong>Don't feed your gremlin after midnight</strong>.
```

```
</p>
```

 element

<code>

The `em` element marks words with high emphasis.

```
<p>  
  Get your bottle of Fluffy Master 3000 <em>now</em>!  
</p>
```


<i> element

<code>

The `i` element marks text as technical terms, foreign languages or thoughts.

```
<p>  
  <i>The cat has never been fluffier</i> thought the cat person  
  while she pets her lovely Siam.  
</p>
```

<small> element

<code>

The `small` element will render text one size smaller than the surrounding text, it represents side-comments.

```
<p>
```

```
  Use Fluffy Master 3000 and get the fluffiest cat on earth.
```

```
  <br>
```

```
  <small>The result highly depends on your cat.</small>
```

```
</p>
```

<abbr> element

<code>

The `abbr` element represents an abbreviation or acronym.

```
<p>  
  Use <abbr title="Fluffy Master 3000">FM3K</abbr> for a fluffy cat.  
</p>
```



The `abbr` element can be combined with the `title` attribute.

<address> element

<code>

The `address` element marks text as contact information, i.e. physical address, email address, phone or social media.

```
<p>  
  Get your Fluffy Master 3000 here:  
  <br><br>  
  <address>  
    Fluffy Master Inc.<br>  
    Castrop-Rauxel<br>  
    Germany  
  </address>  
</p>
```


 element

<code>

The br element creates a line break.

```
<p>  
  0ans, <br>  
  zwoa, <br>  
  drei, <br>  
  g'suffa!  
</p>
```

Anchors

Anchors

- Anchors are used to link to documents
- The destination is defined with the `href` attribute

<a> element

<code>

Anchor tags are primarily used to link to other documents.

```
<a href="contact.html">Contact Form</a>
```

```
<a href="imprint.html">Imprint</a>
```

```
<!-- external website -->
```

```
<a href="https://workshops.de/">Workshops.de</a>
```

```
<!-- external website in new tab -->
```

```
<a href="https://workshops.de/" target="_blank">Workshops.de</a>
```


<a> element - rel attributes

<code>

Anchor tags are primarily used to link to other documents.

```
<!-- Search engine should not follow the link -->  
<a href="https://example.com" rel="nofollow">Advertisement</a>
```

```
<!-- external website with rel for security reasons-->  
<a href="https://example.com" rel="noopener noreferrer">External Site</a>
```

noopener: Browser should not grant the new browsing context access to document, that opened it

noreferrer: Browser should omit the referer-header

<a> element

<code>

Anchor tags can be used to add links to specific parts of the same document.

```
<a href="#contact">Contact Form</a>
```

...

```
<article id="contact">  
  <form>...</form>  
</article>
```

<a> element

<code>

It is also possible to link specific parts of external documents.

```
<a href="whatever.html#contact">Contact Form</a>
```

<a> element

<code>

It is also possible to link to different document types or to mails.

```
<a href="whatever.pdf">pdf document</a>
```

```
<a href="mailto:you@example.com">mail link</a>
```

Images

Images

- Images can be embedded in documents using the `img` element
- By default images will be displayed in its full resolution, i.e. 400px x 400px

 element

<code>

Images can be embedded with the `img` element using the `src` attribute.

```
 <!-- XHTML Syntax -->
```

```
 <!-- HTML Syntax -->
```

 width and height

<code>

Width and height should be used to avoid layout shifts.

```

```

Demo and explanation: <https://web.dev/articles/cls>

max-width of images

<code>

A commonly used technique to restrict the width of images is to set the `max-width` to 100% of the containing block in CSS.

```
img {  
  max-width: 100%;  
}
```

alt attribute and images

<code>

Images can be embedded with the `img` element using the `src` attribute. Screen readers will skip images with empty alt attributes.

```

```

```

```

loading attribute - good for performance

<code>

Image should only be loaded, when the image is in the viewport -> good for images below the fold and other embeds

```

```

Image with srcset and sizes attributes

- HTML5 introduced the `srcset` and `sizes` attributes
- These are used to give the browser a list of images for different screen sizes
- The browser will only load the required image, i.e. on mobile

Picture element

- HTML5 also introduced the `picture` element for art direction
- This element allows to show different images (eg. more or less background or different formats) based on the current situation
- [Article about images in HTML5](#)

Picture element

<code>

```
<picture>  
<source srcset="katze_auf_treppe.jpg" media="(min-width: 700px)">  
  
</picture>
```

Working example
can be found [here](#).

srcset and sizes attributes

<code>

The `srcset` and `sizes` attributes allow to show different pictures depending on the current viewport.

```

```

Working example
can be found [here](#)

srcset and sizes attributes

<code>

The `srcset` attribute allows to show different pictures depending on the resolution.

```

```

This is used for high resolution screens

Working example
can be found [here](#).

Figure and figcaption

<code>

For images with captions - but you can use it for captions of tables/code etc. too

```
<figure>  
    
  <figcaption>A cat in the garden</figcaption>  
</figure>
```

Lists

Lists

- Lists in HTML are represented with ordered and unordered lists
- Lists can contain zero or more list item elements
- Lists are used for navigation bars

List appearance

- Ordered lists usually will have ordering numbers before each item
- Unordered lists usually will have bullet points before each item

 element

<code>

The `ol` element represents an ordered list.

```
<h2>Best bands in the world</h2>
<ol>
  <li>Queen</li>
  <li>Peter Maffay</li>
  <li>Haftbefehl</li>
</ol>
```

 element

<code>

The ul element represents an unordered list.

```
<h2>Some cool names</h2>
<ul>
  <li>Peter</li>
  <li>Paul</li>
  <li>Mary</li>
</ul>
```

Forms

Forms

- Forms are the common use case of business applications
- Forms can contain the following form elements
 - Text inputs, radio buttons, checkboxes, dropdowns and submit buttons

Form element

- A `form` element groups all form controls of a form
- It usually contains a `action` attribute that points to a API or endpoint to which the form is sent
- It can also have a `method` attribute which defines how the information will be sent
 - Either POST or GET

Form element

<code>

The element groups the whole form and has information about the endpoint to which the form is sent.

```
<form action="/api" method="post">  
  ...  
</form>
```

Label element

- `label` elements are used to label form elements
- Labels should use the `for` attribute, because it defines which form control is using the label
- The form controls can also be wrapped inside a label

for and id attributes

<code>

The `for` attribute of the label points to the `id` attribute of the form input.

```
<label for="name">Name</label>  
<input type="text" id="name" name="name">  
  
<label for="email">Email  
  <input type="email" id="email" name="email">  
</label>
```

for and id attributes

<code>

Clicking labels with a valid `for` attribute will focus the input or change the value.

```
<!-- Clicking the name label will focus the input -->  
<label for="name">Name</label>  
<input type="text" id="name" name="name">
```

```
<!-- Clicking the terms label will toggle the checkbox -->  
<input type="checkbox" id="terms">  
<label for="terms">I agree to the terms</label>
```

Radio buttons

<code>

Only one radio button in a given group can be selected at the same time

```
<!-- A group of radio buttons must have the same value for name -->
```

```
<input type="radio" id="red" name="color" value="red" >  
<label for="red">red</label>
```

```
<input type="radio" id="blue" name="color" value="blue" >  
<label for="blue">blue</label>
```

Input element

- The input element can represent different input types
- Input types enable browser validation and restrict the input itself
- Depending on the input type mobile users will see different keyboards

Input types

- Commonly used input types are
 - text
 - email
 - password
 - number
 - tel
 - date
 - color
 - range

Input types

<code>

Browsers will allow only specific values for different input types.

```
<input type="text" />  
<input type="email" />  
<input type="number" />  
<input type="checkbox" />
```

Textarea element

- Textarea elements are used for multiline input
- The textarea can be resized by default
- The initial size can be set with the `cols` and `rows` attributes

Textarea element

<code>

Textarea elements are used for multiline input.

```
<textarea name="message" id="message" cols="30" rows="10"></textarea>
```



define the visible dimensions of the textarea

Textarea element

<code>

Resize of textarea elements can be disabled using the `resize` property in CSS.

```
textarea {  
  resize: none;  
}
```

select list

<code>

Create a select list with a select element and some option elements

```
<select name="country">  
  <option value="de">Germany</option>  
  <option value="at">Austria</option>  
  <option value="fr">France</option>  
</select>
```

Input attributes

- HTML5 defines attributes for inputs that makes form handling extremely comfortable, commonly used attributes are
 - min and max
 - step
 - minlength and maxlength
 - required
 - placeholder
 - autofocus

min and max attributes

<code>

The `min` and `max` attributes can set a input range for numbers.

```
<input type="number" min="0" max="10" step="2" required/>
```

minlength and maxlength attributes

<code>

The `minlength` and `maxlength` attributes can limit the number of characters.

```
<input type="text" minlength="5" maxlength="20" required/>
```


placeholder attribute

<code>

The `placeholder` attribute can be used to display a preview text inside a form control.

```
<input type="text" placeholder="What's your name?"/>
```

Submit and reset


- A submit button is used to send the form
- A reset button is used to reset a form
- Both buttons use the `type` attribute with either the `submit` or `reset` value
- It is possible to use `input` or `button` elements

Submit and reset buttons

<code>

The send button will submit a form and the reset button will reset all values.

```
<form>
  <input type="text" placeholder="What's your name?">
  <button type="submit">Send</button>
  <button type="reset">Reset</button>
</form>
```



These were the most basic and important things about HTML, there are many more elements and things to learn.

More interesting elements to look at

- dialog ([MDN](#)) for modals and more
- details/summary ([MDN](#)) - they can be expanded by click (no js needed)
- video ([MDN](#))
- table ([MDN](#))

Task

Food Blog Content



CSS

Topics CSS

- [Short History of CSS](#)
- [CSS syntax](#)
- [Add CSS to HTML](#)
- [CSS Box Model](#)
- [Typography](#)
- [Basic Selectors](#)
- [Advanced Selectors](#)

- [The Cascade](#)
- [Inheritance](#)
- [Reset, Normalize ...](#)
- [BEM](#)
- [Display Property ...](#)
- [Position](#)
- [Flexbox](#)
- [Aligning and centering](#)
- [Length Units](#)
- [@-Rules](#)

Topics CSS - 2

- [Grid](#)
 - [Grid advanced](#)
 - [Transitions](#)
 - [Transform](#)
 - [CSS Animations](#)
 - [CSS Custom Properties](#)
 - [Background](#)
- [Containing Block](#)
 - [Overflow](#)
 - [CSS Multi-columns](#)
 - [Stacking Context, z-index](#)

While HTML transports information, CSS adds visual interest and prettiness to a document.

Why / What you'll learn



- HTML defines structure and content
- Default browser styling is quite pure
- Separation of concerns
 - HTML for content
 - CSS for styling and visuals

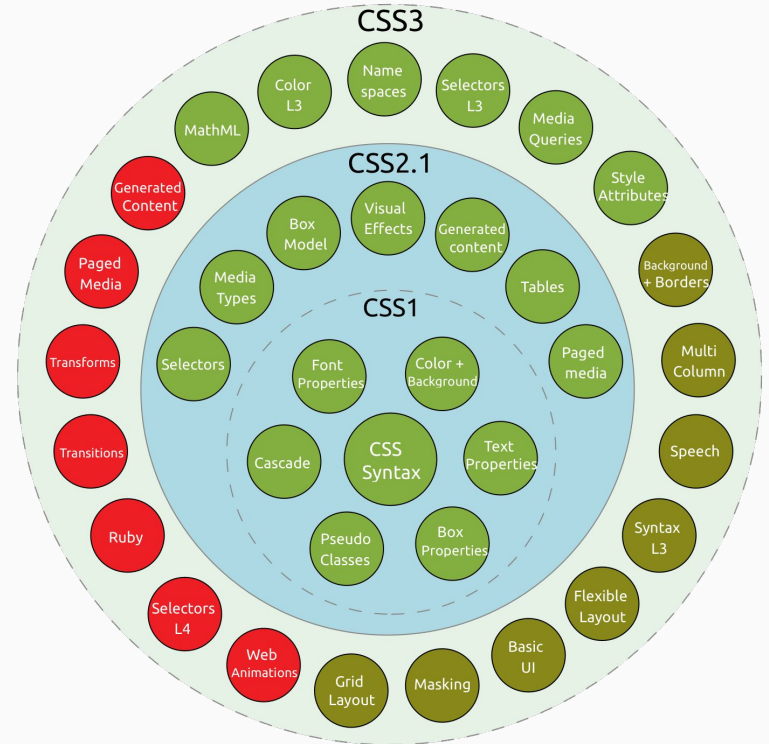
Short History of CSS

CSS version history

- December 1996 → CSS Level 1
- May 1998 → CSS Level 2
- June 1998 → First drafts for CSS Level 3 modules
- For a long time it was planned that there would be no more versioning according to CSS3. In the meantime, versioning is being planned again and there is talk of CSS4 and CSS5
(<https://github.com/CSS-Next/css-next>)

CSS modules

- CSS is developed and released in modules
- Examples for CSS modules
 - Visual effects
 - Color + Background
 - Selectors
 - Media Queries
 - Grid Layout



CSS syntax

CSS ruleset

- CSS is defined by **rulesets**
- Each **ruleset** consists of a **selector** and **style declarations**
- Each **style declaration** is made of a CSS **property** and a **value**

CSS ruleset

<code>

Instead of selector, property and value think of **who, what and how**

```
/* Super technical */  
selector {  
    property: value;  
}
```

```
/* Easy to remember */  
who {  
    what: how;  
}
```

Ruleset

Selector

```
.body {
```

```
background: hotpink;
```

Declaration

Property

Value

```
}
```

Shorthand properties

- Shorthand properties allow to set values of multiple properties at the same time
- Can *be convenient* with one property and *risky* with another property

Shorthand properties

<code>

Shorthand usage for the padding property.

```
/* Shorthands */  
padding: 10px 20px;  
/* top, right, bottom, left */  
padding: 10px 20px 10px 20px;
```

```
/* Non shorthand variant */  
padding-top: 10px;  
padding-right: 20px;  
padding-bottom: 10px;  
padding-left: 20px;
```

💡 The four value padding shorthand will set padding values clockwise starting with padding-top

Comments

<code>

Stuff inside `/* */` will become a CSS comment.

```
/* TODO: take over the world */
```

```
body {  
  background-color: hotpink; /* Gosh I love this color */  
}
```

Comments

<code>

Livehacks: Single declarations can be “disabled” with an underscore (or any prefix that results in an invalid property name).

```
body {  
  // color: red;  
  _background-color: hotpink; /* LIFEHACK! 🚀 */  
}
```

There are no single line comments in CSS 😬

Add CSS to HTML

Add CSS to your HTML

- CSS can be added to HTML in various ways
 - Using the `<link>` element specifying an external file
 - Directly inside a `<style>` element inside the `<head>` element
 - With a `style` attribute on an element
 - With `@import` in a stylesheet (not recommended)

Add CSS to your HTML

<code>

Three different ways to add CSS to HTML.

```
<head>  
  <link rel="stylesheet" href="style.css">  
  
  <style>  
    body { background-color: hotpink; }  
  </style>  
  
</head>  
<body style="background-color: khaki;">  
</body>
```

CSS as external resource

CSS inside `style` element

CSS inside `style` attribute

The diagram illustrates three methods for adding CSS to HTML. It features a code block on the left with three distinct sections highlighted in yellow. On the right, three text labels are connected to these sections by dashed arrows. The first label, 'CSS as external resource', points to the <link> tag. The second label, 'CSS inside style element', points to the <style> block. The third label, 'CSS inside style attribute', points to the style attribute on the <body> tag.

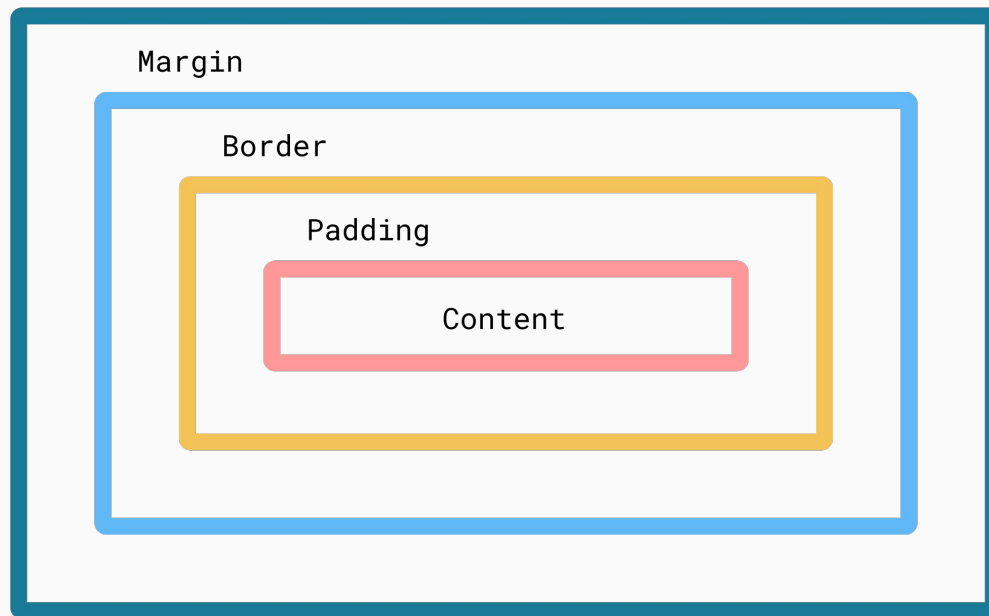
CSS Box Model and box-sizing

The box model is the blueprint of layout on the web.

CSS Box Model

- Every element is made out of the box's
 - Content
 - Padding (defined by the **padding** property)
 - Border (defined by the **border** property)
 - Margin (defined by the **margin** property)
- [Box Model](#) on MDN

CSS Box Model



Box Model in DevTools

models
, and borders
nd inline boxes
ne-block

CS
s start here!

understanding of how it works and the terminology that relates to it.
table.learn-box.standard-table 672 x 204

Prerequisites: Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study Introduction to HTML), and an idea of how CSS works (study CSS first steps.)

Objective: To learn about the CSS Box Model, what makes up the box model and how to switch to the alternate model.

Elements Console Sources Network Performance Memory >> 1

```
<table class="learn-box standard-table"></table> == %0
```

#react-container main div #content article#wikiArticle table.learn-box.standard-table

Event Listeners DOM Breakpoints Properties Accessibility Z-Index

:hov .cls +

```
{
```

```
react-mdn.f...61f1b.css:1
```

```
.color: #f69d3c;
```

```
background-color: #ffe8d4;
```

```
react-mdn.f...61f1b.css:1
```

```
text-align: left; text-content: .learn-box,
```

The diagram illustrates the CSS Box Model for the selected element. It shows four nested rectangular areas:

- margin:** The outermost area, colored orange, with a total height of 20 pixels.
- border:** A thin black line surrounding the padding area.
- padding:** A green area surrounding the content, with 12 pixels of padding on all sides.
- content:** The innermost blue area, with a width of 623 pixels and a height of 180 pixels.

The diagram also shows a total width of 623 x 180 for the content area and a total height of 20 for the margin area.

Quiz time

<code>

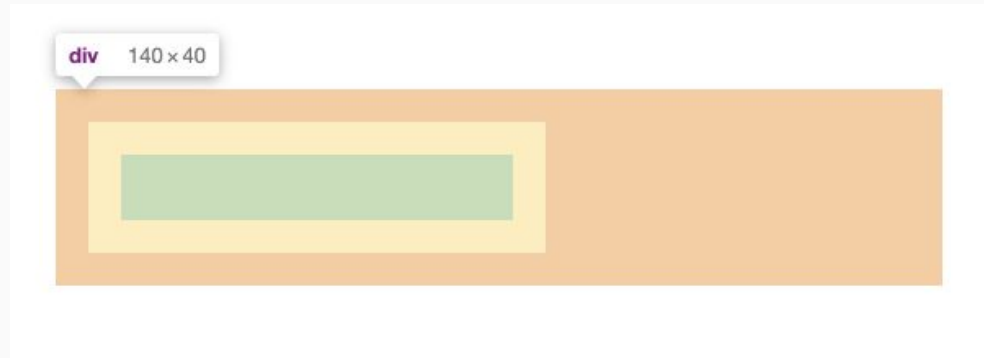
What is the width of the div?

```
/* This is the whole style sheet */
```

```
div {  
  padding: 10px;  
  border: 10px solid blue;  
  margin: 10px;  
  width: 100px;  
}
```

Quiz time

What is the width of the div?



The `box-sizing` property defines how the total width and height of an element is calculated.

box-sizing

- The box-sizing property defines how the size of elements is calculated
- Can be defined per element, but should be done globally

box-sizing

- Two different options
 - `box-sizing: content-box`
 - `box-sizing: border-box`

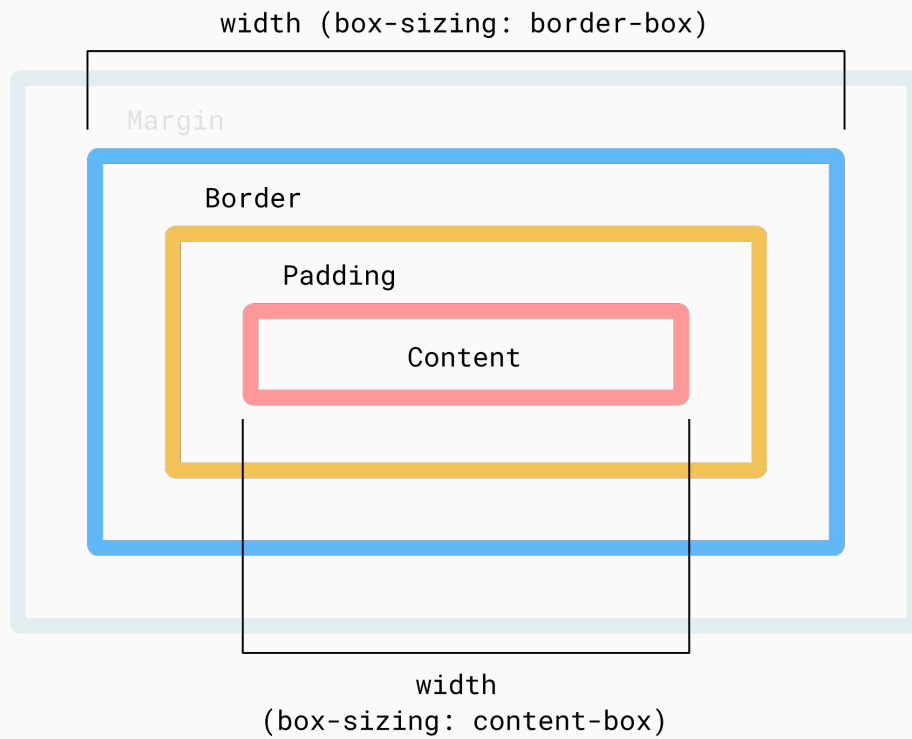
box-sizing: content-box

- `width` and `height` properties only set the size of the elements content box
- The `padding` and `border` values are added to the total dimensions
- This is the browsers default behavior 🤨

box-sizing: border-box

- `width` and `height` properties define the dimensions of the element **including** padding and border values

box-sizing



Global box-sizing

<code>

Projects usually have a global box-sizing like this.


```
*, *::before, *::after {  
  box-sizing: border-box;  
}
```

Global box-sizing

<code>

Can you guess the advantage or disadvantage of both variants?

```
html {  
  box-sizing: border-box;  
}  
  
*, *::before, *::after {  
  box-sizing: inherit;  
}
```



The diagram consists of two dashed arrows. One arrow starts from the `*, *::before, *::after` selector in the second code block and points to the `html` selector in the first code block. The other arrow starts from the `*, *::before, *::after` selector in the second code block and points to the `*, *::before, *::after` selector in the third code block. This illustrates that the `inherit` value in the second block causes all other elements to inherit the `border-box` value from the `html` element.

```
*, *::before, *::after {  
  box-sizing: border-box;  
}
```


Global box-sizing

<code>

Can you guess the advantage or disadvantage of both variants?

```
html {  
  box-sizing: border-box;  
}  
  
*, *::before, *::after {  
  box-sizing: inherit;  
}
```

Smooth integration of components and plugins that have a different box-sizing than the rest of the website

See [Article](#) on css-tricks

max-width and min-width

<code>

You can set a minimal width or a maximal width. There is max-height and min-height, too.

```
div {  
  width: 90%;  
  max-width: 900px;  
  min-width: 500px;  
}
```

```
img {  
  max-width: 100%;  
  height: auto;  
}
```

max-width often used for images



Typography

With CSS you can

- Choose the font family & the font size
- Define italic or bold text
- Use cute text-shadows
- Define the line-height
- Increase or decrease the space between letters and words
- And much more

font-family

<code>

Use lists of fonts. The browser will use the first which is installed on the machine

```
font-family: Verdana, Geneva, Tahoma, sans-serif;
```

```
/* often a good idea: use system fonts */
```

```
font-family:-apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto,  
Oxygen, Ubuntu, Cantarell, 'Open Sans', 'Helvetica Neue', sans-serif;
```

font-family

<code>

Put a generic font-family at the end

```
font-family: Verdana, Geneva, Tahoma, sans-serif;
```

- sans-serif
- serif
- monospace
- cursive
- fantasy
- system-ui

Web fonts

Web fonts must be downloaded

```
@font-face {  
  font-family: "Open Sans";  
  src: url("fonts/OpenSans-Regular-webfont.woff2") format("woff2"),  
  url("fonts/OpenSans-Regular-webfont.woff") format("woff");  
}  
  
h2 {  
  font-family: "Open Sans", sans-serif;  
}
```

Web fonts

- <https://fonts.google.com>
- Better use the fonts on your own server (DSGVO!):
<https://gwfh.mranftl.com/fonts>

font-size

<code>

Defines the size of the font

```
font-size: smaller;
```

```
font-size: 16px;
```

More typographic options

- font-weight: bold | normal
- font-style: italic | normal
- font-variant: small-caps | normal

line-height

- `line-height: 0.9;`
- `line-height: normal;`
- `line-height: 1.5;`

line-height: 0.9: Lorem ipsum dolor sit amet, consetetur s
erat, sed diam voluptua. At vero eos et accusam et justo c
ipsum dolor sit amet. Lorem ipsum dolor sit amet, consete
aliquyam erat, sed diam voluptua. At vero eos et accusam
Lorem ipsum dolor sit amet.

line-height: normal: Lorem ipsum dolor sit amet, consete
aliquyam erat, sed diam voluptua. At vero eos et accusam
Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, c
magna aliquyam erat, sed diam voluptua. At vero eos et a
est Lorem ipsum dolor sit amet.

line-height: 1.5: Lorem ipsum dolor sit amet, consetetur s
erat, sed diam voluptua. At vero eos et accusam et justo c
ipsum dolor sit amet. Lorem ipsum dolor sit amet, consete
aliquyam erat, sed diam voluptua. At vero eos et accusam
Lorem ipsum dolor sit amet.

letter-spacing and word-spacing

letter-spacing: normal - standard

letter-spacing: 2px - little bit more

letter-spacing: -1px - little bit less


word-spacing: normal - standard


word-spacing: 1rem - little bit more


word-spacing: -5px - little bit less


Typography – try it out in Firefox

Verwendete Schriftarten **Arial**
Arial Bold, Arial

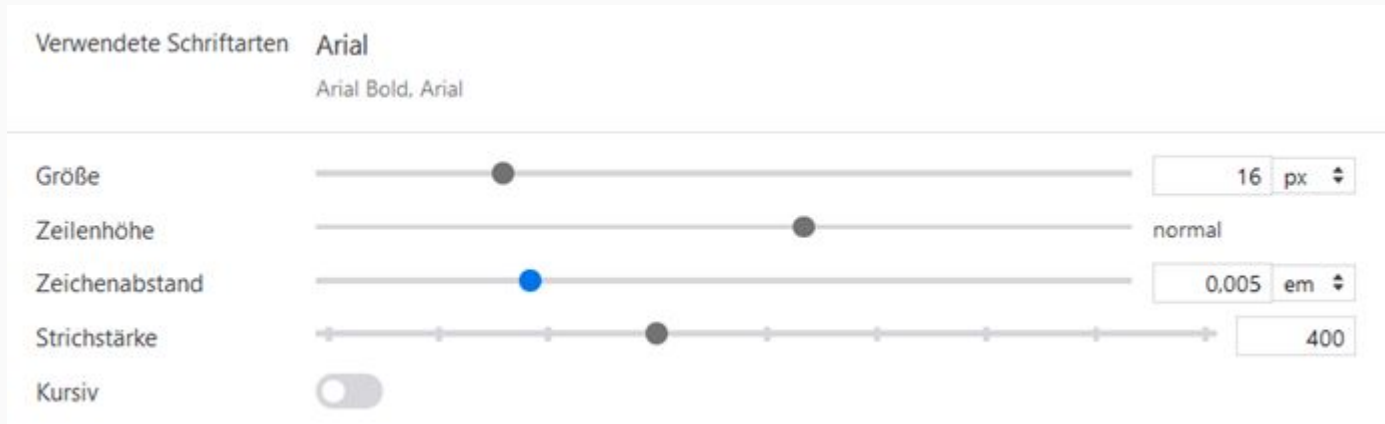
Größe  16 px ▾

Zeilenhöhe  normal

Zeichenabstand  0,005 em ▾

Strichstärke  400

Kursiv

The image shows a screenshot of the Firefox font settings interface. At the top, it displays the current font family as 'Arial' and the font styles as 'Arial Bold, Arial'. Below this, there are five settings: 'Größe' (Size) set to 16 px, 'Zeilenhöhe' (Line height) set to normal, 'Zeichenabstand' (Letter spacing) set to 0,005 em, 'Strichstärke' (Stroke thickness) set to 400, and 'Kursiv' (Italic) which is currently disabled. Each setting is accompanied by a slider or a toggle switch.

font shortcut

<code>

```
/* at least font-size and font-family */  
font: 16px sans-serif;
```

```
/* wrong! */  
font: sans-serif;
```

```
/* wrong! */  
font: 16px;
```

```
/* font-size/line-height and font-family */  
font: 16px/2 sans-serif;
```

```
/* italic, font-size/line-height and font-family */  
font: italic 16px/2 sans-serif;
```

Quiz time – what will it look like?

```
h1 {  
  font: 16px sans-serif;  
}
```

Quiz time – what will it look like?

```
h1 {  
  font: 16px sans-serif;  
}
```

/* not bold anymore -
not specified properties
are set to normal */



```
h1 {  
  font: 16px sans-serif;  
  font-style: normal;  
  font-variant-caps: normal;  
  font-weight: normal;  
  font-stretch: normal;  
  font-size: 16px;  
  line-height: normal;  
  font-family: sans-serif;
```


text-shadow

<code>

```
/* offset-x | offset-y | blur-radius | color */  
text-shadow: 1px 1px 2px black;  
/*multiple text shadows */  
text-shadow: 1px 0px 1px #CCCCCC,  
0px 1px 1px #EEEEEE, 2px 1px 1px #CCCCCC,  
1px 2px 1px #EEEEEE, 3px 2px 1px #CCCCCC,  
2px 3px 1px #EEEEEE, 4px 3px 1px #CCCCCC,  
3px 4px 1px #EEEEEE, 5px 4px 1px #CCCCCC,  
4px 5px 1px #EEEEEE, 6px 5px 1px #CCCCCC,  
5px 6px 1px #EEEEEE, 7px 6px 1px #CCCCCC;
```

text-align

<code>

for aligning texts (p, h1, etc.)

```
text-align: left | right | start | end | center | justify
```

text-align - logical values

<code>

With writing mode: Instead of left and right better use start and end

direction: ltr

```
text-align: right Lorem, ipsum dolor sit amet consectetur  
adipiscing elit. Facere, qui
```

```
text-align: end Lorem, ipsum dolor sit amet consectetur  
adipiscing elit. Facere, qui
```

direction: rtl

```
text-align: right Lorem, ipsum dolor sit amet consectetur  
adipiscing elit. Facere, qui
```

```
text-align: end Lorem, ipsum dolor sit amet consectetur  
adipiscing elit. Facere, qui
```

text-decoration

<code>

text-decoration-color

text-decoration-line: none | underline | overline | line-through |
underline overline

text-decoration-style: solid | double | dotted | dashed | wavy

text-decoration-thickness

text-decoration

<code>

Often used to remove the underline from links or to put it back on hover

```
a { text-decoration: none; }  
a:hover { text-decoration: underline; }
```

list-style-type

<code>

useful for lists (ul/ol/li-items)

```
/* Some examples */  
list-style-type: none;  
list-style-type: disc;  
list-style-type: circle;  
list-style-type: "\1F44D"; /* thumbs up sign */
```

Basic Selectors

Type selector

<code>

The type selector styles all elements with a specific element name.

```
p { color: hotpink; }
```

```
<body>
  <header>...</header>
  <h1>...</h1>
  <p>...</p>
  <div>
    <p>...</p>
  </div>
  <p>...</p>
</body>
```


Class selector

<code>

The class selector styles all elements that have a specific class. An element can have several classes

```
.pink { color: hotpink; }
```

```
<header class="pink"></header>  
<p class="example"></p>  
<div class="pink cats"></div>
```

Class selector

<code>

You can use colons in your class names, but you have to escape them in your CSS

```
.foo\:bar { color: hotpink; }
```

```
<header class="foo:bar">...  
</header>
```

ID selector

<code>

The ID selector styles the element with a specific id, while id *should* be unique.

```
#foo { color: hotpink; }
```

```
<body>
  <header>...</header>
  <h1>...</h1>
  <p>...</p>
  <div>
    <p id="foo">...</p>
  </div>
  <p>...</p>
</body>
```

Attribute selector

<code>

An example that styles all elements with the title attribute.

```
[title] { color: hotpink; }
```

```
<p>...</p>  
<p title="Cats">...</p>  
<div title="Catz">...</div>  
<p id="Cats">...</p>
```

Attribute selector

<code>

An example that styles all elements with the title attribute and a specific value. The matching is **case-sensitive**.

```
[title="Cats"] { ... }
```

```
<p title="cats">...</p>  
<p title="Cats">...</p>  
<div title="Catz">...</div>
```

Attribute selector

<code>

Add the `i` flag to the selector to match strings case-insensitive

```
[title="cats" i] { ... }
```

```
<p title="cats">...</p>  
<p title="Cats">...</p>  
<div title="Catz">...</div>
```

Attribute selector

<code>

The attribute selector can also be used with the value patterns *starts with*, *contains* and *ends with*.

```
/* Starts with */  
[href^="mailto"] { ... }
```

```
/* Contains */  
[title*="foo"] { ... }
```

```
/* Ends with */  
[href$=".pdf"] { ... }
```

Universal selector

<code>

The universal selector styles all elements in a document.

```
* { color: hotpink; }
```

```
<body>  
  <header>...</header>  
  <h1>...</h1>  
  <p>...</p>  
  <div>  
    <p>...</p>  
  </div>  
</body>
```


Universal selector *

- The universal selector should be used very rarely
 - Useful for global box sizing
- Does not match the pseudo-elements `::before` and `::after`

Selector Lists

<code>

One ruleset can have multiple comma-separated selectors.

```
header, .cat, [title] {  
  background-color: hotpink;  
}
```

Task

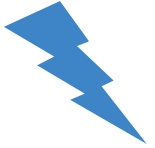
Basic Selectors



Advanced Selectors

Combinators, pseudo classes & elements


Why / What you'll learn



- Basic selectors are a good starting point but are not sufficient for real-world applications
- With the help of advanced selectors common problems can be solved mostly in CSS

Selector **combinators** allow you to **utilize**
multiple basic selectors to a **more powerful**
selector.

Descendant Combinator

- Syntax: `selector1`  `selector2`
- Matches all elements for `selector2` that are descendants (child, grandchild, etc.) of `selector1`

Descendant Combinator

<code>

Matches all descendant elements (children, grandchildren, etc.) of selector 1.

```
main div {  
  color: hotpink;  
}
```

```
<main>  
  <article>  
    <div></div>  
  </article>  
  <div></div>  
</main>  
<div></div>
```


Child Combinator

- Syntax: `selector1 > selector2`
- Matches all elements for `selector2` that are children of `selector1`
 - Only children, no grandchildren and so on

Child Combinator

<code>

Matches all elements for selector 2 that are children of selector 1.

```
main > div {  
  color: hotpink;  
}
```

```
<main>  
  <article>  
    <div></div>  
  </article>  
  <div></div>  
</main>  
<div></div>
```

Adjacent Sibling Combinator

- Syntax: `selector1 + selector2`
- Matches all elements for `selector2` that is the **next** sibling of `selector1`

Adjacent Sibling Combinator

<code>

Matches only the first sibling element of selector 1.

```
main + div {  
  color: hotpink;  
}
```

```
<main>  
  <article>  
    <div></div>  
  </article>  
</main>  
<div></div>  
<div></div>
```

General Sibling Combinator

- Syntax: `selector1 ~ selector2`
- Matches all elements for `selector2` that are following siblings of `selector1`

General Sibling Combinator

<code>

Matches all following sibling elements of selector 1.

```
article ~ div {  
  color: hotpink;  
}
```

```
<article>  
  <div></div>  
</article>  
<div></div>  
<section>  
  <div></div>  
</section>  
<div></div>
```

Quiz Time

<code>

How can you add a margin only **between** the list items?

```
<ol>
  <li>Uno</li>
  <li>Dos</li>
  <li>Tres</li>
  <li>Quattro</li>
</ol>
```

Quiz Time

<code>

Like this? 🤔

```
li {  
  margin-top: 1rem;  
}
```

```
li:first-child {  
  margin-top: 0;  
}
```

```
<ol>  
  <li>Uno</li>  
  <li>Dos</li>  
  <li>Tres</li>  
  <li>Quattro</li>  
</ol>
```


Lobotomized owl selector

- The lobotomized owl selector follows the pattern * + *
- It is used to style successive elements with of same selector
- [Article](#) on A list apart

Lobotomized Owl selector

<code>

Oh yeah 🎉

```
li + li {  
  margin-top: 1rem;  
}
```

```
<ol>  
  <li>Uno</li>  
  <li>Dos</li>  
  <li>Tres</li>  
  <li>Quattro</li>  
</ol>
```

Pseudo-classes or -elements are keywords which can be added to a selector.

Pseudo-class and -element

<code>

Pseudo-classes use a single colon, pseudo-elements a double colon.

```
selector:pseudo-class { }
```

```
selector::pseudo-element { }
```

Pseudo-class and -element

<code>

The pseudo-class or -element is added immediately to another selector - no space ⚠ (or this will be a descendant selector)

```
selector:pseudo-class { }
```

```
selector::pseudo-element { }
```

Pseudo-classes describe a **special state** of an element, i.e. if it's currently focussed.

Pseudo-classes

- `:link` → for hyperlinks a user didn't visit yet
- `:visited` → for hyperlinks a user already visited
- `:hover` → a *pointing device* is moved over an element
- `:active` → for elements currently activated by the user
- `:focus` → user clicks or taps on element/selects it with the Tab key.
- `:focus-visible` → user selects element with the Tab key.

Pseudo-classes

<code>

Example for pseudo-classes with anchor elements.

```
a:link { background-color: gold; }
```

```
a:visited { background-color: lightblue; }
```

```
a:hover { background-color: hotpink; }
```

```
a:active { background-color: dodgerblue; }
```

```
a:focus { background-color: blue; }
```

The order of the selectors matters - but why? 🤔

Pseudo-class :not()

<code>

not() represents elements that do not match a list of selectors

```
/* Elements that are not p elements */  
:not(p) { background-color: gold; }
```

```
/* Elements that are not div and not span elements */  
:not(div):not(span) { background-color: lightblue; }
```

```
/* you can use a list of selectors instead */  
:not(div, span) { background-color: lightblue; }  
/* https://developer.mozilla.org/en-US/docs/Web/CSS/:not */
```

Pseudo-class where() and is()

<code>

Select any element that can be selected by one of the selectors

```
:where(header, footer) p:hover { background-color: gold; }
```

```
:is(header, footer) p:hover { background-color: gold; }
```

```
/* is the same as */
```

```
header p:hover, footer p:hover { background-color: gold; }
```

```
/*The selectors inside is() count towards the specificity, the selectors  
inside :where() have specificity 0 */
```

Pseudo-classes for form validation

- `:valid` → for forms and inputs that validated successfully
- `:invalid` → for forms and inputs that are invalid
- `:user-invalid` → for forms and inputs that are invalid and the user has interacted with
- `:checked` → for inputs (radio, checkbox) and option elements that are checked or selected
- `:required` → for form elements with a required attribute

Pseudo-classes for form validation

- `:out-of-range` → for inputs whose current value exceeds the min or max attributes
- `:in-range` → for inputs whose current value is within the min or max attributes
- `:placeholder-shown` → for inputs whose placeholder is visible

Pseudo classes for form validation

<code>

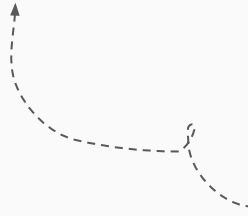
Minimal form validation example with pseudo-classes,


```
input:valid {  
  border: 1px solid green;  
}
```

```
form:invalid button {  
  opacity: .5;  
}
```

Pseudo-classes (*-child)

- `:first-child` → is the first element among its siblings
- `:last-child` → is the last element among its siblings
- `:only-child` → is the only element of its parent



Einzelkind-Selektor 

Pseudo-classes (*-of-type)

- `:first-of-type` → is the first type of element among its siblings
- `:last-of-type` → is the last of type of element among its siblings
- `:only-of-type` → is the only type of element among its siblings

Pseudo-classes (nth-*)

- `:nth-child()` → matches elements based on their position among its siblings
- `:nth-of-type()` → matches elements based on their type and position among its siblings

How :nth-* works

- :nth-* selectors accepts
 - keyword values (**odd** and **even**)
 - position numbers (1,2,3,...)
 - functional notation

:nth-* with position value

<code>

The position of an element can be passed.

```
li:nth-child(1) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-* with keyword value

<code>

The odd keyword will create a fancy 🦓 look.

```
li:nth-child(odd) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-* functional notation

- Syntax: $\langle An+B \rangle$
 - :nth-child(n)
 - :nth-child($2n+3$)
 - :nth-child($-3n+4$)

:nth-* functional notation

- With the functional syntax n will be 0, 1, 2, 3, 4, 5, ...
- n can be multiplied
- Numbers can be added to n

:nth-* functional notation

<code>

Let's try this 🙋

```
div:nth-child(n) { }
```

```
/*  
n  
-----  
0 = nth-child(0)  
1 = nth-child(1)  
2 = nth-child(2)  
3 = nth-child(3)  
4 = nth-child(4)  
*/
```

:nth-* functional notation

<code>

Let's do some math 🎓

```
div:nth-child(2n) { }
```

```
/*  
2 x n = 2n  
-----  
2 x 0 = 0 = nth-child(0)  
2 x 1 = 2 = nth-child(2)  
2 x 2 = 4 = nth-child(4)  
2 x 3 = 6 = nth-child(6)  
2 x 4 = 8 = nth-child(8)  
*/
```

:nth-* functional notation

<code>

Let's do some more math



```
div:nth-child(3n+2) { }
```

```
/*  
3 x n + 2 = 3n+2  
-----  
3 x 0 + 2 = nth-child(2)  
3 x 1 + 2 = nth-child(5)  
3 x 2 + 2 = nth-child(8)  
3 x 3 + 2 = nth-child(11)  
3 x 4 + 2 = nth-child(14)  
*/
```


:nth-child(n)

<code>

When n is passed, all elements will be styled.

```
li:nth-child(n) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-child(n+3)

<code>

When $n+3$ is passed, all elements starting from the 3rd will be styled.

```
li:nth-child(n+3) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-child(2n)

<code>

2n is the same as the even keyword.

```
li:nth-child(2n) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-child(2n+1)

<code>

2n+1 is the same as the odd keyword.

```
li:nth-child(2n+1) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-child(3n+1)

<code>

3n+1 will style every third element starting from the first element.

```
li:nth-child(3n+1) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

:nth-child(-n+3)

<code>

-n+3 will only style the first three elements.

```
li:nth-child(-n+3) {  
  background-color: gold;  
}
```

```
<ul>  
  <li>Item 1</li>  
  <li>Item 2</li>  
  <li>Item 3</li>  
  <li>Item 4</li>  
  <li>Item 5</li>  
  <li>Item 6</li>  
  <li>Item 7</li>  
  <li>Item 8</li>  
</ul>
```

Pseudo-class :has()

- `a:has(img)` → a element with an img inside
- `h1:has(+ p)` → h1 which is followed by a p
 - See browser support on [caniuse](#)

Pseudo-classes

- `:focus` → for elements that are currently focussed
- `:focus-within` → for elements where a descendant element is currently focussed
- `:focus-visible` → for elements that are currently focussed and the User Agent determines that the focus should be visible

Pseudo-classes

There are more pseudo classes, find the complete [list](#) on MDN.

Task

Combinators and Pseudo-classes



Pseudo-elements describe a **special part** of an element, i.e. the first letter within a paragraph.

Pseudo-elements syntax

- Pseudo-elements start with a **double colon**
- In many code examples the single colon syntax is used for [historical reasons](#)

::before and ::after

- With the pseudo-elements `::before` and `::after` you can place some content after the content of an element
- Used to decorate, prefix or suffix an element
- Use cases of **`::before`** or **`::after`** :

<https://css-tricks.com/7-practical-uses-for-the-before-and-after-pseudo-elements-in-css/>

::before and ::after

- `::before` and `::after` are activated with the `content` property
- Without `content` property they are not visible
- `::before` and `::after` are only part of the render tree
 - No dedicated event handlers can be attached

::before and ::after

<code>

Before and after can be used like this, see [example](#).

```
.cat::before {  
  content: '🐱';  
}
```

```
.cat::after {  
  content: '😺';  
}
```

```
I <br>  
like <br>  
<span class="cat">cats</span>
```



The diagram illustrates the application of CSS pseudo-classes. On the left, two CSS rules are shown: `.cat::before { content: '🐱'; }` and `.cat::after { content: '😺'; }`. Dashed arrows point from these rules to the corresponding parts of the HTML output on the right. The HTML output is `I
 like
 cats`. The word `cats` is highlighted in yellow, and the opening and closing tags of the `` are also highlighted in yellow. A dashed arrow points from the `cats` text back to the `.cat::after` rule, and another dashed arrow points from the `` tag back to the `.cat::before` rule, showing how the pseudo-classes insert the cat emojis before and after the text.

::first-letter

<code>

Can be used to style the first letter, i.e. of a paragraph
Only a subset of CSS properties are [allowed](#).

```
p::first-letter {  
  font-size: 300%;  
}
```


::first-line

<code>

Can be used to style the first line (**not the sentence**) of an element. Only a subset of CSS properties are allowed.

```
p::first-line {  
  font-size: 150%;  
  font-weight: 700;  
}
```

::selection

<code>

Can be used to style the current selection. Only a subset of CSS properties are [allowed](#).

```
p::selection {  
  background-color: black;  
  color: white;  
  font-size: 200%;  
}
```



The font-size property will be ignored.

More pseudo-elements

- More pseudo-elements
 - `::placeholder`
 - `::marker`
 - `::file-selector-button`
- See [pseudo-elements](#) on MDN

Task

Pseudo-Elements



Task

CSS Shopping



The Cascade

The cascade (illustration)



The cascade

On a simplified level the cascade defines which styles should be applied to an element in the following order:

1. Origin of CSS declarations
2. Importance
3. Specificity
4. Source order

Importance

- The `!important` keyword overwrites existing style declarations
- Avoid `!important` with the following exceptions
 - For user style sheets
 - Integration of 3rd party libraries

Importance

- The only way to overwrite an important declaration is to overwrite it again with another important declaration
 - Only works with the same specificity

Importance

<code>

The `!important` flag will overrule all style declarations.

```
p {  
  color: red !important;  
}
```

```
/* 1000 lines of css */
```

```
.fancy-p {  
  color: black;  
}
```

```
<p class="fancy-p">  
  Look at the color of my text  
</p>
```

Importance

<code>

Example for multiple `!important` declarations.

```
p {  
  color: black !important;  
}
```

```
.fancy-p {  
  color: red !important;  
}
```

```
<p class="fancy-p">  
  Look at the color of my text  
</p>
```

Importance

<code>

Multiple `!important` declarations with the same specificity are evaluated by declaration order (**Attention**: It is the class selector `.p` here and not `p`)

```
.fancy-p {  
  color: red !important;  
}
```

```
.p {  
  color: black !important;  
}
```

```
<p class="p fancy-p">  
  Look at the color of my text  
</p>
```

Specificity



Thomas Fuchs

@thomasfuchs



Two CSS properties walk into a bar.

A barstool in a completely different bar falls over.

♥ 3,205 5:10 PM - Jul 28, 2014



💬 3,198 people are talking about this



Specificity

- Specificity measures how specific a selector is
- More specific selectors overrule generic selectors
- If two selectors have the same specificity, the last defined style declaration is used (a.k.a. order in source code)

Specificity order

- Selector specificity in ascending order
 - Universal selector
 - Type selector, pseudo-elements
 - Class selector, attribute selector, pseudo-classes
 - ID selector
 - Inline styles

Specificity score

- Each selector has a specificity score
 - Type selector and pseudo elements → 1
 - Class, pseudo-class, attribute → 10
 - ID → 100
- [Specificity calculator](#)

Specificity

<code>

What is the specificity for each ruleset?

```
#main section.my-section p {  
  color: cyan;  
}
```

```
body div:first-of-type p {  
  color: maroon;  
}
```

- *Type selector and pseudo-elements*
→ 1
- *Class, pseudo-class and attribute*
→ 10
- *ID* → 100

Specificity

<code>

What is the specificity for each ruleset?

```
#main section.my-section p {  
  color: cyan;  
}
```

```
100 #main  
..1 section  
.10 .my-section  
..1 p  
-----  
112
```

```
body div:first-of-type p {  
  color: maroon;  
}
```

```
..1 body  
..1 div  
.10 :first-of-type  
..1 p  
-----  
013
```

Specificity best practices

- Use type selectors only for resets / normalize
- Use class selectors for application styling
- **Try** to avoid selector combinators and selector nesting
 - ◆ Use code linter (sass-lint, style-lint)
- Scope your styles (i.e. shadow dom) or/and use a methodology (BEM, SMACSS, OOCSS)

Order of CSS rulesets

- If a selector is defined multiple times, the last specified selector will take priority
- Common issue in projects with large CSS codebase 😭

Order of CSS rulesets

<code>

CSS rulesets with the same selector overwrite previous defined declarations.

```
.box {  
  background-color: fuchsia;  
}
```

```
/* 1000 lines of crap css */
```

```
.box {  
  background-color: mediumseagreen;  
}
```

Guess we will have a nice
mediumseagreen background 🙌

Dev Tools are your best friend

<code>

They'll tell you when a rule is not applied.

```
.panel {  
  border: 0px;  
  padding: 0px;  
  box-shadow: none;  
}  
  
.panel-default {  
   border-color: #ddd;  
}
```

Origin of CSS declarations

CSS declarations can have different origins and will be applied in descending order:

1. User-agent stylesheets → Styles provided by the browser
2. Author stylesheets → Styles provided by website itself
3. User stylesheets → Styles provided by the user

Origin of CSS declarations

Order (low to high)	Origin	Importance
1	user-agent (browser)	normal
2	user	normal
3	author (developer)	normal
4	CSS @keyframe animations	
5	author (developer)	!important
6	user	!important
7	user-agent (browser)	!important
8	CSS transitions	

https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#cascading_order

@layer

<code>

New possibility to deal with cascade

```
@layer components, utilities;
```

This order matters

```
@layer utilities { }
```

```
@layer components { }
```

No matter how specific the rules in components are, the rules in utilities are applied

Inheritance

In CSS, inheritance controls what happens *when no value is specified for a property on an element.*

- [Mozilla Developer Network](#)

Inheritance

- Elements can inherit specific property values from parent elements
- Inherited properties are mostly text related (i.e. font, color, direction)
- Layout related properties are not inherited (i.e. width, height, padding)

Inheritance

<code>

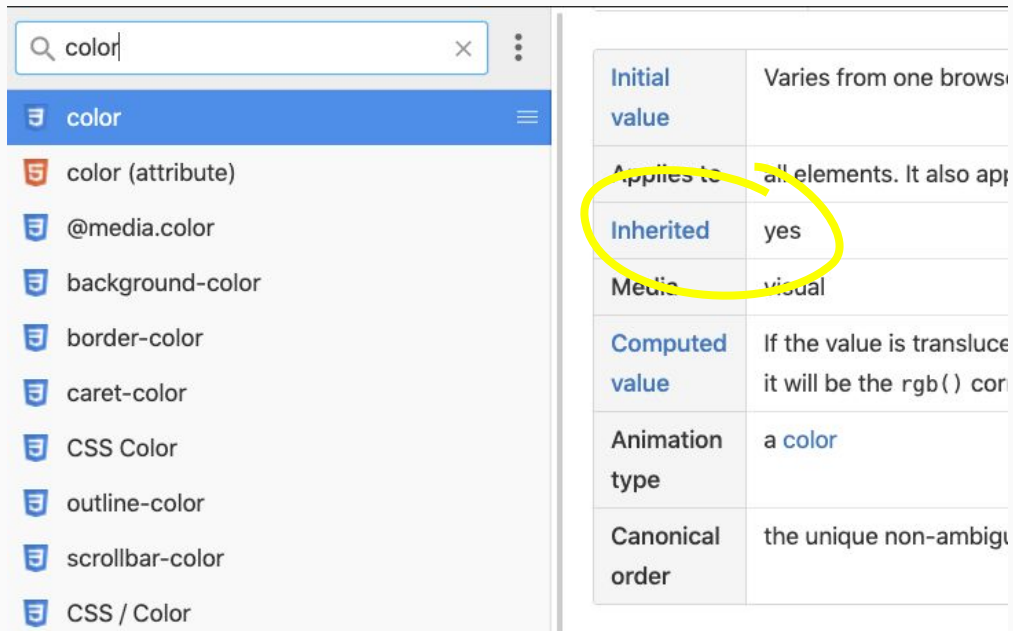
Certain properties are inherited for the paragraph.

```
body {  
  font-size: 12px;  
  color: dodgerblue;  
  height: 500px;  
}  
  
<body>  
  <p>  
    I'm dodgerblue and 12px big  
  </p>  
</body>
```

Height property is not inherited.

Inheritance

Check if a property is inherited on devdocs.io



The image shows a search interface on devdocs.io. The search bar contains the text 'color'. Below the search bar, a list of search results is displayed, with 'color' selected. To the right of the search results, a table provides detailed information about the 'color' property. The 'Inherited' property is highlighted with a yellow circle, and its value 'yes' is also highlighted. The table includes the following rows:

Initial value	Varies from one brows
Applies to	all elements. It also app
Inherited	yes
Media	visual
Computed value	If the value is transluce it will be the rgb () con
Animation type	a <code>color</code>
Canonical order	the unique non-ambigu

Inheritance

Check if a property is inherited inside the DevTools

The image shows a browser window displaying a blog post snippet. The snippet title is "asm!" and the text discusses a proposed extension to core WebAssembly. Below the snippet, it says "Geschrieben am November 21, 2019 von Nick Fitzgerald".

Below the browser window, the Chrome DevTools interface is visible. The "Elements" panel on the left shows a tree view of the page's DOM. The selected element is a paragraph with class "entry-summary". The "Styles" panel on the right shows the styles for this element. A yellow circle highlights the "Inherited from" section, which shows that the "background-color" and "color" properties are inherited from the "html" element.

```
Elements Console Sources Network Performance Memory Application Security Audits Augury axe Layers
▼<div class="home-hacks">
  ▼<div class="column-container center">
    ▼<div class="column-hacks" dir="ltr">
      <h2></h2>
      ▼<ul class="hfeed">
        ▼<li class="hentry">
          ▼<h2 class="entry-title"></h2>
          <p class="entry-summary"> == $0
          <p class="entry-meta vcard"></p>
          </li>
          <li class="hentry"></li>
          <li class="hentry"></li>
          <li class="hentry"></li>

```

```
Styles Computed Event Listeners DOM Breakpoints Proc
Filter
Inherited from html.gr_dev...
html {
  background-color: #fff;
  color: #333;
}
html {
  color: #333;
}
```


Inheritance keywords

- The inheritance of a property can be controlled with these keywords
 - `inherit`
 - `initial`
 - `unset`

inherit keyword

<code>

The property value is inherited from the parent element.

```
article {  
  color: dodgerblue;  
}
```

```
a {  
  color: inherit;  
}
```

```
<article>  
  <a href="#">dodgerblue</a>  
</article>
```



initial keyword


<code>

With `initial` the element will use the initial (or default) property value.

```
article {  
  color: dodgerblue;  
}
```

```
a {  
  color: initial;  
}
```

```
<article>  
  <a href="#">black</a>  
</article>
```



unset keyword

- The unset keyword behaves differently depending if the property is a inherited property
 - Property is inherited → uses the inherited value
 - Property is not inherited → uses the default value

unset keyword

<code>

If the property is inherited it will use the ancestors value.

```
article {  
  color: dodgerblue;  
}
```

```
a {  
  color: unset;  
  outline: unset;  
}
```

color is inherited

```
<article>  
  <a>I will be dodgerblue</a>  
</article>
```

outline will be the default outline

The diagram illustrates the inheritance of the 'color' property. A dashed arrow points from the text 'color is inherited' to the 'I will be dodgerblue' text within the HTML code. Another dashed arrow points from the text 'outline will be the default outline' to the '<a>' tag in the HTML code.

unset keyword

<code>

Reset all properties of an element. This can be useful for custom components.

```
.custom-checkbox {  
  --custom-checkbox-size: 10vmin;  
  all: unset;  
  position: relative;  
  width: calc(2 * var(--custom-checkbox-size));  
  height: var(--custom-checkbox-size);  
  border: 2px solid black;  
  border-radius: 8px;  
  overflow: hidden;  
}
```

Task

What will it look like? 🤔



What will it look like? #1

<code>

What is the color of the text?

```
* {  
  color: black;  
}
```

```
body {  
  color: red;  
}
```

```
<body>  
  <p>Foo!</p>  
</body>
```


How will it look like? #2

<code>

What is the color of the text?

```
p {  
  color: red !important;  
}  
  
* {  
  color: black !important;  
}
```

```
<body>  
  <p>Foo!</p>  
</body>
```

How will it look like? #3

<code>

What is the color of the text?

```
body main p {  
  color: red;  
}  
  
.p {  
  color: black;  
}
```

```
<body>  
  <main>  
    <p class="p">Foo!</p>  
  </main>  
</body>
```

How will it look like? #4

<code>

What is the color of the text?

```
.p { color: red; }
```

```
.p.p { color: black; }
```

```
<p class="p">Foo!</p>
```

How will it look like? #5

<code>

What is the color of the text?

```
p {  
  color: red !important;  
}
```

```
<body>  
  <p style="color: black;">Foo!</p>  
</body>
```

Reset, Normalize and Fallbacks

Default style sheets

Browsers have different default style sheets

→ Websites without author styles sheets will look (minimal) different

Reset and Normalize

Two different ways to handle this

- Reset
- Normalize
- Compare: <https://codepen.io/chriscoyier/pen/JpLzjd?editors=1100>

reset.css

- Most properties will be set to 0 or a reasonable default value
 - Margins and paddings are set to 0
 - Quotes signs are removed for q and blockquote
 - List styles are removed for ul and ol


reset.css

- Reset.css [source code](#) on GitHub
- [Article](#) by Eric A. Meyer

normalize.css

- preserve useful browser defaults and normalize inconsistencies between different browsers
- [Normalize.css](#) package
- Normalize.css [source code](#) on GitHub
- Newer: <https://github.com/sindresorhus/modern-normalize>

Fallbacks

- Fallbacks can be used for
 - unsupported features in old browsers
 - slow network conditions 
 - unavailable resources

Fallbacks

- Browsers will ignore style declarations they don't know
- A fallback usually includes multiple style declarations and overrides

CSS fallback

<code>

Fallback for `rgba()` function.

```
p {  
  /* fallback */  
  color: rgba(255,0,0,0.5);  
  
  /* will be used in modern browsers */  
  color: hwb(0 0% 0% / 0.5);  
}
```

CSS fallback slow network

<code>

What's the worst case here?

```
body {  
  background-color: white;  
}  
  
header {  
  background-image: url(header.jpg);  
  color: white;  
}
```



CSS fallback slow network

<code>

Problem solved 

```
body {  
  background-color: white;  
}  
  
header {  
  background-color: black;  
  background-image: url(header.jpg);  
  color: white;  
}
```

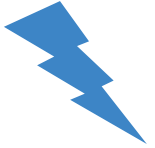
Task

Cascade, Inheritance & Normalize



BEM

Why / What you'll learn




- CSS specificity can be a pain in large projects without style scoping (hey, why is this button green?)
- BEM provides an easy to remember methodology without complicated rules which will make your CSS life easier

BEM is a highly useful, powerful, and simple naming convention that makes your front-end code easier to **read and understand**, easier to work with, easier to **scale**, more **robust** and **explicit**, and a lot more **strict**.

- [GetBEM.com](https://getbem.com)

BEM is all about avoiding any specificity issues and having one way of naming things.

BEM is not BEM

- There are different versions or approaches of BEM
- The slides are based on getbem.com (with small adjustments)
- It's ok to make adjustments because a methodology has to work for you and your team 

The problem

- During development the CSS is usually structured into smaller chunks
 - Manageable scope of CSS
 - Duplicate selectors or specificity issues are not visible
- In the browser → one big pile of CSS

Real life CSS

- Declarations with the same selector overwrite each other
- Different specificities will overwrite styles
- `!important` is sometimes the easiest way to fix things
- New CSS gets added to the end of a large file
 - is suitable for the current developer 🙄
 - might break things for someone else 🤖

Real life CSS

- The color depends on
 - parent elements
 - Classes on parent elements
 - Classes on the p element
- Developers → 🤖

```
p { color: black; }  
  
/* 100 lines of other stuff */  
  
.important-paragraph { color: #333; }  
  
/* 200 lines of crap */  
  
.news p { color: #222; }  
  
/* 50 lines of something else */  
  
#content .news p { color: #111; }  
  
/* 200 lines more crap */  
  
.text { color: #000 !important; }
```


BEM

- BEM stands for **B**lock **E**lement **M**odifier

BEM block

- A block is a standalone entity
- A block is meaningful on its own
- Can be everything
 - blog post
 - site footer
 - button

BEM element

- An element is a part of a block
- An element is semantically tied to its block
- Elements have no standalone meaning



BEM element

- BEM elements can be
 - Inputs of a form
 - Buttons of a button panel
 - Link within a menu

BEM modifier

- A modifier defines a special state or behavior
- Can be applied to any BEM block or element
- A modifier can be
 - The disabled state of a button
 - A different size for a picture

BEM rules

- BEM only uses classes to style elements
 - All selectors have the same specificity 
- Avoid combinators, because the specificity changes
 - No rule without exception 

BEM rules

- Class names use dash casing
 - a.k.a. kebab-case

```
.sub-menu { ... }
```

```
.section-header { ... }
```

```
.personen-vereinzelungs-anlage { ... }
```

```
.button { ... }
```

BEM naming

- The block is a regular class name
 - i.e. `.header` or `.button`
- Block and Element are separated with double underscore
 - i.e. `.header__title` or `.button__icon`
- Modifiers are prefixed with a double dash
 - i.e. `.header__title--highlighted` or `.button--warning`

BEM naming

<code>

BEM naming example for button and its modifiers.

```
.button {  
  color: #000;  
}  
  
.button--warning {  
  background-color: red;  
}  
  
.button--success {  
  background-color: green;  
}
```

BEM naming

<code>

Button example with BEM.

```
<button class="button">Normal button</button>
```

```
<button class="button button--warning">Please don't</button>
```

```
<button class="button button--success">You did it</button>
```

BEM naming

<code>

BEM will not save you from mistakes.

```
<button class="button button--success button--warning">Ooops!</button>
```

Order of BEM rulesets

- Block rulesets
- Block Modifier rulesets
- Block Element rulesets
- Block Element Modifier rulesets

Order of BEM rulesets

<code>

Always keep the order of rulesets like this to avoid any surprises 🎁

```
.block {}
```

```
.block--modifiers {}
```

```
.block__elements {}
```

```
.block__element--modifiers {}
```

But why?

Modifiers can safely overwrite declarations of blocks or elements for a special use case → calculated risk

Contextual Formatting

- BEM blocks should not set any contextual formattings
 - i.e. `margin` or `position + offsets`
- Blocks should be formatted and arranged by it's usage context

Contextual Formatting

<code>

A BEM block should not contain any contextual formatting like in this example.

```
/* ⚠ Wrong */  
.button + .button {  
  margin-left: 1rem;  
}
```

```
/* ✅ Right */  
.button-panel > .button + .button {  
  margin-left: 1rem;  
}
```

```
<div class="button-panel">  
  <button class="button">...</button>  
  <button class="button">...</button>  
</div>
```

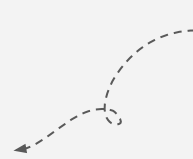

BEM mistakes

<code>

BEM should not reflect the HTML structure.

```
<section class="card">
  <header class="card_header">
    <h2 class="card_header_title"></h2>
  </header>
</section>
```

Nope 



BEM mistakes

<code>

BEM should not reflect the HTML structure.

```
<section class="card">
  <header class="card__header">
    <h2 class="card__title"></h2>
  </header>
</section>
```

Solved 



<h2 class="card__title"></h2>

Downsides of BEM

- Markup can contain a lot of class names
- Repetitive naming
- Ugly class names

Task

BEM



Display Property and Flow Layout

The display property defines how an element participates in the layout process.

Why / What you'll learn



- Knowing the different display types is the first step of understanding the browsers layout process
- Stop guessing what's going to happen when you save your CSS

Display

- The `display` property defines the display type of an element
- Most used [display](#) types
 - `block`
 - `inline`
 - `inline-block`
 - `flex`
 - `grid`
 - `table`
 - `none`

Display

- The display property defines two things
 - Outer display type
 - How does the element participate in the flow layout?
 - Inner display type
 - How will children of the element laid out?

Quiz time

<code>

What are the widths of the elements?

```
div {  
  width: 200px;  
}
```

```
button {  
  width: 200px;  
}
```

```
span {  
  width: 200px;  
}
```

Quiz time

<code>

What are the widths of the elements?

```
div {  
  width: 200px; ✓  
}
```

```
button {  
  width: 200px; ✓  
}
```

```
span {  
  width: 200px; ✗  
}
```

Quiz time

<code>

What are the widths of the elements?

```
div {  
  width: 200px; ✓  
}
```

```
button {  
  width: 200px; ✓  
}
```

```
span {  
  display: block;  
  width: 200px; ✓  
}
```

Display: block

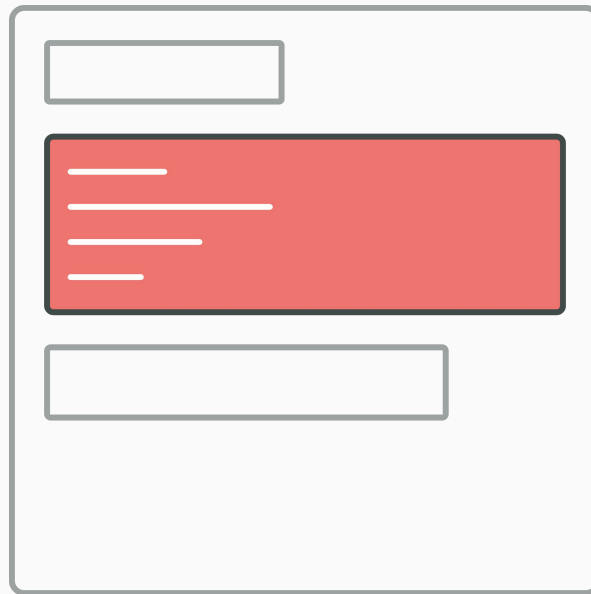
- Elements with `display: block` are called *block elements*
- Creates a new line before and after itself
- Takes up full width of the parent element by default

Display: block

- Adjusts to content height by default
- `width` and `height` properties can be set
- Can contain *any** elements

Display: block

- 100% width of parent element
- Adjusts to content height
- New lines before and after
- Width and height can be set



Block-level elements

<code>

Example of some block-level elements. Complete list can be found [here](#).

```
<header></header>  
<main></main>  
<footer></footer>  
<div></div>
```

```
<p></p>
```

Paragraph tags are block-level elements, but only specific [elements](#) are allowed as content.

Display: inline

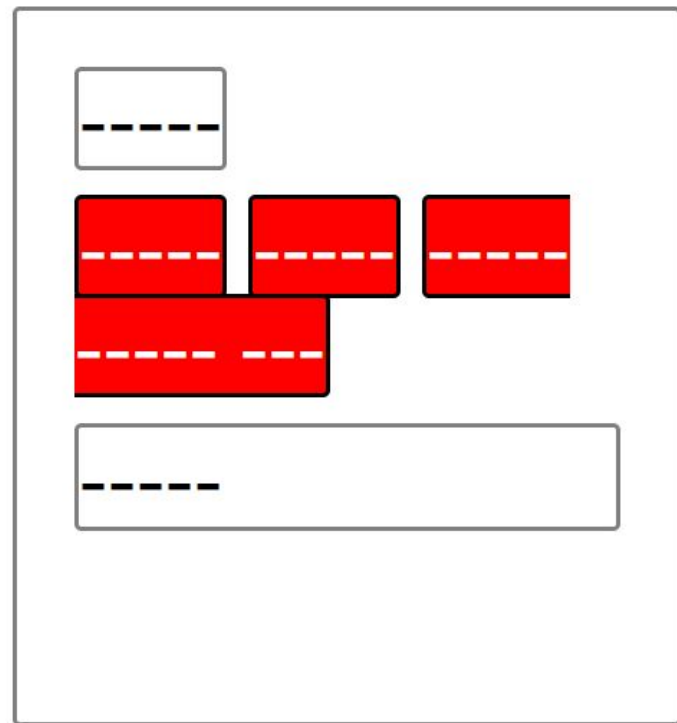
- Elements with `display: inline` are called *inline elements*
- `width` and `height` properties are ignored
- As big as the elements content

Display: inline

- Overflows to new line if there is not enough space
- Creates no new lines
- Can only contain other inline elements or data
- Vertical margin is ignored and vertical padding behaves odd
 - See [inline formatting context](#)

Display: inline

- No new lines
- Overflows to new line
- width and height are ignored
 - As big as content



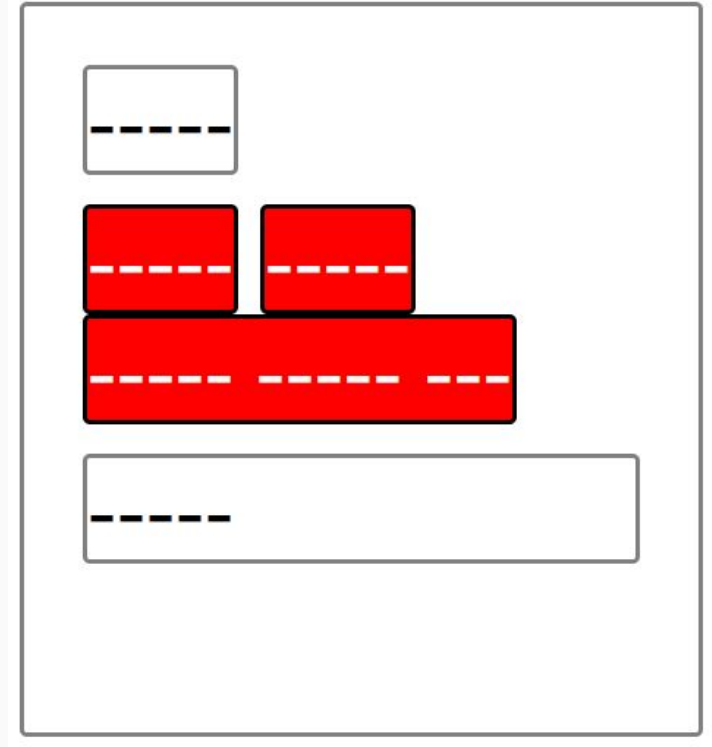
What about `display: inline-block`? 🙄

Display: inline-block

- Behaves like `display: inline` but also accepts `width` and `height` properties

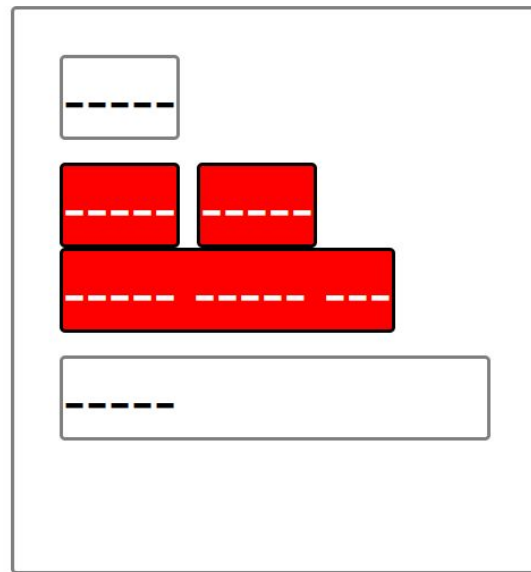
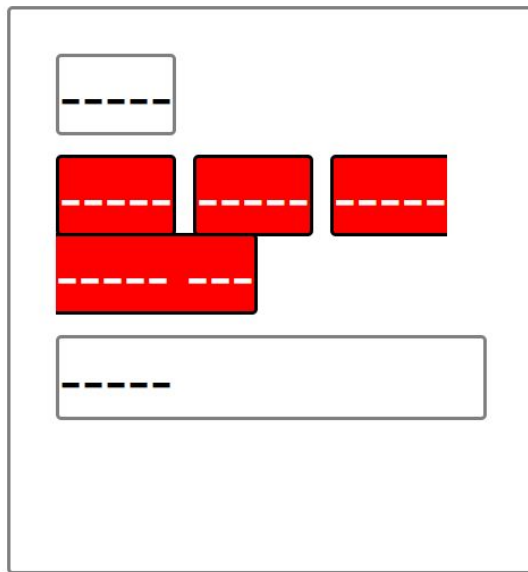
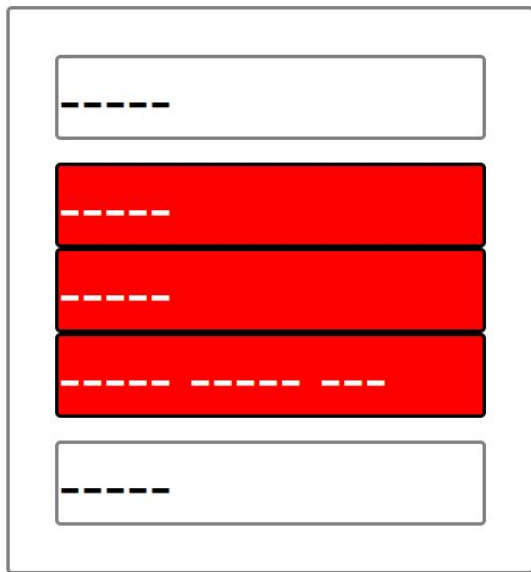
Display: inline-block

- No new lines
- flows to new line
- width and height can be set



Display types

Visual reminder for block-level, inline and inline-block elements.



Display: none

- If an element has `display: none` it will not be rendered and no space will be reserved

Visibility: hidden

- When using `visibility: hidden` the element is indeed hidden
- But it still takes up space in the normal document flow

Position

The position property allows **taking elements out of the normal document flow** to create complex components.

Why / What you'll learn



- You will learn
 - the difference between non-positioned and positioned elements
 - how positioning can affect the normal document flow
- Position is still important today, but in some situations you can use grid instead or even anchor positioning (developer.chrome.com)

Position

- The `position` property defines how an element is positioned
- The `top`, `bottom`, `left` and `right` properties define the final location of *positioned elements*

Position: static

- `position: static` is the default position of all elements
- Offset properties (`top`, `right`, `bottom`, `left`) are ignored
- The element is not positioned

Position: static



Position: relative

- The element still takes up the original space in the normal document flow
- `top` or `bottom` set vertical offset
- `left` or `right` set the horizontal offset
- The element is positioned

Position: relative



position: absolute can be *slightly* tricky
because it depends on the **containing block**
and if offset properties (top, bottom, left, right)
are set.

Containing block

- The containing block for absolute positioned elements is *the nearest positioned parent element*

Position: absolute

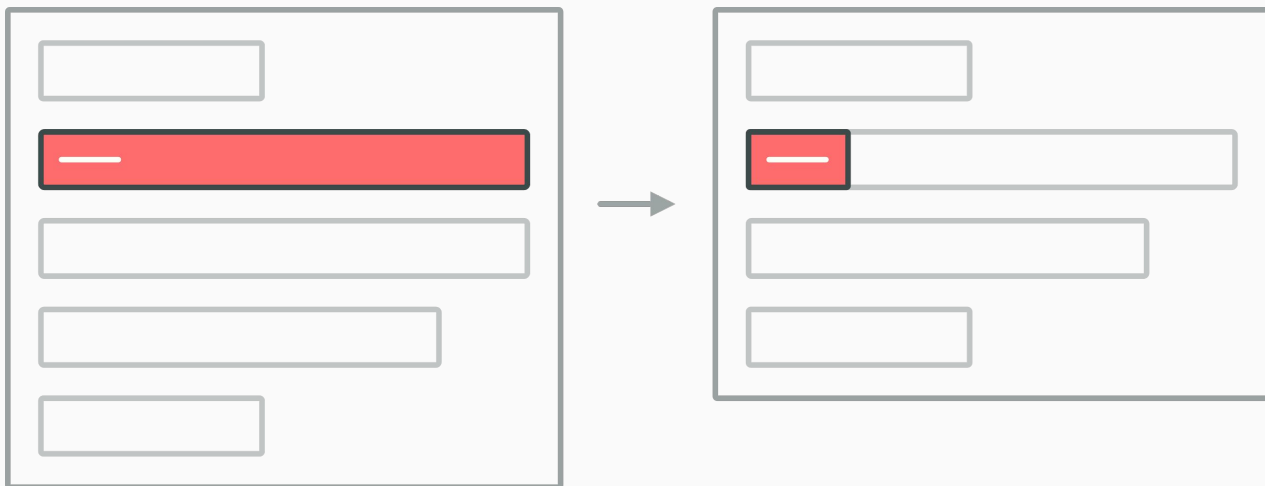
- Element is **removed** from the normal document flow
- Block-level elements shrink to its content size if **no** width or height property is set
- The element is positioned

Position: absolute

- If no offset properties are given, the element stays in its normal document flow position
- If offset properties are given, it's positioned according to the offsets within the padding box of the containing block

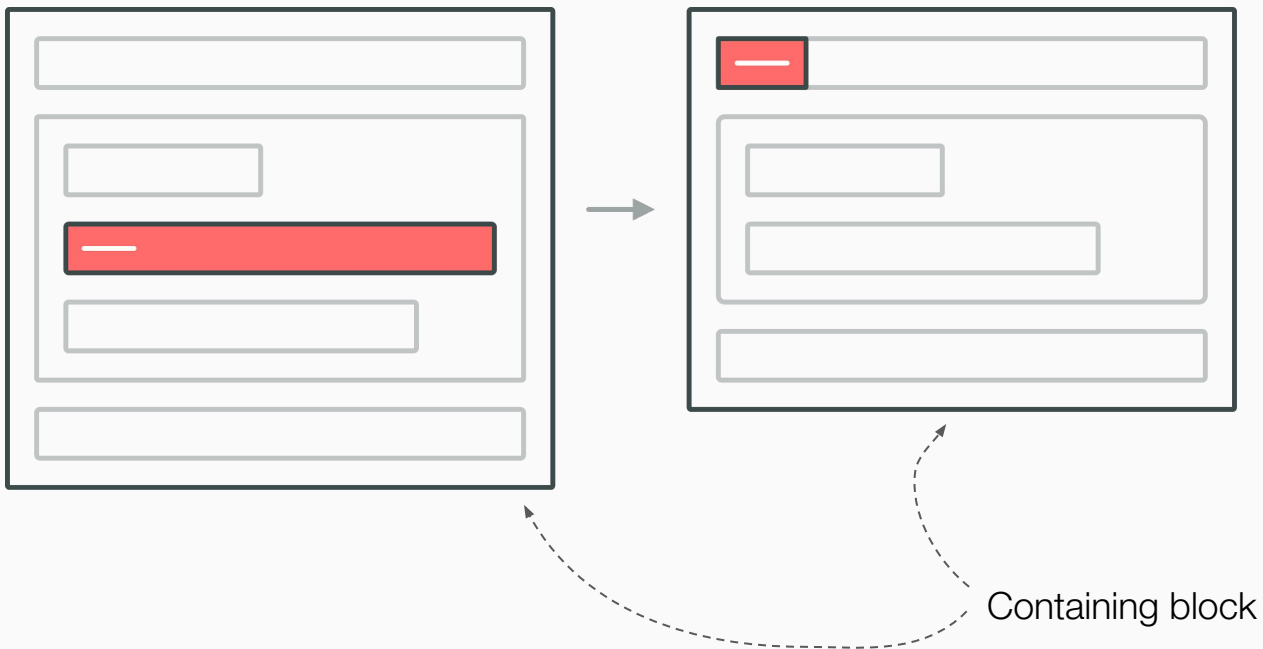
Position: absolute

Absolute positioned element without any given offset properties.



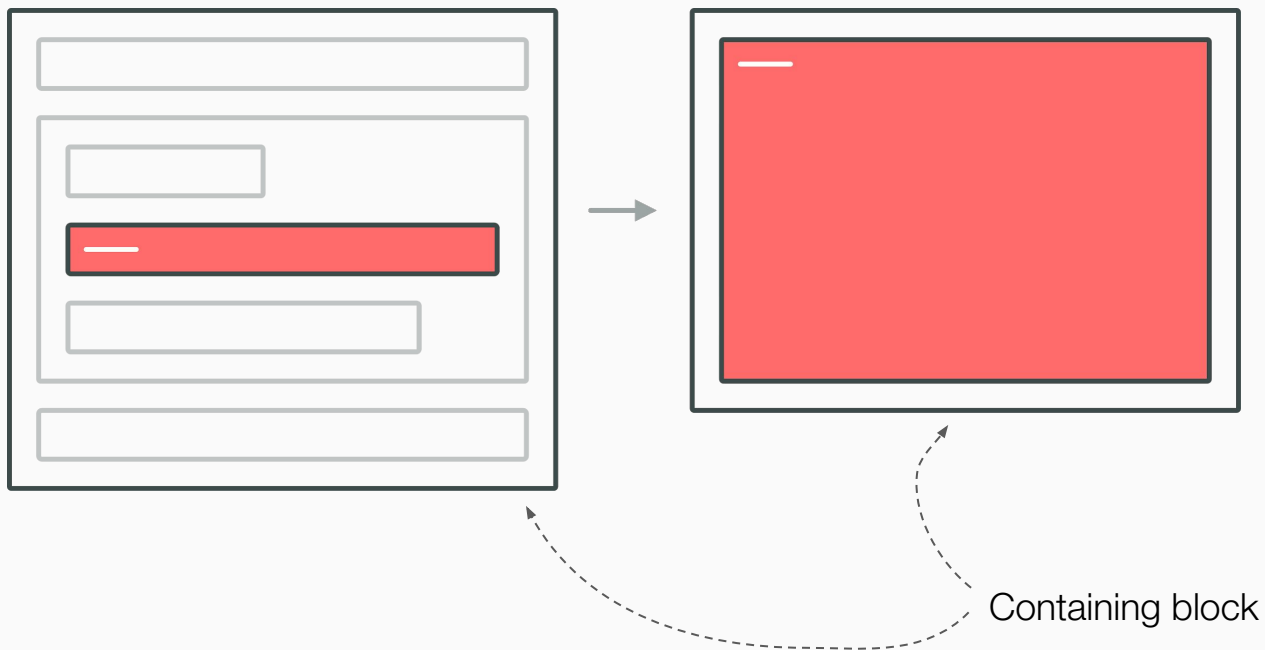
Position: absolute

Given offsets are top: 0px and left: 0px



Position: absolute

Given offsets are top: 0px, bottom: 0px, left: 0px and right: 0px (or you can use inset:0px instead)



Position: fixed

- Behaves like `position: absolute`
- Only difference: the containing block is the viewport
 - Viewport = visible portion of the document in the browser window
- PS: the element is positioned

Position: sticky

- Native sticky headers (elements that stops scrolling at a certain position)
- Is positioned to the elements nearest scrolling ancestor
- Behaves like normal flow content until the defined offsets (usually `top` property) are reached
- After the defined offsets are reached, behaves like fixed positioned
- Example: <https://codepen.io/FlorenceM/pen/RwQKwww>

Position

Position	Positioned?	Positioned to?
static	No	-
relative	Yes	Normal flow + offsets
absolute	Yes	Containing block + offsets
fixed	Yes	Viewport + offsets
sticky	Yes	Scrolling ancestor + offsets

Task

Position



Flexbox

Flexbox is a efficient and modern way of
layouting

Why / What you'll learn



- Flexbox allows creating layouts (for components) with ease

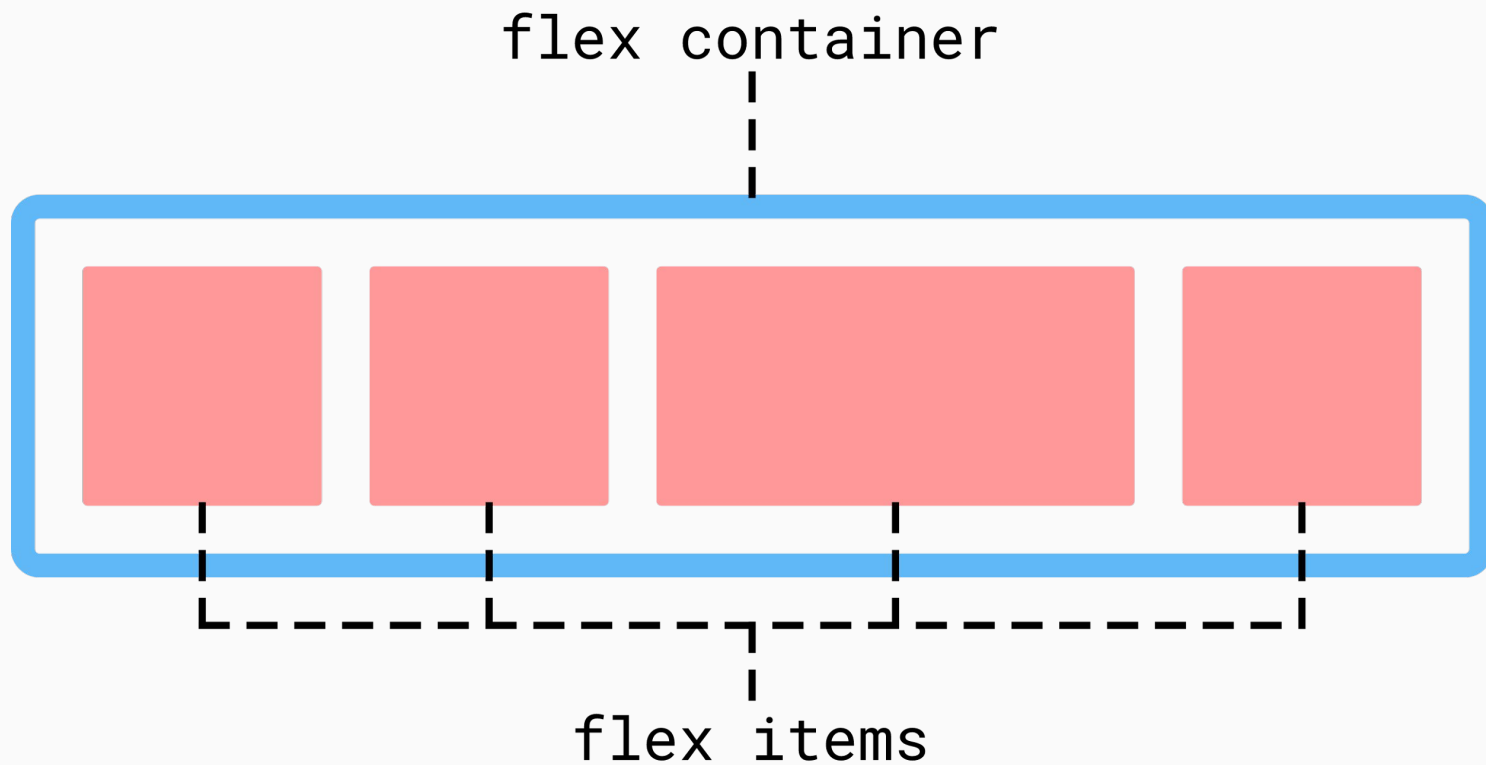
Flexbox

- Flexbox allows you to distribute items among available space
- Primarily designed for one-dimensional layouts
- Powerful alignment capabilities

Flex container and flex items

- An element with `display:flex` becomes the *flex container*
- The children of the flex container are called *flex items*

Flex container and flex items



flex and inline-flex

- There are two different flexboxes available
 - `display: flex` will create a block-level flexbox
 - `display: inline-flex` will create an inline-level flexbox

flex and inline-flex

<code>

Respecting the initial display type with flex and inline-flex.

```
.header {  
  display: flex;  
}
```

```
<header class="header"></header>
```

```
.span {  
  display: inline-flex;  
}
```


```
<span class="span"></span>
```

flex-direction

- A flexbox has a main- and cross-axis
- The direction of the main-axis is defined with the `flex-direction` property
- Flex items are laid out according the `flex-direction`

flex-direction

- The different values for `flex-direction` are
 - `row` (default value) → items are placed from **left to right**
 - `row-reverse` → items are placed from **right to left**
 - `column` → items are stacked from top to bottom
 - `column-reverse` → items are stacked from bottom to top

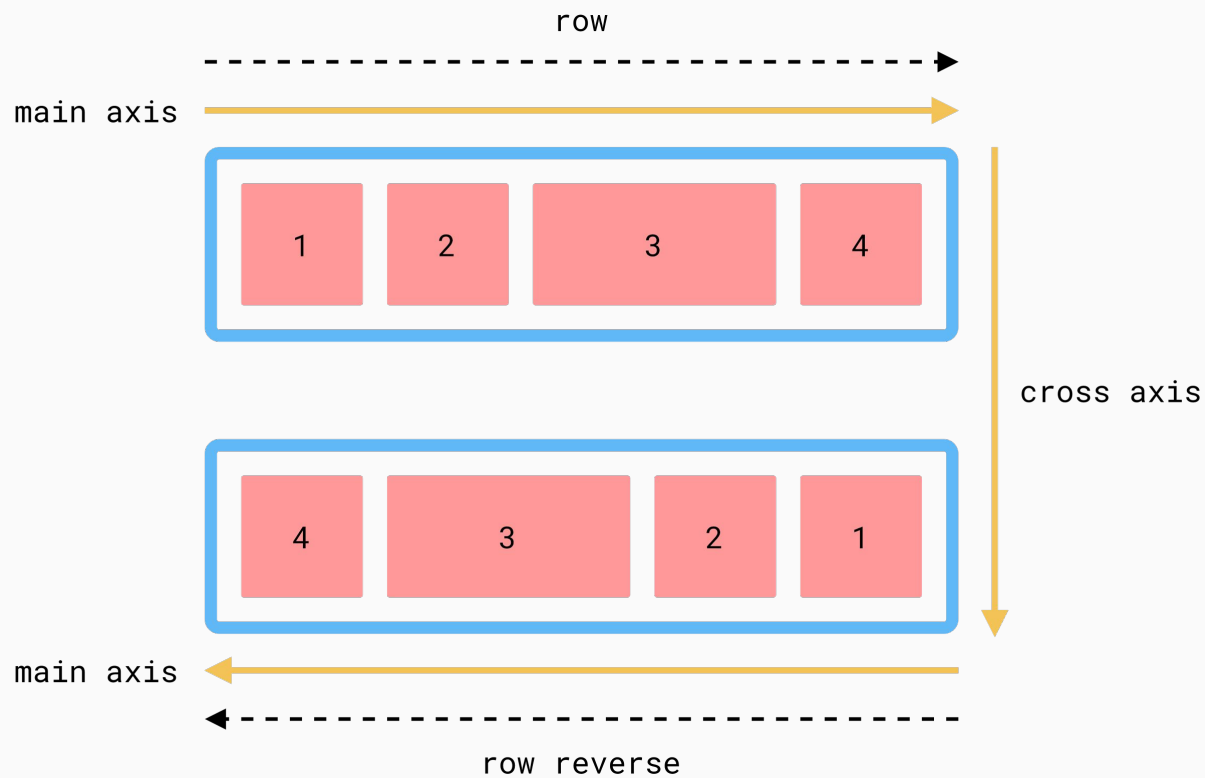


⚠ Assuming a left to right writing mode

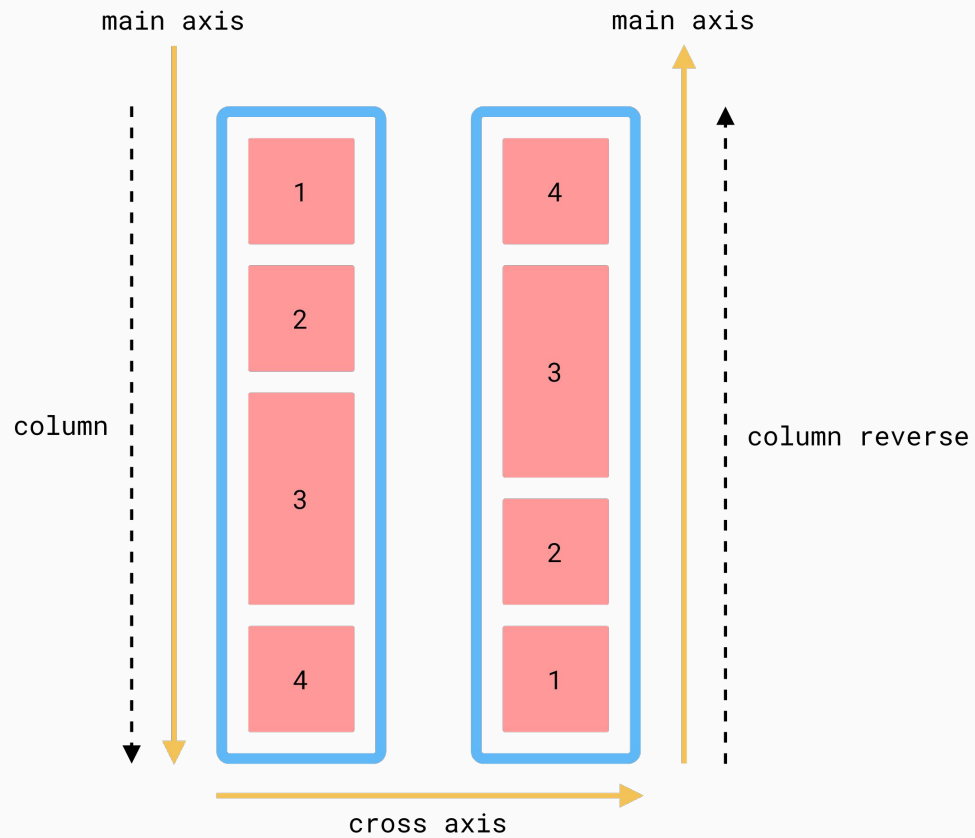
Flex-direction and writing mode

- The `flex-direction` variants `row` and `row-reverse` are always based on the writing mode
- Altering the writing mode will also reverse the display order of the flexbox when using `row` or `row-reverse`

flex-direction: row and row-reverse



flex-direction: column and column-reverse



justify-content (for flex containers)

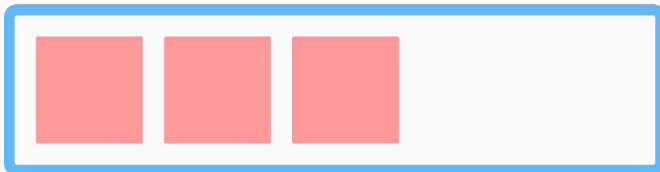
- The `justify-content` property defines the distribution of space and flex items across the main-axis

justify-content (for flex containers)

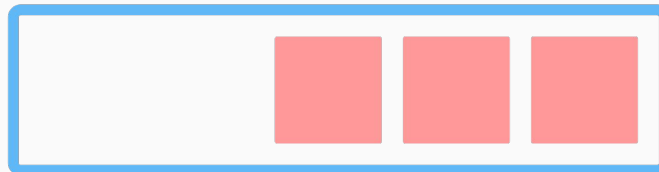
- Values for justify-content are
 - normal (default)
 - flex-start
 - start
 - flex-end
 - end
 - center
 - space-between
 - space-around
 - space-evenly

justify-content for a flexbox with flex-direction: row

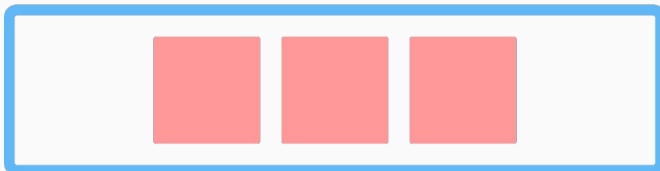
flex-start



flex-end



center



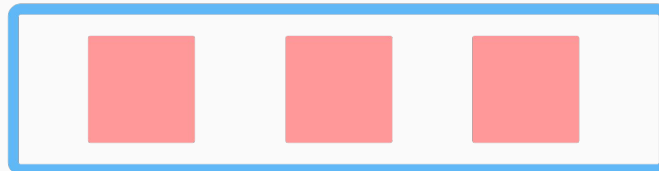
space-between



space-around



space-evenly



align-items (for flex containers)

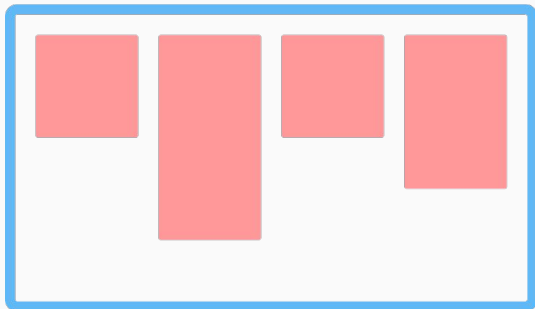
- The `align-items` property defines the distribution of space and flex items on the cross-axis

align-items (for flex containers)

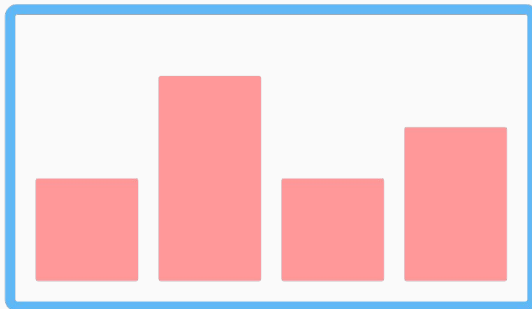
- Values for `align-items` are
 - `flex-start`
 - `start`
 - `flex-end`
 - `end`
 - `center`
 - `baseline`
 - `stretch` (default)

align-items for a flexbox with flex-direction: row

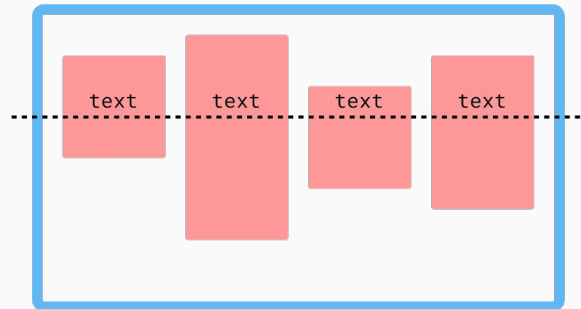
flex-start



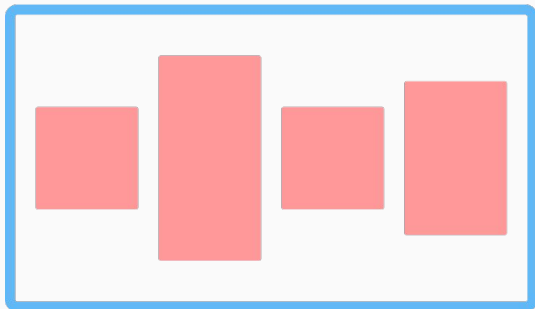
flex-end



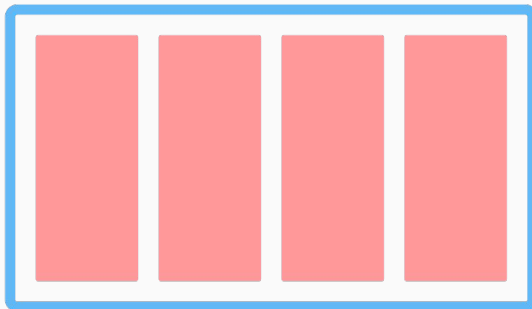
baseline



center



stretch



gap (for flex containers)

<code>

Use the `gap` property for gaps

```
.flexcontainer {  
  display: flex;  
  gap: 10px;  
}
```

<https://caniuse.com/flexbox-gap>

order (for flex items)

- The `order` property is used to order flex items other than the source order
- Flex items will be ordered in ascending order (from negative to positive)
- The default value of `order` is 0

order

<code>

Use the `order` property to change the displayed order of flex items.

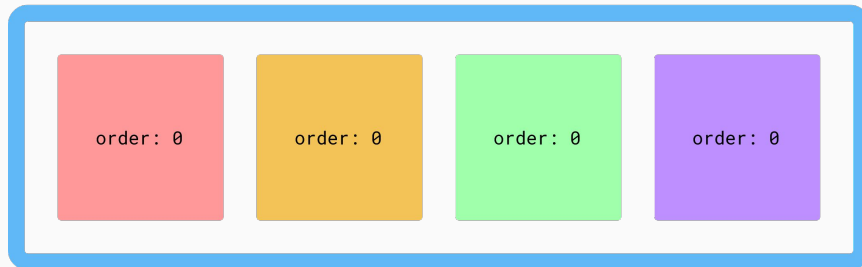
```
.item-1 {  
  order: 1;  
}  
  
.item-2 {  
  /* default */  
  order: 0;  
}  
  
.item-3 {  
  order: -10;  
}
```

```
<div class="item-1"></div>  
<div class="item-2"></div>  
<div class="item-3"></div>  
  
<!--  
<div class="item-3"></div>  
<div class="item-2"></div>  
<div class="item-1"></div>  
-->
```

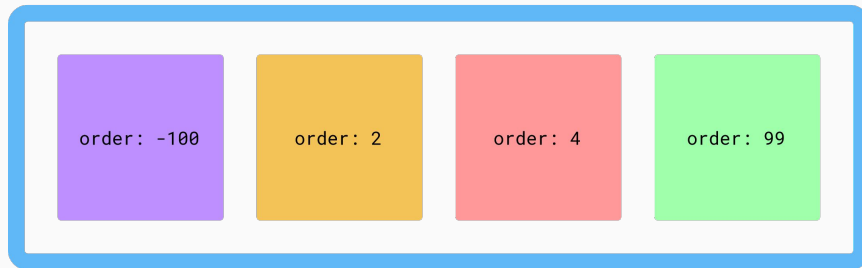


order for a flexbox with flex-direction: row

items with default order



items with custom order



flex-grow (for flex items)

- The `flex-grow` property defines how the remaining space is distributed among the flex items
- The `flex-grow` property takes a grow factor as number value

flex-grow (for flex items)

- Remaining space of the flexbox is divided by the total number of `flex-grow` values
- If all flex items have the same `flex-grow` value, the remaining space is shared equally

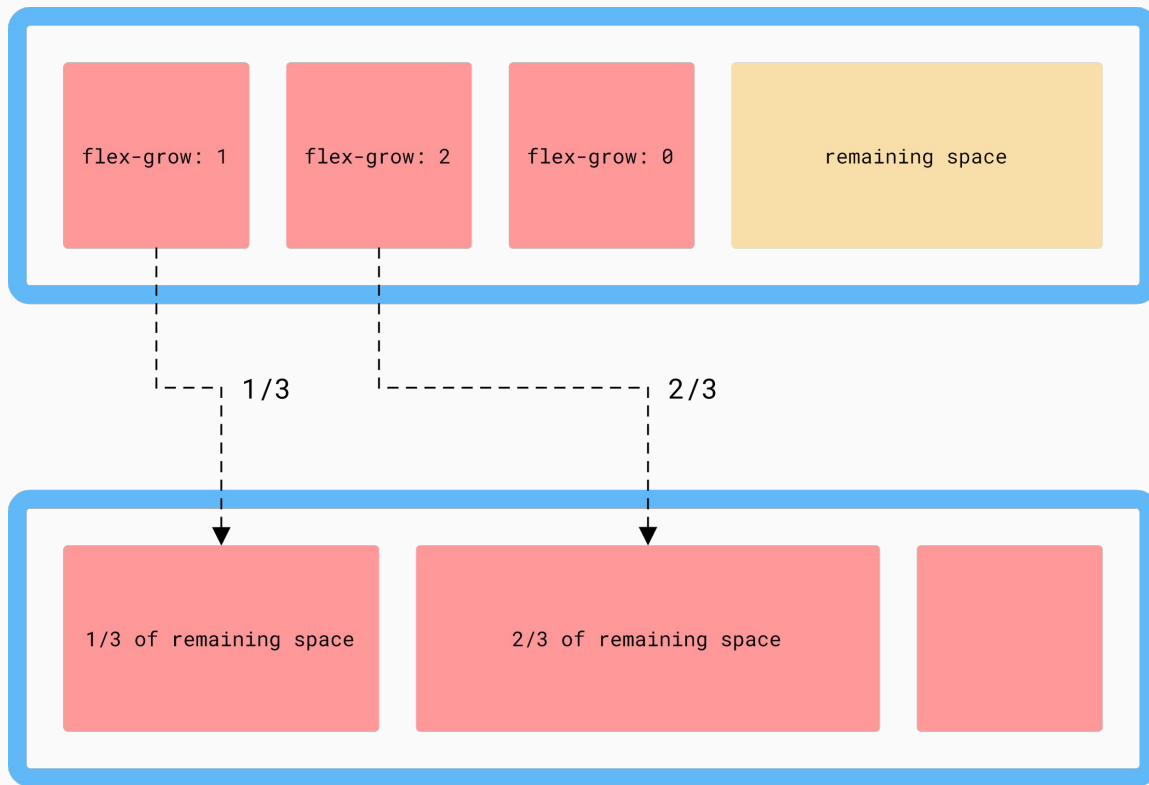
flex-grow

<code>

Flex-grow controls the distribution of remaining space.

```
.flex-item-1 {  
  flex-grow: 1; /* 1/3 of remaining space */  
}  
  
.flex-item-2 {  
  flex-grow: 2; /* 2/3 of remaining space */  
}  
  
.flex-item-3 {  
  flex-grow: 0; /* No remaining space will be distributed 🙄 */  
}
```

flex-grow



flex-shrink (for flex items)

- The `flex-shrink` property defines a shrink factor for each flex item
- Flex items will only shrink if there is not enough space available
- Default `flex-shrink` value is 1 → all items shrink equally

flex-shrink (for flex items)

<code>

Flex items will shrink by their `flex-shrink` value.

```
.flex-item-1 {  
  flex-shrink: 0; /* Will not shrink */  
}  
  
.flex-item-2 {  
  flex-shrink: 2; /* Shrinks by factor 2 */  
}  
  
.flex-item-3 {  
  flex-shrink: 1; /* Shrinks by factor 1 */  
}
```

flex-shrink

items overflow flex-container



items shrink by individual factor



flex-basis (for flex items)

- The `flex-basis` property sets the initial *main size* of a flex item
- The main size related to the current `flex-direction`
 - Horizontal main axis → `width` property (in ltr writing mode)
 - Vertical main axis → `height` property (in ltr writing mode)

flex-basis (for flex items)

- The default value for `flex-basis` is `auto` → will look for elements `width` or `height`
- A given length value for `flex-basis` will overwrite either the elements `width` or `height` property

flex-basis (for flex items)

<code>

In this example the `width` property is used, because `flex-basis` is set to `auto`

```
.flexbox {  
  display: flex;  
}  
  
.flex-child {  
  width: 500px;  
  height: 500px;  
  flex-basis: auto;  
}
```

flex-basis (for flex items)

<code>

In this example the `flex-basis` property takes precedence over `width`.

```
.flexbox {  
  display: flex;  
}  
  
.flex-child {  
  width: 500px;  
  height: 500px;  
  flex-basis: 100px;  
}
```

flex shorthand (for flex items)

- The `flex` property shorthand allows to set `flex-grow`, `flex-shrink` and `flex-basis` depending on the given value(s)

flex shorthand (for flex items)

<code>

Depending on the given value(s), the flex shorthand behaves differently.

```
flex: 5;
```

```
/* flex-grow: 5; flex-shrink: 1; flex-basis: 0%; */
```

```
flex: 100px;
```

```
/* flex-grow: 1; flex-shrink: 1; flex-basis: 100px; */
```

```
flex: 2 0;
```

```
/* flex-grow: 2; flex-shrink: 0; flex-basis: 0%; */
```

```
flex: 5 100px;
```

```
/* flex-grow: 5; flex-shrink: 1; flex-basis: 100px; */
```


align-self (for flex items)

- The `align-self` property allows flex items to be positioned individually on the cross-axis

align-self

<code>

Individual alignment of flex items can be done with `align-self`.

```
.flexbox {  
  display: flex;  
  align-items: flex-end;  
}
```

```
.flex-child {  
  align-self: flex-start;
```

YOLO 🤪



margin (for flex items)

- You can use `margin: auto` to position flex items on the main-axis to the start or end

margin (for flex items)

<code>

Alignment of flex items can be done with `margin: auto`.

```
.flexbox {  
  display: flex;  
}
```

```
.flex-child {  
  margin-left: auto;  
}
```

`margin-left: auto;`



flex-wrap (for flex containers)

- The items of the flexbox will always stay on one line
- If the flexbox contains more items than it can fit, it will overflow on the same line
- `flex-wrap` allows to wrap these items on multiple lines
 - Possible values: `no-wrap`, `wrap`, `wrap-reverse`

flex-wrap (for flex containers)

<code>

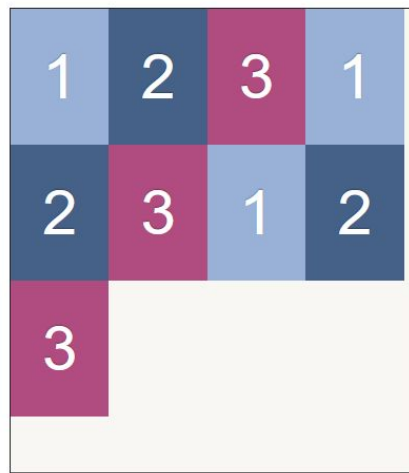
The flex-wrap property allows to wrap items on multiple lines.

```
.flexbox {  
  display: flex;  
  flex-wrap: wrap;  
}
```

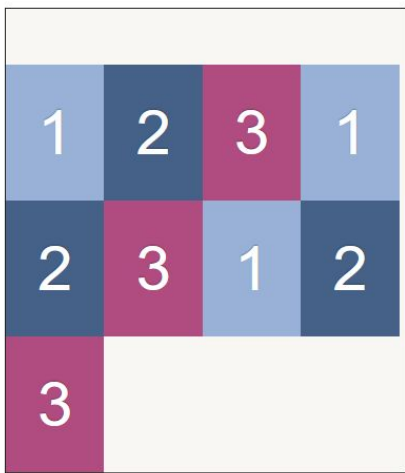
align-content (for flex containers)

- It is used to align flex items on the cross axis.
- It has only an effect when `flex-wrap: wrap` is applied.
- Possible values: `flex-start`, `flex-end`, `center`, `space-between`, `space-around`, `stretch` (default)

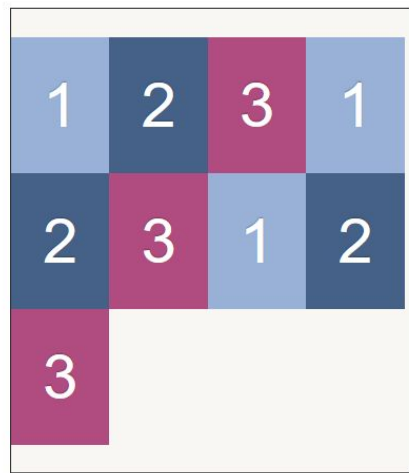
align-content (for flex containers)



align-content: start



align-content: end



align-content: center

Additional resources

- [Guide on flexbox on css-tricks](#)
- [Flexbox froggy](#)
- [Flexbox playground and editor](#)
- [Live flexbox tester](#)
- [Flexbox patterns](#)

Task

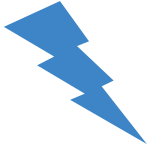
Flexbox



Aligning and centering

Center everything

Why / What you'll learn



- You often want to align/center an element
- You will learn the different techniques and when to use them

text-align

<code>

When you want to align a text, for example the text of a paragraph

```
p.center {  
  text-align: center;  
}  
p.right {  
  text-align: right;  
}
```

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Perferendis, molestiae.

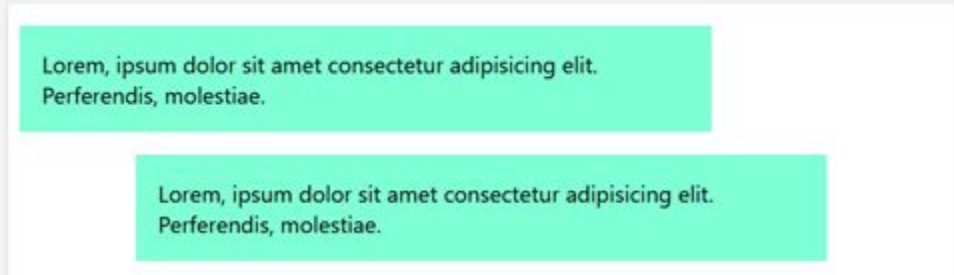
Lorem, ipsum dolor sit amet consectetur adipisicing elit. Perferendis, molestiae.

margin: auto

<code>

Centering block elements horizontally

```
p {  
  width: 80%;  
}  
p.centered {  
  margin-left: auto;  
  margin-right: auto;  
}
```



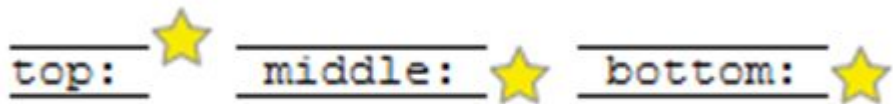
Lorem, ipsum dolor sit amet consectetur adipiscing elit.
Perferendis, molestiae.

Lorem, ipsum dolor sit amet consectetur adipiscing elit.
Perferendis, molestiae.

vertical-align

<code>

Only for images or table cells



```
  
  

```

Vertical centering

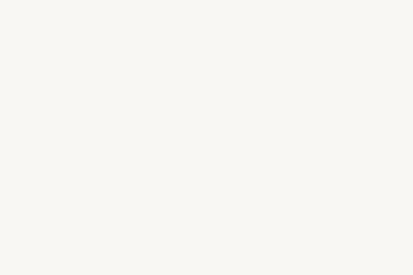
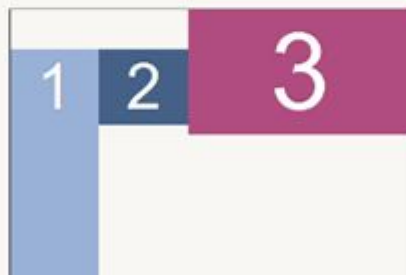
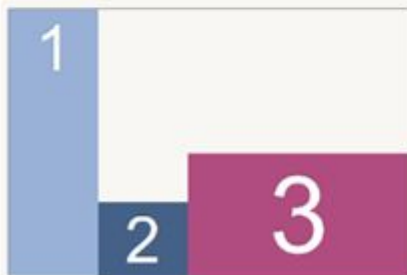
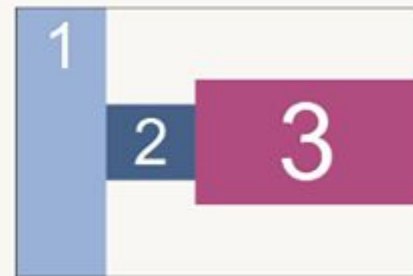
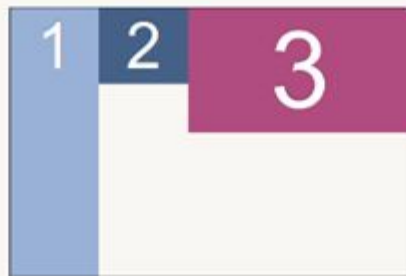
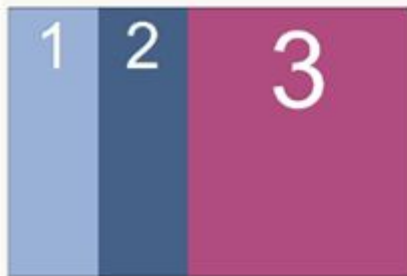
- To center one line of text you can try:
 - the same value for line-height and height
 - the same value for padding-top/padding-bottom
- Everything else: use flexbox or grid

Aligning with Flexbox 1

<code>

`align-items: stretch | flex-start | center | flex-end | baseline`

```
.flexbox {  
  display: flex;  
  align-items: ...  
}
```

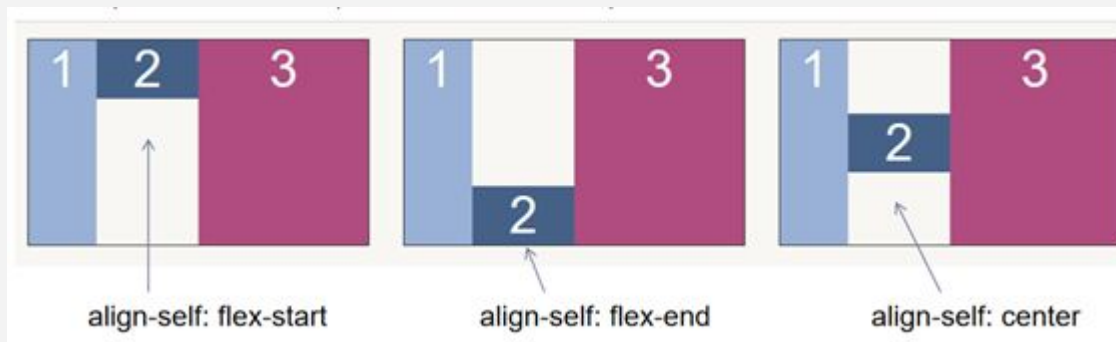


Aligning with Flexbox 2

<code>

`align-self: auto | stretch | flex-start | center | flex-end | baseline`

```
.flexbox {  
  display: flex;  
}  
.flex-item {  
  align-self: ...;  
}
```

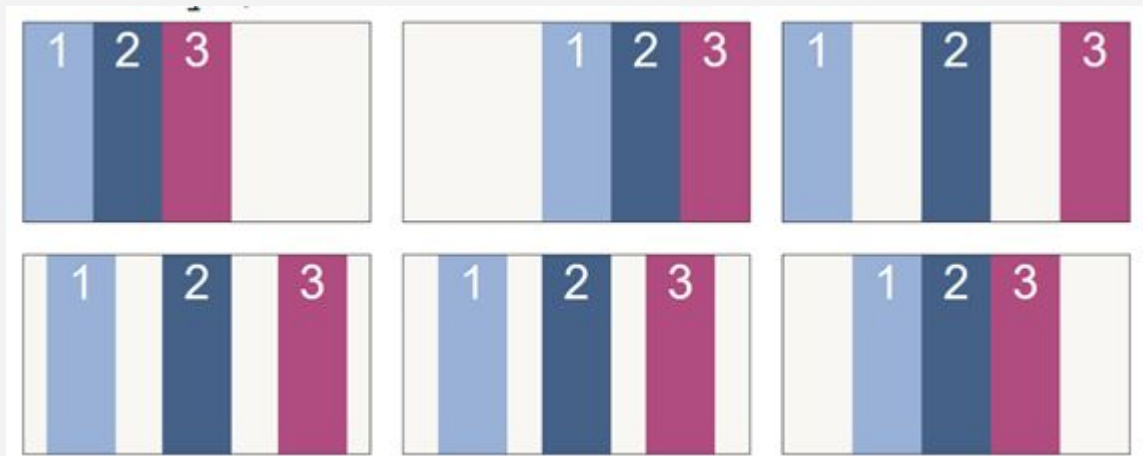


Aligning with Flexbox 3

<code>

```
justify-content: flex-start | flex-end | space-between | space-around  
| space-evenly | center
```

```
.flexbox {  
  display: flex;  
  justify-content: ..;  
}
```



Task

Centering/aligning with flexbox



Centering with Flexbox - 1

<code>

Use margins for the flex item

```
.flexbox {  
  display: flex;  
}  
.flex-item {  
  margin: auto;  
}
```

Centering with Flexbox - 2

<code>

Use align-items/justify-content for the flex container

```
.flexbox {  
  display: flex;  
  align-items: center;  
  justify-content: center;  
}
```

Useful links for centering and aligning

- <https://css-tricks.com/tips-aligning-icons-text/>
- <https://css-tricks.com/centering-css-complete-guide/>
- <https://web.dev/centering-in-css/>

Length Units

Why / What you'll learn



- Length units in CSS can be confusing - or can you quickly explain the difference between `em` and `rem`? 😊
- You will learn the difference between absolute and relative units

Length units

- The CSS length data type represents a distance value (i.e. `width` or `height` of an element)
- Length values consist of a number and a unit

Font-relative lengths

- Are used to define the size of text or elements
- The actual size depends on the element itself or a parent element
- Most used font-relative lengths are `em` and `rem`

em

- em stands for *equal m*
- Depends on the font size of the parent element / same element

em

<code>

In this case, em relates to the font-size of a parent element.

```
.parent {  
  font-size: 16px;  
}  
  
.child {  
  width: 1em; /* = 16px */  
}
```

em

<code>

In this case, both parent and the element define a font-size.

```
.parent {  
  font-size: 20px;  
}  
  
.child {  
  font-size: 10px;  
  width: 2em; /* 20px = 2 * 10px */  
}
```

rem

- rem stands for *root equal m*
- Represents the font-size of the root element
- Is often used in the context of responsive web design

rem

<code>

rem always relates to the globally defined font-size.

```
.child {  
  width: 1rem; /* = 16px */  
}
```

The default font-size of the browser is 16px



```
.another-child {  
  font-size: 100px;  
  width: 2rem; /* = 32px */  
}
```


rem

<code>

rem always relates to the globally defined font-size.

```
html {  
  font-size: 32px;  
}  
  
.child {  
  width: 1rem; /* = 32px */  
}  
  
.another-child {  
  font-size: 100px;  
  width: 2rem; /* = 64px */  
}
```

rem/em

Em: Problems with nested elements, e.g. lists

Rem: No problems with nested elements/lists

<https://codepen.io/FlorenceM/pen/eYRaoMV>

Other font-relative lengths


- `ch` = represents the width of the glyph “0” of the current element
- `ex` = represents the height of the letter “x” of the current element

Usage of ch unit

<code>

Since ch represents the approximate width of one character, it can be used to limit the maximum line length.

```
p {  
  max-width: 50ch;  
}
```



The ideal line length is between 45 and 75 characters.

See [article](#) on smashing magazine.

Viewport units

- Viewport units are relative to the current viewport
- Viewport is the visible portion of the browser

Viewport units

CSS defines the following viewport units

- `vw` = viewport width
- `vh` = viewport height
- `vmin` = smaller axis of the viewport
- `vmax` = larger axis of the viewport

Viewport units

<code>

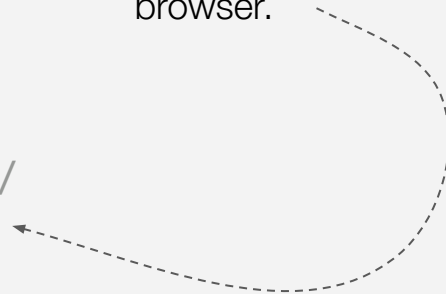
Some examples of viewport units.

```
/* Full width of the viewport */  
width: 100vw;
```

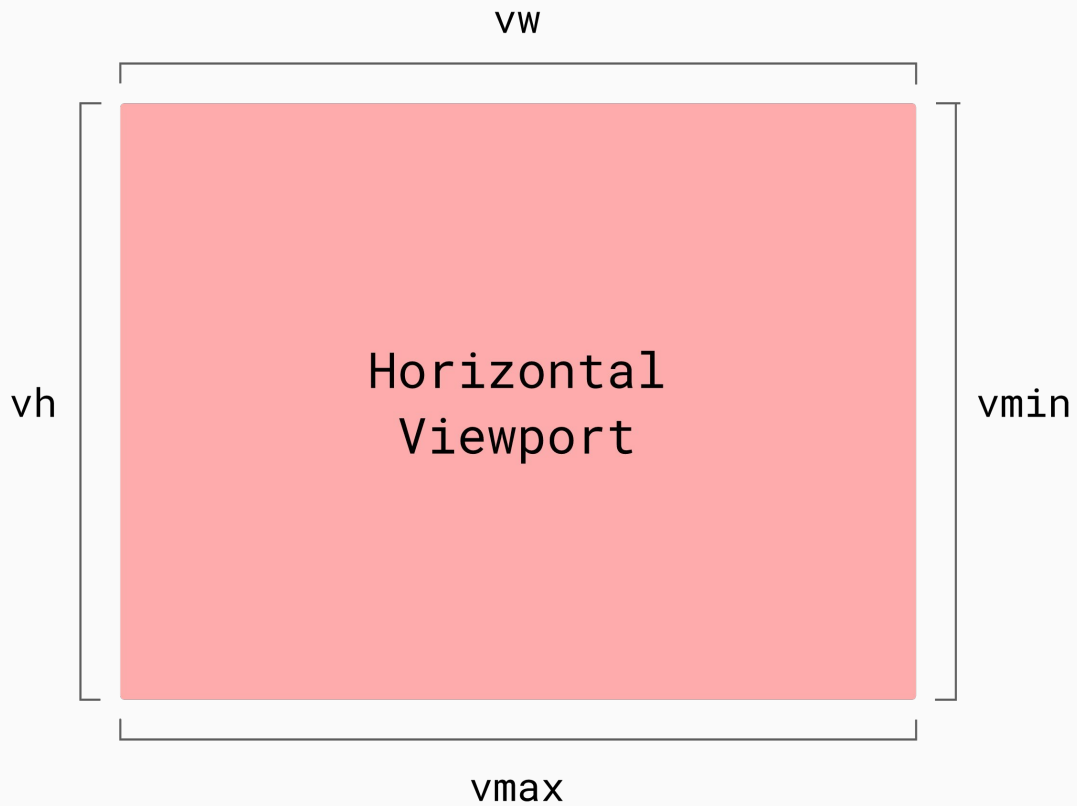
```
/* Half width of the viewport */  
width: 50vw;
```

```
/* 80% of the smaller side of the viewport */  
width: 80vmin;
```

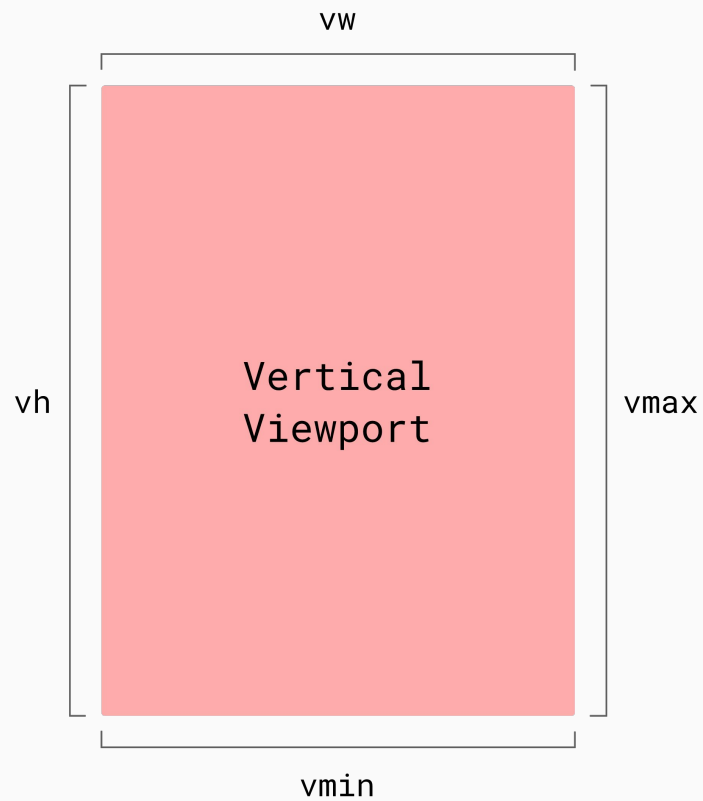
👍 I always use a vmin values for small demos that are displayed in the middle of the browser.



Viewport units horizontal viewports



Viewport units vertical viewport



Better Viewport Units

- Large Viewport Units (lvw, lvh etc.): represent largest possible viewport
- Small Viewport Units (svw, svh etc.): represent the smallest viewport
- Dynamic Viewport Units (dvw, dvh): takes the actual viewport in account (with/without keyboard etc.)

Absolute units

- Absolute units represent physical measurements of a device (screen, printer)
- Examples for absolute units are **px**, **cm**, **mm**

px

- In the old days **1px** represented an actual device pixel.
- For high resolution displays px became a logical unit (one CSS pixel does not necessarily correspond to one physical device pixel)
- Browsers scale the pixel to maintain legibility, ensuring consistency across various devices.

Absolute units

- $\text{cm} \rightarrow 1 \text{ cm} = 96\text{px}/2.54$
- $\text{mm} \rightarrow 1 \text{ mm} = 1/10\text{th of } 1 \text{ cm}$
- $\text{in} = \text{inch} \rightarrow 1 \text{ inch} = 96\text{px}$

Relative vs. absolute units

- Absolute units can cause accessibility issues because they don't scale when the user agents font-size changes
- Prefer relative units over absolute units when possible

Task

Length Units



@-Rules

Media queries and friends

@-rules tell CSS how to behave.

Why / What you'll learn



- A modern website will be used from a wide variety of devices, and this is where @-rules and media queries come into play
- Learn how to adapt your CSS to the needs of your user

Most important @-rules

- @media → queries for a specific device criteria
- @supports → queries for specific CSS feature
- @keyframes → defines a keyframe animation
- @font-face → defines an external font (to be downloaded)

Media queries

- @media rules are commonly known as *media queries*
- Media queries can query a specific feature of the user agent, device or environment
- Media queries are used for [responsive web design](#)

Media queries

- Examples for specific features are
 - Width and height of the viewport
 - Resolution
 - Orientation
 - User preferences
 - Input mechanism

Media types

- Media types are used to target a specific media
- Possible media types are
 - all
 - screen
 - print
 - speech

Media types

<code>

Print Stylesheet

```
.theme {  
  background-color: black;  
  color: white;  
}  
  
@media print {  
  .theme {  
    background-color: white;  
    color: black;  
  }  
}
```

Logical operators

<code>

Logical operators help to create specific media queries.

```
/* Combine media feature expressions */  
@media screen  
and (max-width: 1000px)  
and (orientation: landscape) { }
```

```
/* comma as logical or */  
@media print, screen { }
```

```
/* Every media type but screens */  
@media not screen { }
```


Width

- Is used to query the current width of the rendering surface
- Should be used with a media type
- Can be prefixed with min or max
 - `min-width: 100px` is equal to `width >= 100px`
 - `max-width: 100px` is equal to `width <= 100px`

Width

<code>

Media queries for width are commonly used for RWD.

```
html {  
  font-size: 100%;  
}  
  
/* equals width >= 600px */  
@media screen and (min-width: 600px) {  
  html {  
    font-size: 110%;  
  }  
}
```

Viewport meta

<code>

Don't forget to use the meta-element to set the viewport, otherwise it won't work as intended.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Screen orientation

- The orientation media feature can be used to query for the current screen orientation
- Supported values are `landscape` and `portrait`
- Orientation asks for the current aspect ratio of the viewport, NOT the device orientation

Screen orientation

<code>

Orientation asks for aspect ratio, not device orientation.

```
@media (orientation: landscape) {  
  /* viewport width > viewport height */  
}
```

```
@media (orientation: portrait) {  
  /* viewport height > viewport width */  
}
```

Mobile first vs. desktop first

- When using media queries for responsive web design, there are two possible approaches
 - Mobile first
 - Desktop first

Mobile first

- With mobile first the default expected device is a mobile
- When the viewport gets larger additional rules are applied

Mobile first

<code>

Example for a mobile first approach.

```
/* default 1 column layout */
.content {
  display: block;
}

/* 3 column layout for larger screens */
@media screen and (min-width: 60em) {
  .content {
    display: grid;
    grid-template-columns: 20% 60% 20%;
  }
}
```


Desktop first

- With desktop first the default expected device is desktop computer
- When the viewport gets smaller additional rules are applied

Desktop first

<code>

Example for a desktop first approach.

```
/* default 3 column layout */
.content {
  display: grid;
  grid-template-columns: 20% 60% 20%;
}

/* 1 column layout for small screens */
@media screen and (max-width: 30em) {
  .content {
    display: block;
  }
}
```

Responsive, adaptive - WHAT?

- What is responsive design?
- And what is adaptive design?
- And what about liquid or static?
- [Article](#) from kulturbaunause
- Example adaptive: <https://www1.wdr.de/>
- Example responsive: <https://www.tagesschau.de/>

Task

Media Queries



User preferences - dark or light mode?

<code>

```
body {  
  background-color: #fff;  
  color: #000;  
}  
  
@media (prefers-color-scheme: dark) {  
  body {  
    background-color: #000;  
    color: #fff;  
  }  
}
```

Input type: hover or not?

<code>

Show `.additional` on hover - if hover is possible, otherwise show it directly

```
.additional {  
  opacity: 1;  
}  
@media (hover: hover) {  
  .button + .additional {  
    opacity: 0;  
  }  
  .button:hover + .additional {  
    opacity: 1;  
  }  
}
```

<https://codepen.io/FlorenceM/full/zYLXxwG>

Media Queries vs Container Queries

- Media queries can “only” handle the whole screen width or height
- Sometimes you need media queries for single elements. These are called *container queries*
 - See [article](#) from css tricks
 - [Can use container queries](#)

Container Queries

```
.container {  
  container-type: inline-size;  
}  
  
@container (max-width: 30em) {  
  .card {  
    /* special formatting if the container is less than 30em wide */  
  }  
}
```

<https://codepen.io/FlorenceM/pen/gOjdOJO>

Grid

Basics

CSS Grid is the efficient and modern way of building complex layouts.

Why / What you'll learn



- Complex layouts usually required tons markup and CSS
- You will learn how to use CSS Grid to build layouts with minimal code used

CSS Grid

- CSS Grid suits best for two dimensional layouts
- To define a CSS Grid, set the **display** property of an element to `grid` or `inline-grid`

CSS Grid vs CSS Flexbox



Figure 1 Representative Flex Layout Example



Figure 2 Representative Grid Layout Example

CSS Grid

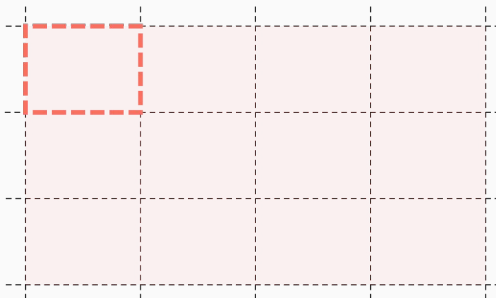
- The CSS Grid is defined through columns and rows
- Child elements of the grid container are placed automatically row-wise by default
- But items can also be placed individually

grid terminology

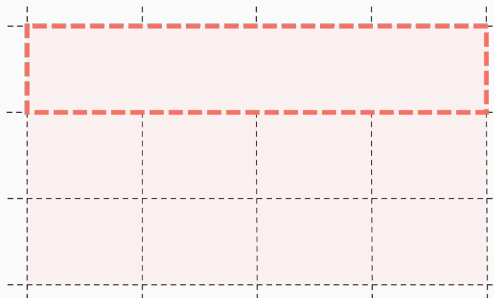
Grid



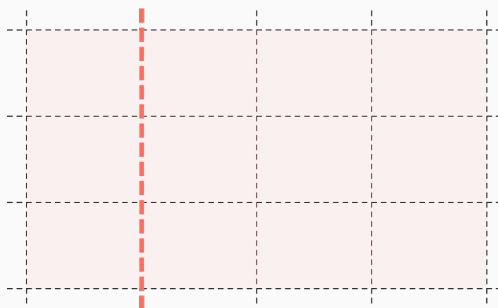
Grid cell



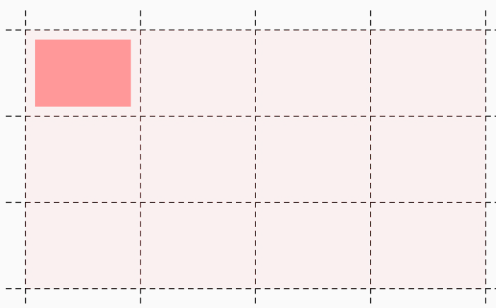
Grid track



Grid line



Grid item



Grid area



grid-template-columns

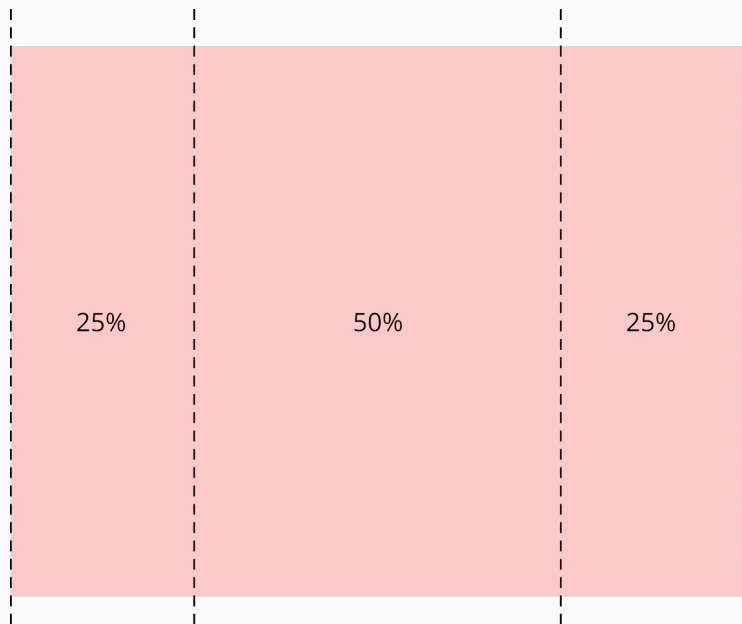
- The `grid-template-columns` property defines the columns of the grid
- It takes a list of values defining the width of columns

grid-template-columns

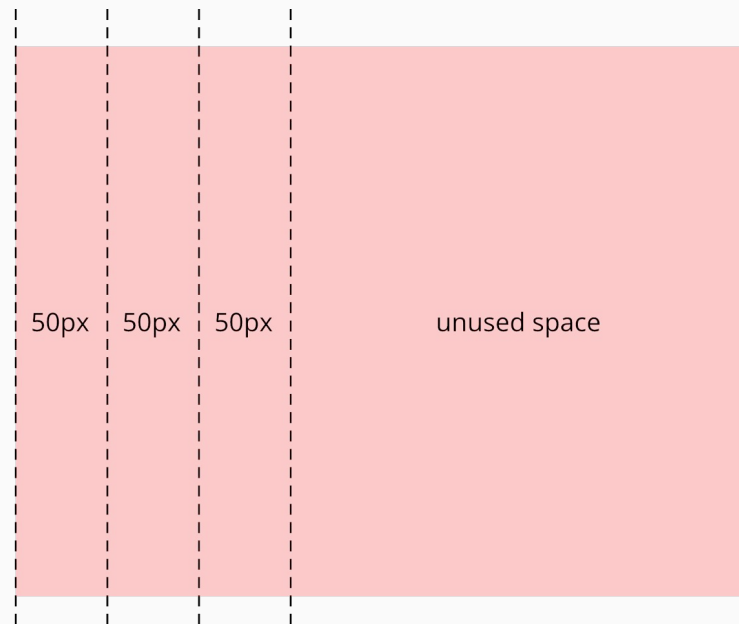
- Possible values for defining the width of grid columns are
 - Length values → i.e. px or rem
 - Percentage values → 30% of the grid container
 - Fractional values → the fr unit represents a fraction of the available space

grid-template-columns

`grid-template-columns: 25% 50% 25%`

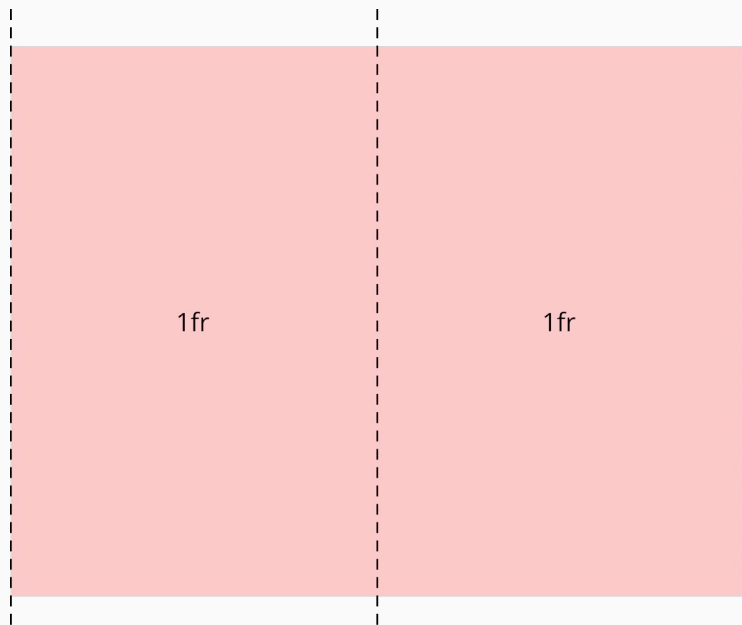


`grid-template-columns: 50px 50px 50px`

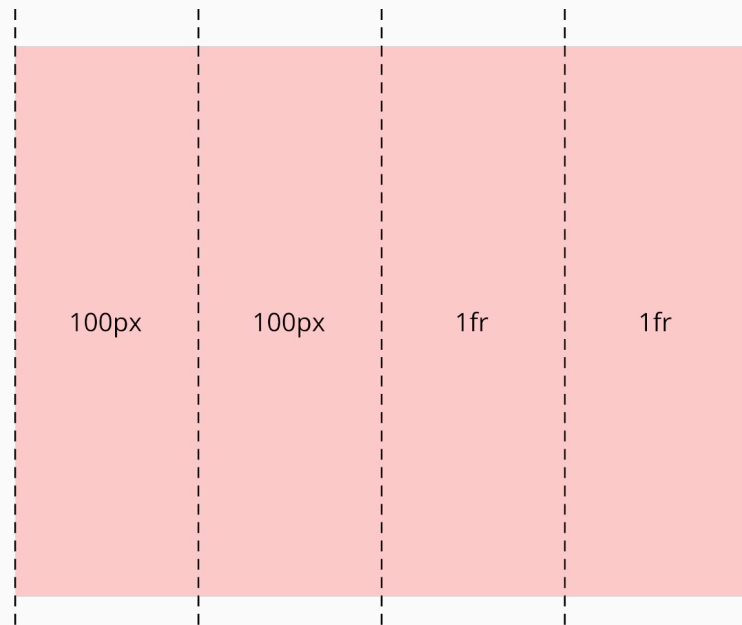


grid-template-columns

`grid-template-columns: 1fr 1fr`



`grid-template-columns: 100px 100px 1fr 1fr`

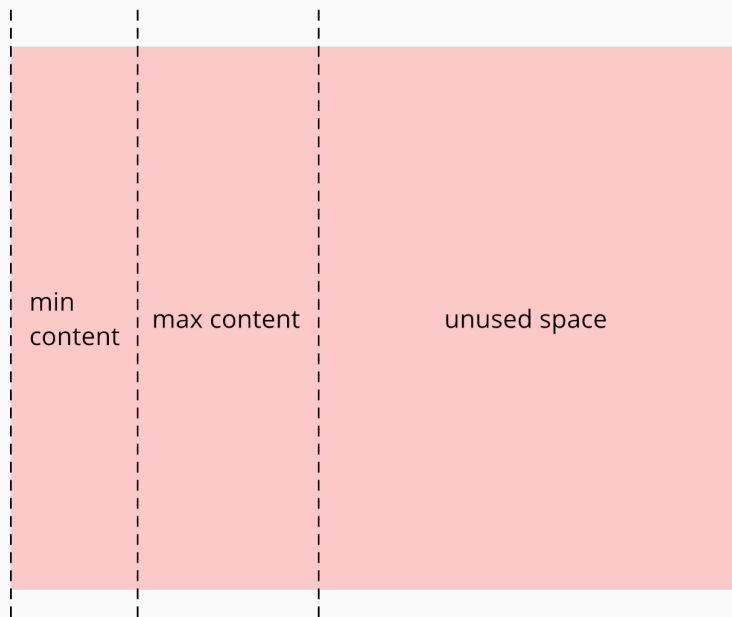


min-content and max-content

- It is possible to use the `min-content` and `max-content` keywords to define the column width
- `min-content` → the column has the smallest possible width to fit the content
- `max-content` → the column is wide enough to fit its content

min-content and max-content

```
grid-template-columns: min-content max-content
```



minmax() function

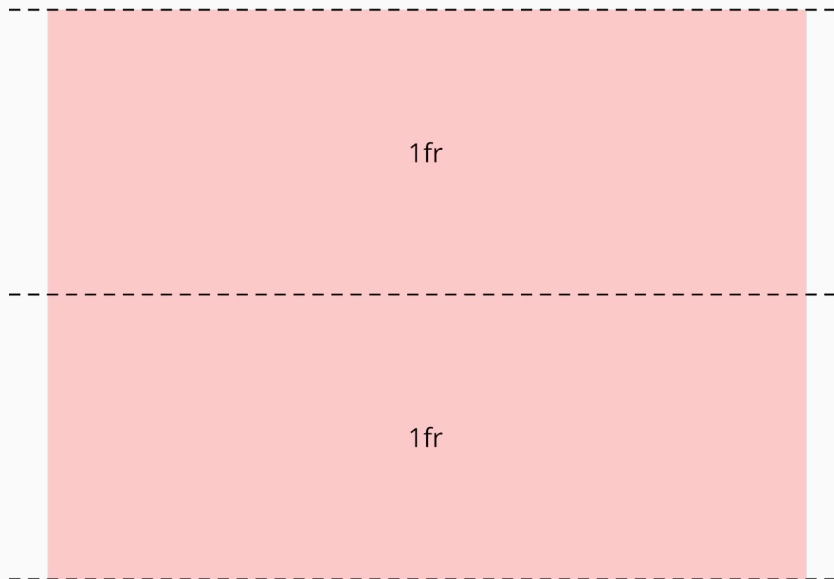
- The `minmax(min,max)` function can be used to define a minimal and maximal size (of a column or row)
- Accepts the following values
 - Length values → i.e. px, fr or percentages
 - max-content or min-content keywords

grid-template-rows

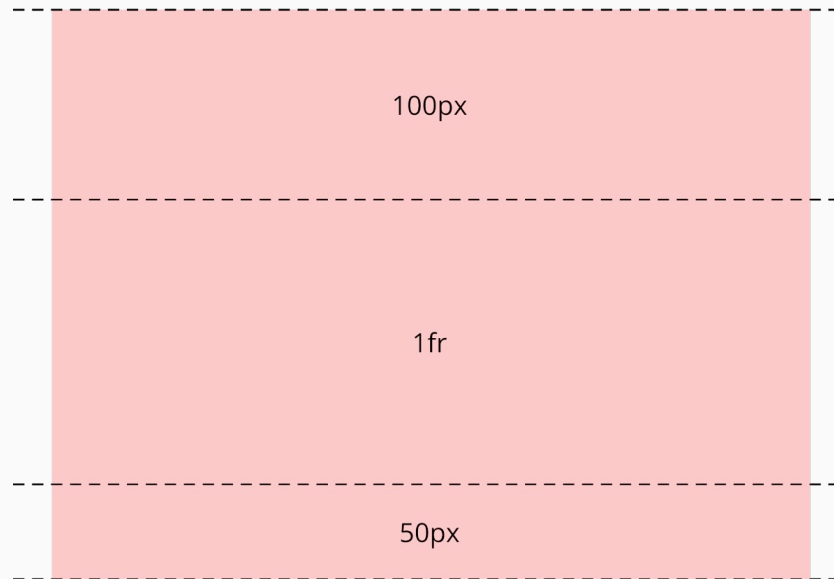
- The `grid-template-rows` property defines the rows of the grid
- It accepts a list of values defining the row widths (like `grid-template-columns`)

grid-template-rows

grid-template-rows: 1fr 1fr



grid-template-rows: 100px 1fr 50px



repeat function

<code>

The `repeat(n, size)` function is used to define repetitive columns or rows.

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(4, 100px);  
  /* grid-template-columns: 100px 100px 100px 100px; */  
}
```

grid-template

- The `grid-template` property is a shorthand for `grid-template-columns` and `grid-template-rows`
- The values for `grid-template-columns` and `grid-template-rows` are separated with a slash


grid-template

<code>

Example usage of the grid-template property shorthand.

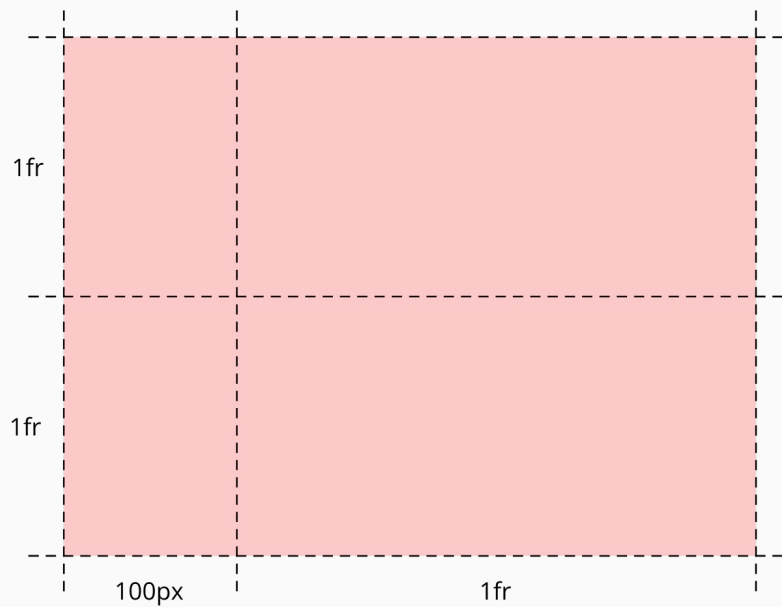
```
.grid {  
  display: grid;  
  /*  
  grid-template-columns: 100px 200px;  
  grid-template-rows: 1fr 400px;  
  */  
  grid-template: 100px 200px / 1fr 400px;  
}
```

Convenient for small grids and maybe a bit messy for larger grids.

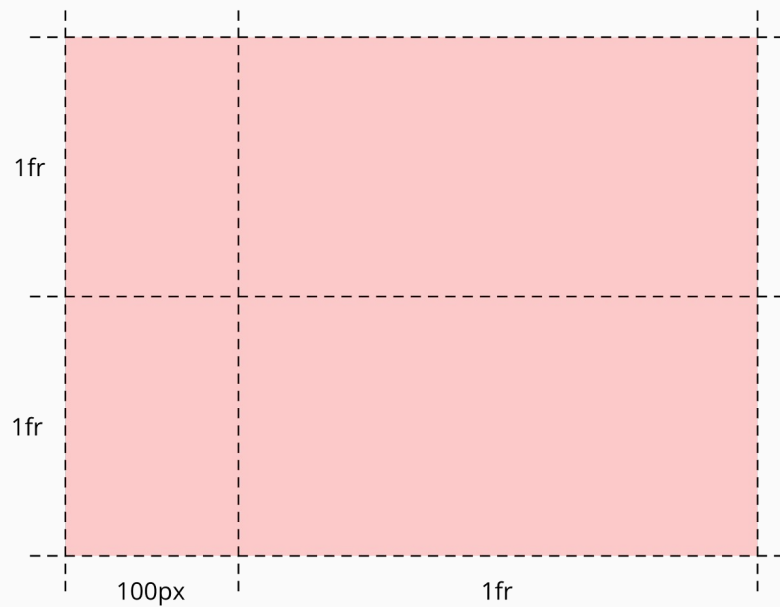


grid-template

```
grid-template-columns: 100px 1fr  
grid-template-rows: 1fr 1fr
```



```
grid-template: 100px 1fr / 1fr 1fr
```



Gap

- Grid columns and rows can have a gap (aka. gutter)
- A gutter can be defined with the `grid-column-gap/column-gap` or `grid-row-gap/row-gap` properties
- The `grid-gap/gap` shorthand can set both gap values

Gap

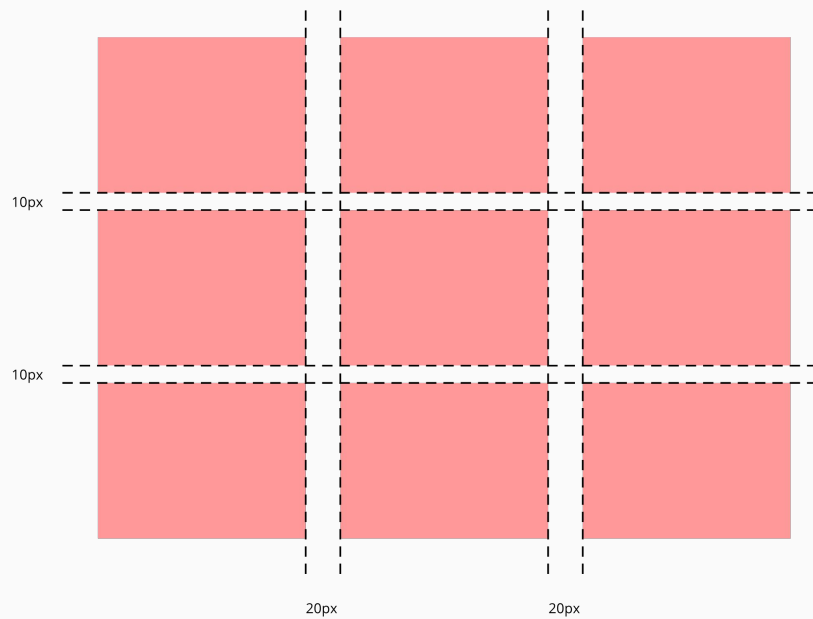
<code>

Brief example of the gap properties.

```
.grid {  
  display: grid;  
  
  grid-column-gap: 10px;  
  grid-row-gap: 20px;  
  
  /* grid-gap shorthand */  
  grid-gap: 10px 20px;  
}
```

grid-gap

grid-gap: 20px 10px

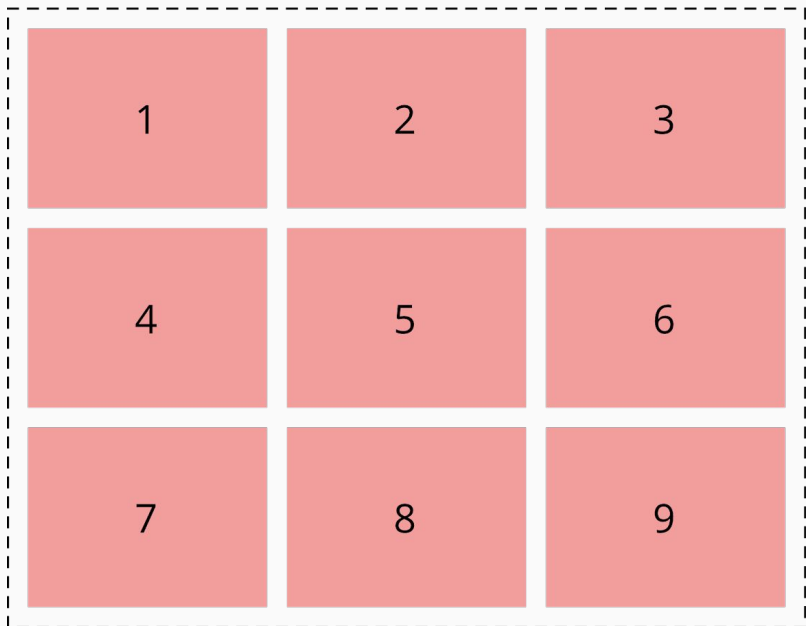


grid-auto-flow

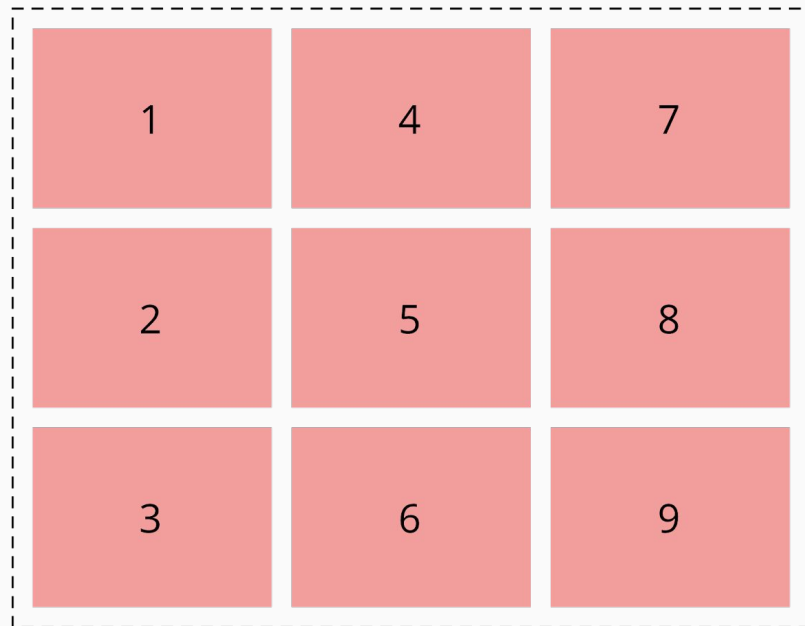
- The grid-auto-flow property defines how auto-placement for grid items is done
- Possible values are
 - row → grid items are auto placed on grid rows (default)
 - column → grid items are auto placed on grid columns

grid-auto-flow

grid-auto-flow: row



grid-auto-flow: column

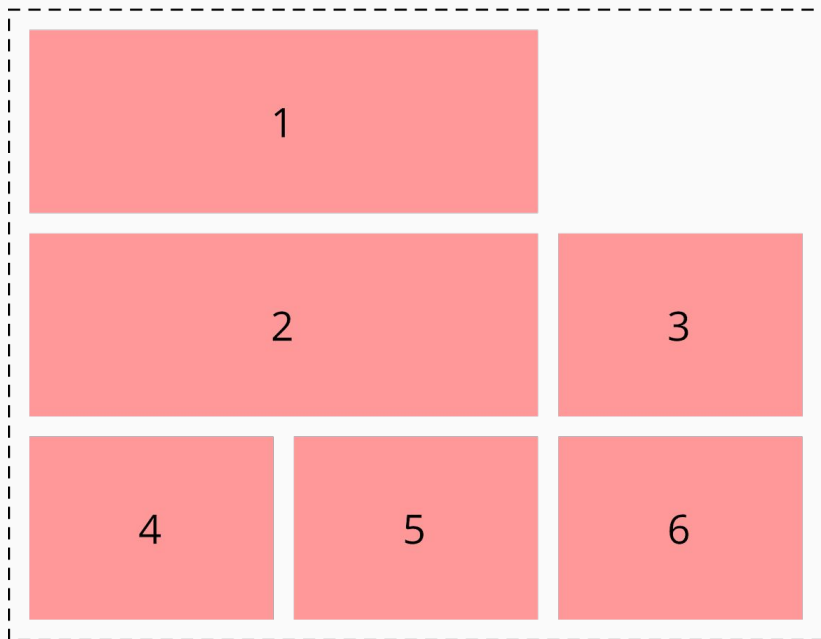


grid-auto-flow

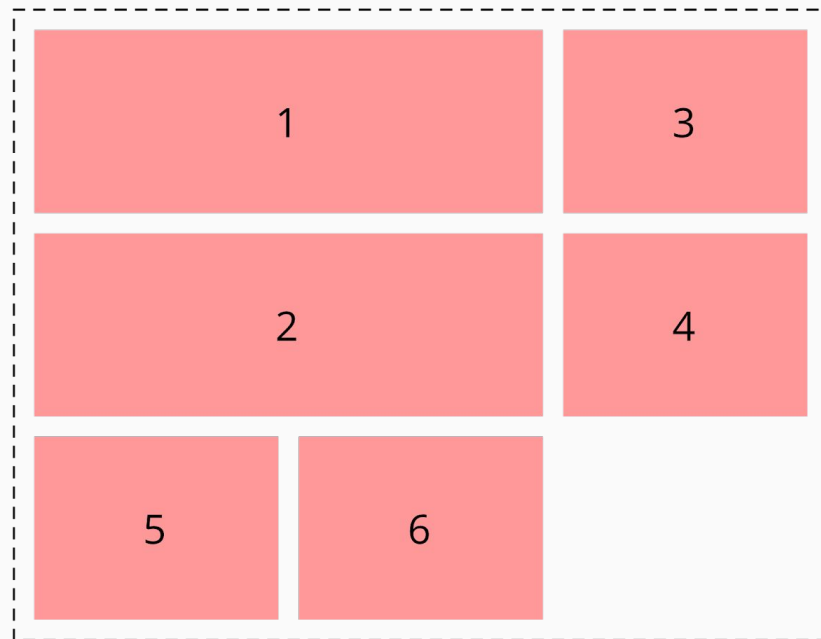
- Its possible to add the `dense` keyword to the row and column keywords
- `dense` tries to fill any holes in the grid
- `dense` will eventually mix up the original order of grid items

grid-auto-flow with dense keyword

grid-auto-flow: row



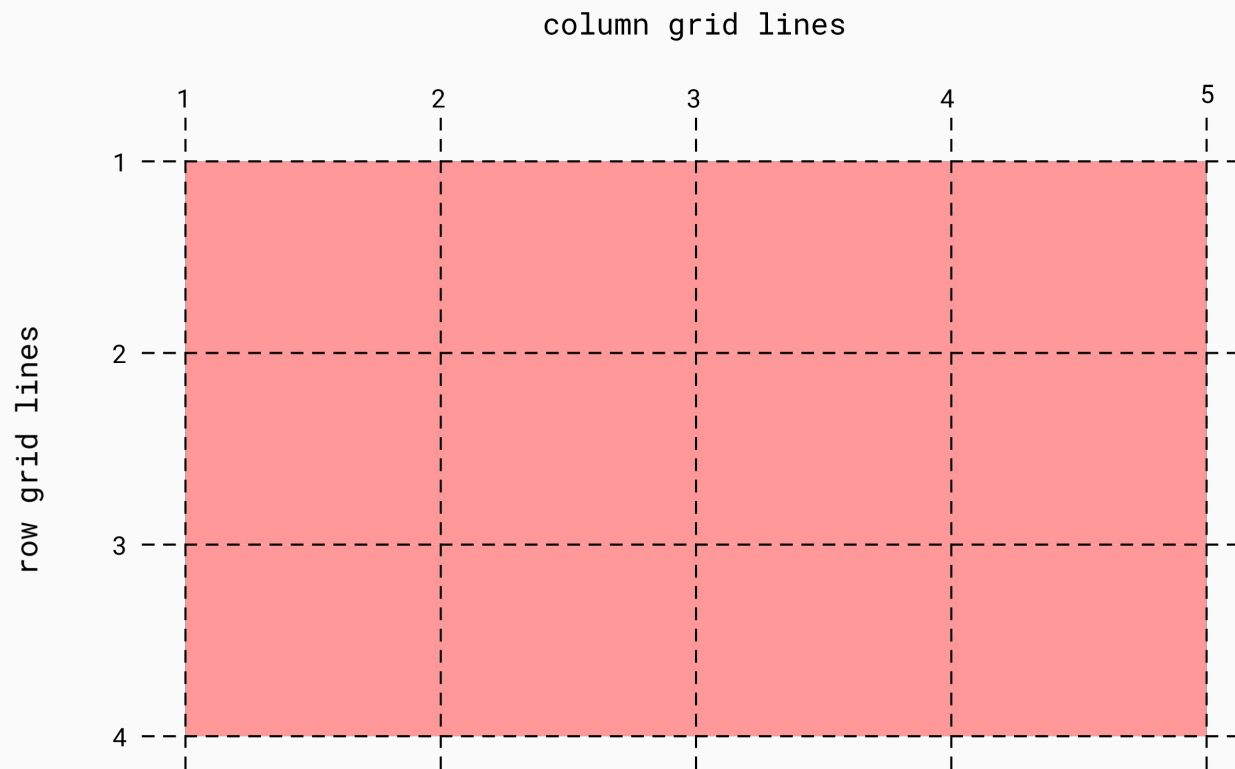
grid-auto-flow: row dense



Grid lines

- Grid items can be placed according to grid lines
- Grid lines are placed
 - before the first column / row
 - between columns and rows
 - after the last column / row

Grid lines



grid-column-start / grid-column-end

- The properties `grid-column-start` and `grid-column-end` are used to place grid items within grid lines
- The `grid-column` property shorthand will set both values

grid-column-start / grid-column-end

- `grid-column-start: 1` will set the start position on the first grid line
- `grid-column-end: 4` will set the end position on the fourth grid line
- Negative numbers will count in reverse from the last grid line

grid-column-start / grid-column-end

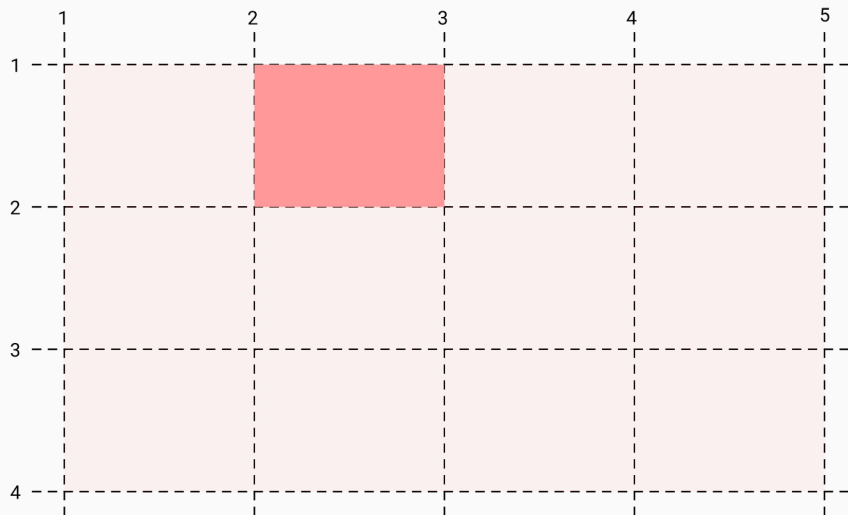
<code>

Example for grid item placement using the grid lines.

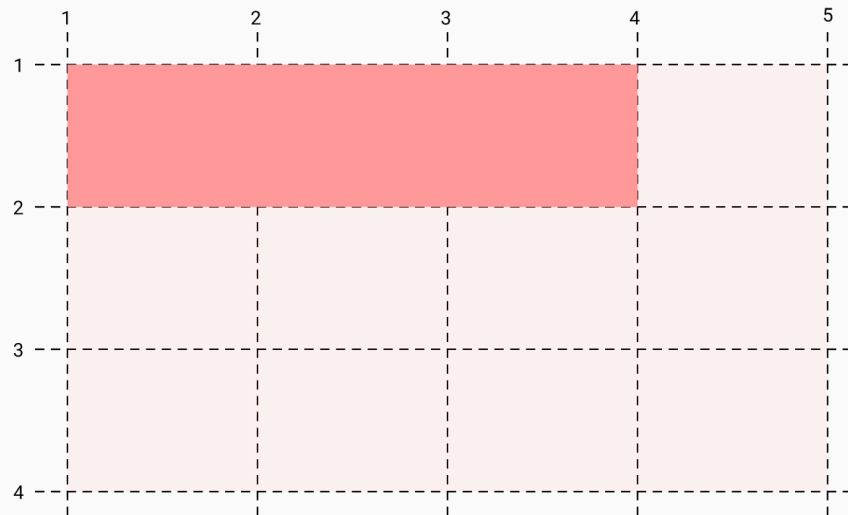
```
.grid {  
  display: grid;  
  grid-template-columns: 100px 100px 100px;  
}  
  
/* the grid item will span across all columns */  
.item {  
  grid-column-start: 1;  
  grid-column-end: 4;  
}
```


Grid lines

grid-column-start: 2

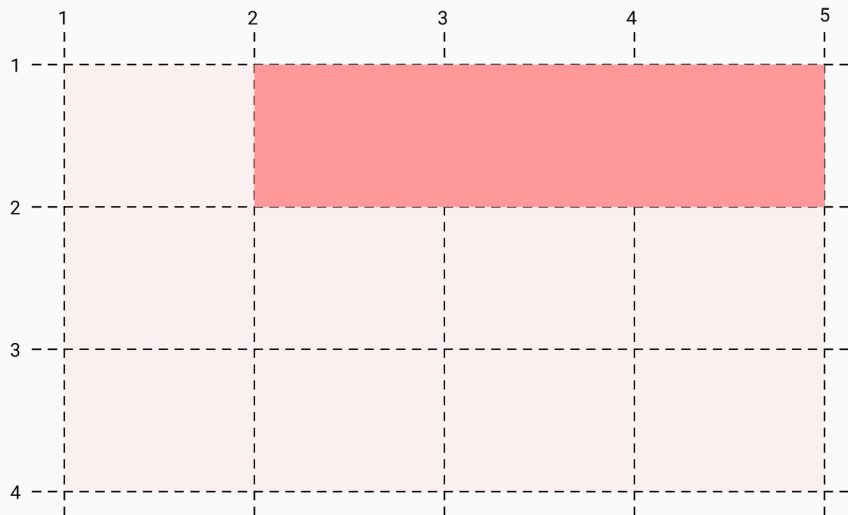


grid-column-end: 4

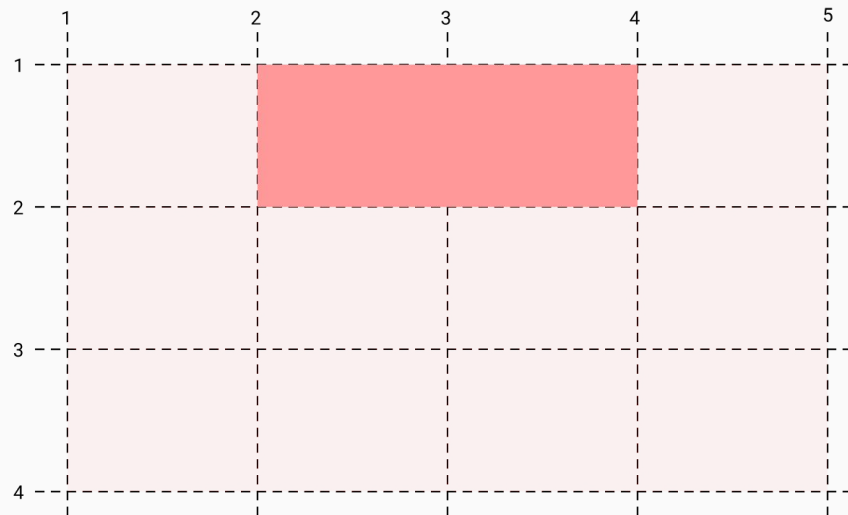


Grid lines

grid-column: 2 / 5



grid-column: 2 / -2



span keyword

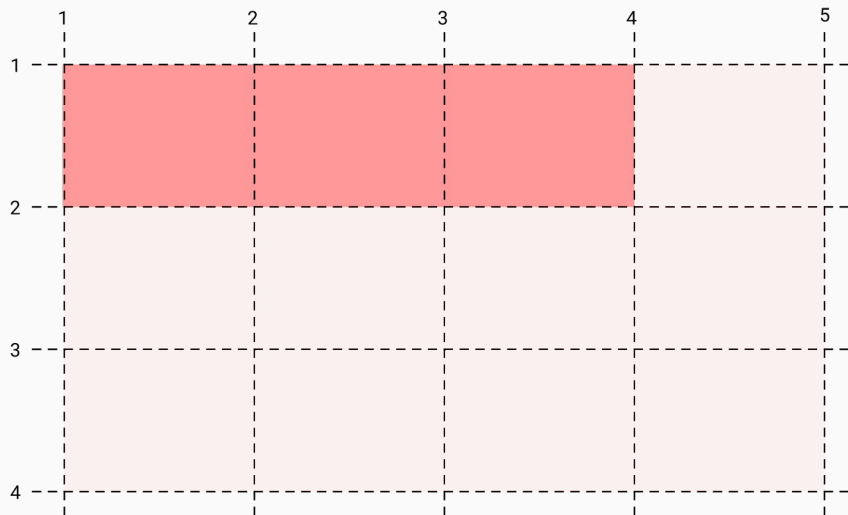
- `grid-column-start` and `grid-column-end` can be combined with the `span` keyword
- The `span` keyword is followed by a number, which indicates the number of columns or rows to be consumed by the grid item

span keyword

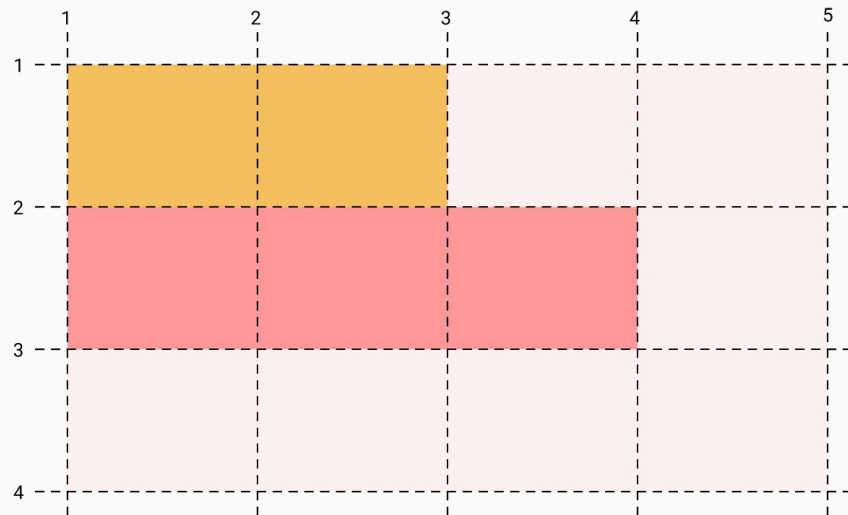
- `grid-column-start: span 2` will span the item from its (automatic) start position across 2 grid lines
- `grid-column-end: span 3` will span the item from its (automatic) end position across 3 grid lines to the start

span keyword

grid-column-start: span 3



grid-column-start: span 3



grid-row-start / grid-row-end

- The properties `grid-row-start` and `grid-row-end` work like the properties for columns, but instead for rows
- The `grid-row` shorthand will set both properties

Task

Place items in a grid



grid-template-areas

- The `grid-template-areas` property can define named areas within a grid
- Named grid areas are defined per grid row as a string
- Column names are separated with a space
- Undefined areas can be marked with a dot

grid-template-areas

<code>

Example for the usage of grid-template-areas.

```
.grid {  
  display: grid;  
  grid-template: repeat(4, 100px) / repeat(3, 100px);  
  
  grid-template-areas:  
    "header header header header"  
    ". main main ."  
    "footer footer footer footer"  
}  
  
.header { grid-area: header; }  
.main { grid-area: main; }  
.footer { grid-area: footer; }
```

Task

Responsive Grid



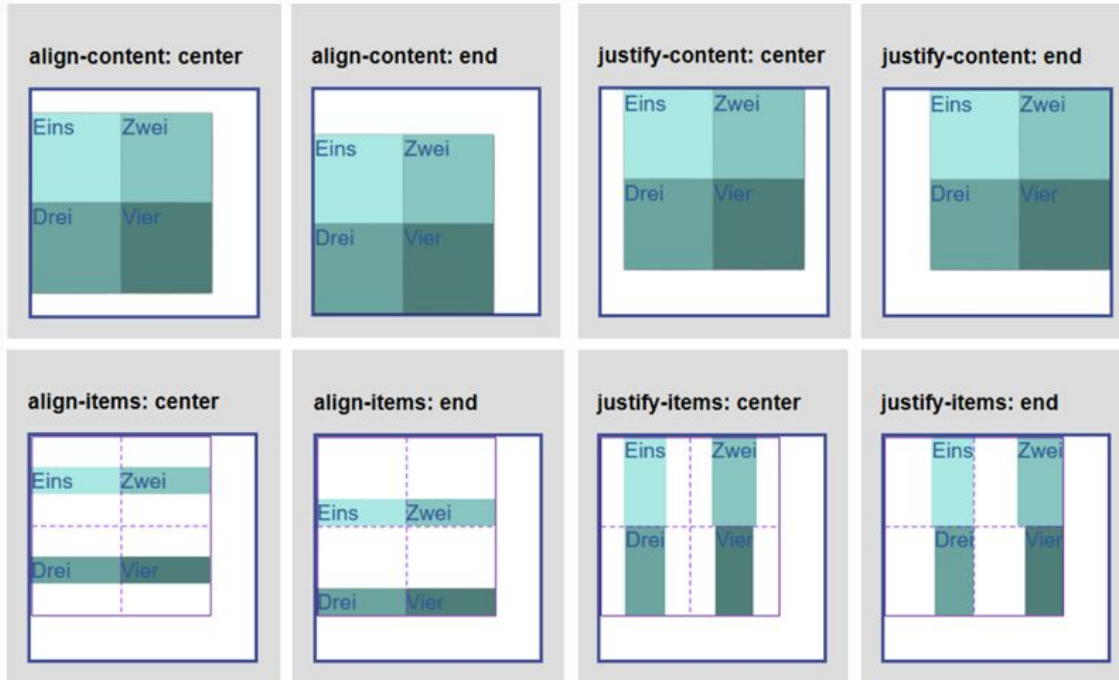
Grid advanced

more possibilities

Aligning items with grid

- align-content: alignment of the grid as a whole, vertically
- justify-content: Alignment of the grid as a whole, horizontally
- align-items: alignment of the grid items, vertically
- justify-items: alignment of the grid items, horizontally

Aligning items with grid



justify-content and align-content

- The `justify-content` and `align-content` properties come into play when the grid items don't take up the available space of the grid
- The properties are used to place the grid items horizontally and vertically

justify-content and align-content

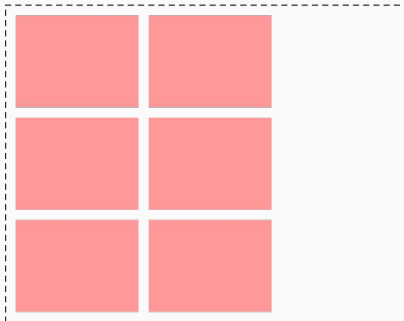
- `justify-content` is used for horizontal placement of grid items
- `align-content` is used for vertical placement
- Possible values are
 - `start`, `end`, `center`, `stretch`, `space-around`, `space-between`, `space-evenly`

place-content

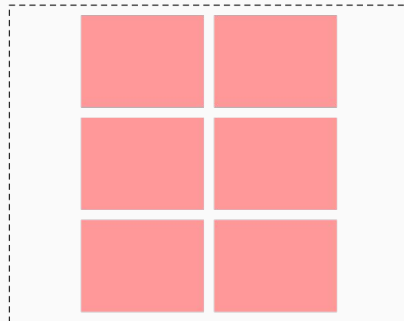
- The `place-content` property shorthand is used to both values for `align-content` and `justify-content`

justify-content

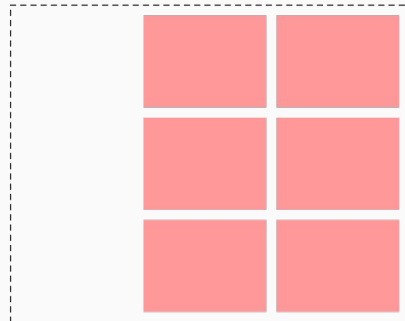
justify-content: start



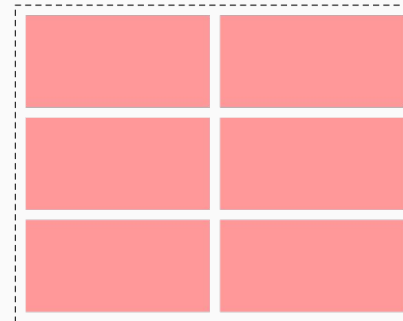
justify-content: center



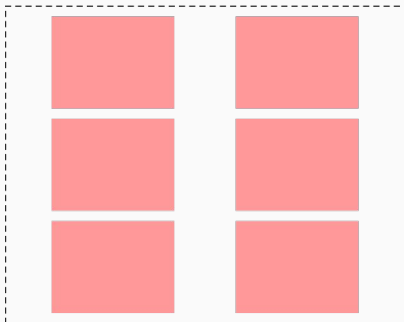
justify-content: end



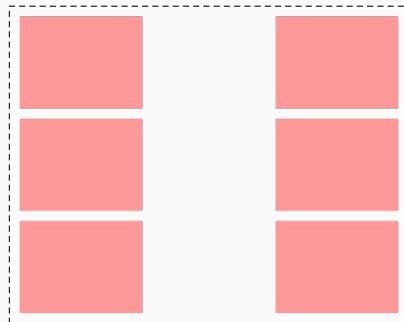
justify-content: stretch



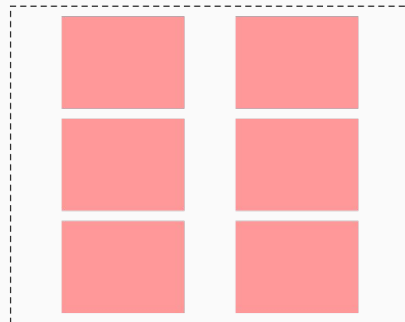
justify-content: space-around



justify-content: space-between



justify-content: space-evenly



justify-items and align-items

- If a grid item does not fill a grid cell, the justify-items and align-items properties can be used to align the item horizontally and vertically
- Possible values are
 - start, end, center, stretch (default)

place-items

- The `place-items` property shorthand is used to set both values for `justify-items` and `align-items`

justify-self and align-self

- The properties `justify-self` and `align-self` can be used to place a grid item individually in a grid cell

completely flexible grid

- You can create a grid, which adapts to every viewport.
- The ingredients:
 - `repeat()` with `autofit`
 - `minmax()`
- <https://codepen.io/FlorenceM/full/powmVLr>

completely flexible grid

<code>

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(auto-fit, minmax(120px, 1fr));  
  grid-auto-rows: minmax(120px, auto);  
  grid-auto-flow: dense;  
  grid-auto-columns: minmax(120px, auto);  
}
```

repeat function with autofit

<code>

The browser decides how many columns fit.

```
.grid {  
  display: grid;  
  grid-template-columns: repeat(autofit, 100px);  
}
```

minmax() function

- The `minmax(min,max)` function can be used to define a minimal and maximal size (of a column or row)
- Accepts the following values
 - Length values → i.e. px, fr or percentage
 - max-content or min-content keywords

Additional resources

- [Grid Garden](#)
- [Article on CSS Grid](#) by Kulturbanause
- [Complete Guide Grid](#) on css-tricks
- Online Grid Editor [griddy.io](#)
- [Grid by example](#)

Transitions

Websites with the right amount of animations
and transitions are a joy to use.

Why / What you'll learn



- Modern UX uses animations and transitions to direct the users attention
- Transitions are fun 🎉

Transitions

- A transition defines an animated change from one state to another state
 - Element → hovered element
 - Notification invisible → Notification visible
- A transition can be either triggered by JavaScript or user interaction

Transitions

- The `transition` property is a shorthand for
 - `transition-property`
 - `transition-duration`
 - `transition-timing-function`
 - `transition-delay`

Transitions

<code>

This transition will be animated from a transparent background color to salmon.

```
.background-color {  
  transition: background-color .5s;  
}
```

```
.background-color:hover {  
  background-color: salmon;  
}
```

transition-property

- The `transition-property` property defines what properties should be animated during the transition
- Can be multiple properties

transition-property

<code>

Multiple values for transition-property are separated with comma.

```
.transition {  
  
    /* One property */  
    transition-property: opacity;  
  
    /* Multiple properties */  
    transition-property: opacity, transform;  
}
```

transition-property

Nearly all properties are animatable, but only a few should be animated.

Transitions and render performance

- Properties that change the layout (width, height, padding, etc.) should not be animated → bad performance
- Properties that only change the appearance (color) can be animated → good performance
- Opacity and transform have the best performance → handled by the GPU

transition-duration

- The transition-duration property defines the duration of the transition
- The default duration is 0s
- Can be defined in
 - Seconds → i.e. 1s or .5s
 - Milliseconds → 500ms

Recommended transition duration

- There is no golden rule for the best transition duration
- Material Design defines recommended durations by the size of the animated element
 - Small (i.e. checkbox) → 100ms
 - Medium (i.e. bottom sheet) → 200ms to 250ms
 - Large (i.e. card) → 250ms to 300ms

transition-timing-function

- The `transition-timing-function` defines how intermediate values are calculated
- Can be defined with
 - keyword values
 - `steps(n)` function
 - `cubic-bezier()` function

transition-timing-function

- The predefined keywords are
 - **ease** → slow start, then fast, slow end
 - **linear** → same speed from start to end
 - **ease-in** → slow start
 - **ease-out** → slow end
 - **ease-in-out** → slow start and end
- See <https://codepen.io/FlorenceM/full/ExXzRrj>

Multiple transitions

- Multiple transitions can be defined per element
- When using the transition shorthand each transition definition is separated with comma
- When using the single properties the values are separated with comma

Multiple transitions

<code>

Transitions can animate multiple properties at the same time.

```
transition: opacity .2s, transform .4s;
```

```
/* These two transitions do the same thing */
```

```
transition-property: opacity, transform;
```

```
transition-duration: .2s, .4s;
```

Transition and hover

<code>

Will the transition work in both directions? See [example](#).

```
div {  
  background-color: lightblue;  
}  
  
div:hover {  
  background-color: salmon;  
  transition: background-color 1s;  
}
```

Transitions and hover

- If the transition is defined in the `:hover` selector, the transition will only work in one direction

CSS games using transitions and animations

- CSS only platform [game](#)
- CSS only [hogs shooter](#)

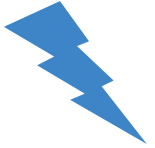
More resources on rendering

- [csstriggers](#): List of animatable properties and performance impact
- [Anatomy of a frame](#)
- [Pixel Pipeline](#)
- [Martin Splitt: Life of a pixel](#)

Transform

Transform is the main ingredient of beautiful and performant web animations.

Why / What you'll learn



- Modern UX uses animations to guide users
- Performant animations can't be done without the transform property
- Transform allows us to visually manipulate an element in 2D or 3D space without disrupting the normal document flow

Transform

- The `transform` property modifies the coordinate space of elements
- `transform` provides several transform functions
 - i.e. `translate`, `scale`, `rotate`, `skew`
- Transform functions can be concatenated

Translate

- The `translate(tx, ty)` function repositions an element on the horizontal and/or vertical axis
- Takes two [length](#) values

Translate

- Dedicated transform functions for single axis use cases
 - `translateX(tx)`
 - `translateY(ty)`

Translate with absolute values

- When translate is called with absolute values (i.e. px, rem or viewport units), the element is repositioned with the passed absolute value

translate(tx, ty)

<code>

Usage of the `translate` function with two absolute values.

```
transform: translate(100px, 200px);
```




Will move the element 100px on the x-axis to the right (ltr direction) and 200px down on the y-axis

translate(tx, ty)

<code>

When called with a single parameter, **ty** will be 0.

```
/* transform: translate(100px, 0); */  
transform: translate(100px);
```



When only one parameter is given, the element will not be positioned on the vertical axis.

Translate with percentage values

- When `translate` is called with percentage values, the percentage value always refers to the size of the element

translate(tx, ty)

<code>

Percentage values passed to `transform` always refer to elements size.

```
.transform {  
  width: 100px;  
  height: 100px;  
  transform: translate(200%, 50%);  
}
```

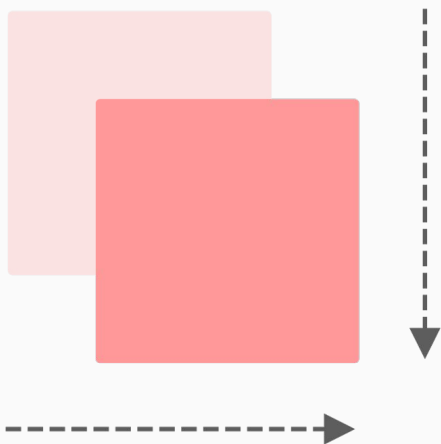
The element will be moved 200px to the right (ltr direction) and 50px to the bottom.



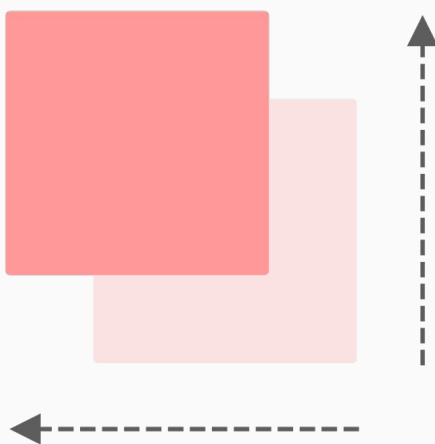
Translate

The `translate` function repositions an element on the x- and/or y-axis.

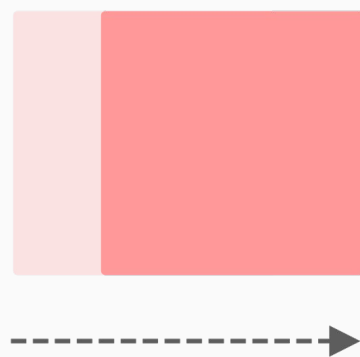
`translate(40px, 40px)`



`translate(-40px, -40px)`



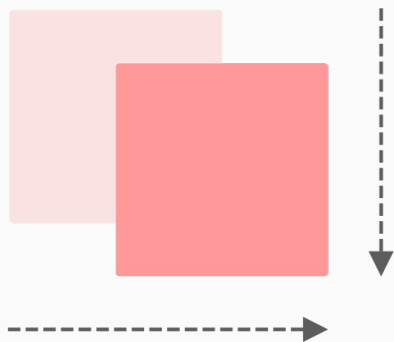
`translate(40px)`



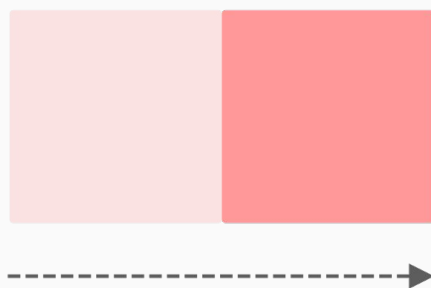
Translate

Percentage values always refer to the size of the transformed element.

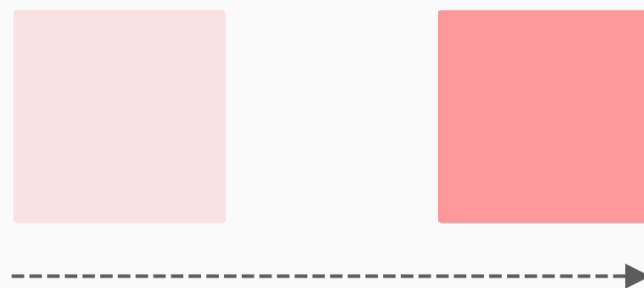
`translate(50%, 25%)`



`translate(100%)`



`translate(200%)`



TranslateZ

- `translateZ(d)` repositions an element on the z-axis
- $d > 0 \rightarrow$ element becomes larger
- $d < 0 \rightarrow$ element becomes smaller

TranslateZ

- `translateZ` has no effect without a given `perspective`
- `Perspective` defines the distance between the screen and the user
- Small `perspective` values will look more dramatic than high values

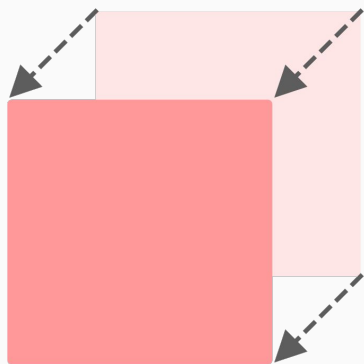
TranslateZ

- The perspective can be defined using the
 - `perspective(d)` function on same element
 - `perspective` property on any parent element

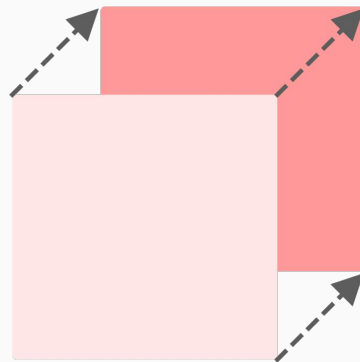
TranslateZ

`translateZ` will move an element on the z-axis.

`translateZ(40px)`



`translateZ(-40px)`



Rotate

- The `rotate(a)` function will rotate an element around a fixed point
- Fixed point is defined by `transform-origin` property
- Default `transform-origin` value is the center of the element

Rotate

- The rotation angle a is specified as an [angle](#) value
- Positive angles will rotate clockwise
- Negative angles will rotate counter clockwise
- Use `rotateX(a)` and `rotateY(a)` to rotate an element on a single axis

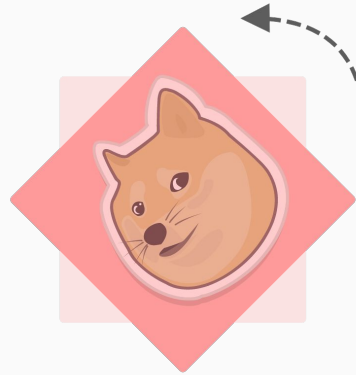
Rotate

Positive angles rotate clockwise and negative ones counter clockwise.

`rotate(45deg)`



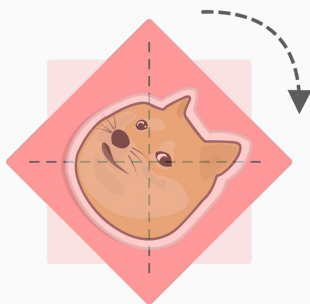
`rotate(-45deg)`



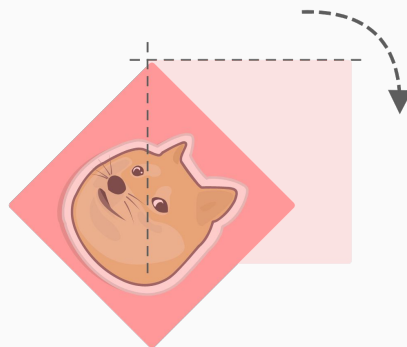
Rotate and transform-origin

The `transform-origin` defines the fixed point for the rotation.

```
rotate(45deg)  
+  
transform-origin: 50% 50%  
  (center)
```



```
rotate(45deg)  
+  
transform-origin: 0% 0%  
  (top left corner)
```



transform-origin

- The `transform-origin` property defines an elements origin for transformations
- `transform-origin: x-offset y-offset z-offset`
- By default `transform-origin` is the center of the element
 - `transform-origin: 50% 50% 0`

transform-origin

- `transform-origin` property accepts
 - length
 - percentage
 - keyword values

transform-origin

- The keyword values are
 - left = 0%
 - right = 100%
 - center = 50%
 - top = 0%
 - bottom = 100%

transform-origin keywords

<code>

The keyword values are quite convenient.

```
transform-origin: 50% 50%;  
transform-origin: center center;
```

```
transform-origin: 0% 100%;  
transform-origin: left bottom;
```

The two grouped declarations mean the same.

Scale

- The `scale(s)` function is used to scale elements (change their size)
- The scale factor `s` is specified as [number](#)

Scale

- Scale factor 1 is the default size
 - 0.5 is half the elements size
 - 2 is twice the elements size
 - With a scale factor of 0, the element will be invisible
 - -1 will flip the whole element

Scale

- Use `scale(s)` to scale the whole element
- Use `scale(sx, sy)` to scale x and y axis differently
- Use `scale(sx, sy, sz)` to perform a 3d scale on the x, y and z axis
- Use `scaleX(s)` or `scaleY(s)` to perform a scale operation on a single axis

Scale

Examples for different scale factors.

`scale(1)`



`scale(.5)`



`scale(2)`



`scale(-1)`



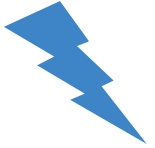
Task

Transform and Transitions



CSS Animations

Why / What you'll learn



- How to use animations for stunning effects
- When to prefer animations instead of transitions 🎉

Animations

Animations consist of two components

- Keyframes (@keyframes) for the start and end states of the animations
(+ intermediate waypoints)
- The description of the animation with the animation property

Animations - example

<code>

The animation starts immediately

```
@keyframes fade-in {  
  0% {  
    opacity: 0;  
  }  
  100% {  
    opacity: 1;  
  }  
}  
a {  
  animation: fade-in 5s ease;  
}
```

name of the animation

Animations - @keyframes

<code>

With intermediate waypoints

```
@keyframes fade-in {  
  0% {  
    opacity: 0;  
  }  
  50%, 80% {  
    opacity: .8;  
  }  
  100% {  
    opacity: 1;  
  }  
}
```


Animations - @keyframes

<code>

Instead of 0% and 100% you can use „from“ and „to“

```
@keyframes slide-in {  
  from {  
    transform: translateX(0%);  
  }  
  to {  
    transform: translateX(100%);  
  }  
}
```

Animations - animation property

<code>

The `animation` property defines how the animation will be applied

```
a {  
  animation: fade-in 5s ease;  
}  
/* or more explicit */  
a {  
  animation-name: fade-in;  
  animation-duration: 5s;  
  animation-timing-function: ease;  
}
```

animation-* part 1

- `animation-name`: specifies the name
- `animation-duration`: length of time for one cycle
- `animation-timing-function`: timing of the animation
- `animation-delay`: delay between the time the element is loaded and the beginning of the animation sequence

animation-* part 2

- `animation-iteration-count`: how often?
- `animation-direction`: should the animation change direction after each run?
- `animation-fill-mode`: what values are applied by the animation before and after it is executing?
- `animation-play-state`: to pause and resume the animation sequence

Animations - path animation

<code>

offset-path defines a movement path to follow during the animation

```
@keyframes move {  
  0% {  
    offset-distance: 0;  
  }  
  100% {  
    offset-distance: 100%;  
  }  
}  
  
#motion-demo {  
  offset-path: path('M20,20 C20,100 200,0 200,100');  
  animation: move 3000ms infinite alternate ease-in-out;  
}
```

Comparison: animation and transition

<code>

```
@keyframes fade-in {
  0% {
    opacity: 0;
  }
  100% {
    opacity: 1;
  }
}
a {
  animation: fade-in 5s ease;
}

a {
  opacity: 0;
  transition: opacity 5s ease;
}
a:hover {
  opacity: 1;
}
```

Difference between animation and transition

- transition needs a defined state to change (hover, focus) or a class
- animation can start directly
- animation can repeat as often as you want
- animation: allows for more fine tuning

Quiz time - what to use?

- Loading spinner
- Pulsing button
- Page transition
- Animated hover effect

Useful for animations: **box-shadow**

<code>

```
/* offset-x | offset-y | blur-radius | spread-radius | color */  
box-shadow: 2px 2px 2px 1px rgba(0, 0, 0, 0.2);
```

Task

Animation



Animate.css - a lot of useful animations

<code>

```
<!-- include it-->
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.m
in.css">
<!-- use it -->
<h1 class="animate__animated animate__bounce">An animated element</h1>
```

Animation libraries

- <https://animate.style/>
- <https://animejs.com/>
- <https://greensock.com/gsap/>

CSS Custom Properties

Why / What you'll learn



- Custom properties are the key to customizable components
- Custom properties can be used for real time theming

Custom Properties

- Custom properties can be changed during runtime
- They can be used to create customizable web components

Custom Properties

- Custom Properties are defined with the double dash prefix
 - i.e. `--brand-color: salmon`
- A custom property can then be used with the `var()` function
 - i.e. `background-color: var(--brand-color)`

Custom Properties

<code>

Custom properties are defined with the double dash prefix and used with the `var()` function.

```
:root {  
  --color: black;  
}  
  
p {  
  color: var(--color);  
}
```

Custom Properties

<code>

Minimal approach for theming using custom properties.

```
body {
  --color: black;
  --background: white;
  background-color: var(--background);
}

.dark-theme {
  --color: white;
  --background: black;
}

p {
  color: var(--color);
}
```

Inheritance

<code>

Custom properties are inherited.

```
body {  
  --color: blue;  
  color: var(--color);  
}
```

```
main {  
  --color: red;  
}
```

```
p {  
  color: var(--color);  
}
```

```
<body>  
  <main>  
    <p>Paragraph in main</p>  
  </main>  
  <p>Paragraph in body</p>  
</body>
```

Fallback value

- What happens if a custom property is undefined?
- The `var()` function takes a second argument, which represents a fallback value
- Syntax → `var(--property, <fallback-value>)`

Fallback value

<code>

The `var()` function takes a second parameter as the fallback value.

```
/* Oooops! 🙄 */  
/* --color: salmon */  
  
p {  
  color: var(--color, black);  
}
```

calc()

<code>

The `calc()` function can be for calculations with custom properties.

```
:root {  
  --base: 16rem;  
}  
  
.class {  
  /* 80% of the --base variable */  
  padding: calc(.8 * var(--base));  
}
```

Custom properties and JavaScript

<code>

Custom properties can be accessed with JavaScript with low effort.

```
const el = document.querySelector(".foo");

// Get custom property from inline style
el.style.getPropertyValue("--color");

// Get custom property from computed styles
getComputedStyle(el).getPropertyValue("--color");

// Update custom property to inline style
el.style.setProperty("--color", "red");
```

Task

Custom Properties



Background

Why / What you'll learn



- Multiple backgrounds can create stunning effects
- Most websites rely on backgrounds
- Gradients are fun 🎨

Element background(s)

- An element can have a background color and multiple background images
- The backgrounds are stacked on top of each other
- Each of them can have different sizes and positions

Render order of multiple backgrounds

<code>

The background-color is rendered first followed by the background-images.

```
.background-order {  
  background-image:  
    url(foo.png),           /* Position 1 */  
    url(bar.png);         /* Position 2 */  
  background-color: salmon; /* Position 3 */  
}
```

background-color

- The `background-color` property sets the background color of an element
- Given value can be
 - Color name → i.e. `salmon` or `papayawhip`
 - Hexadecimal value → `#RGB`, `#RRGGBB`, `#RGBA`, `#RRGGBBAA`
 - RGB value → `rgb()` or `rgba()` function
 - HSL value → `hsl()` or `hsla()` function
 - `currentColor` or `transparent` keyword

background-clip

- The `background-clip` property defines which box of the box model is used to clip the background color and image
- Possible values
 - `border-box` (default)
 - `padding-box`
 - `content-box`
 - `text` (see [caniuse](#))

background-clip: text

<code>

Text can be given a fancy background (see [example](#)).

```
main {  
  font-size: 5rem;  
  font-weight: 800;  
  color: transparent;  
  background-image: linear-gradient(to right, salmon, black);  
  -webkit-background-clip: text;  
  background-clip: text;  
}
```

background-image

- The `background-image` property can set one or multiple images on an element

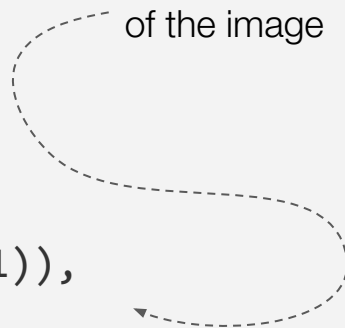
background-image

<code>

Multiple background images are stacked top to bottom, see [example](#).

```
.example {  
  width: 400px;  
  height: 400px;  
  background-color: salmon;  
  background-image:  
    linear-gradient(rgba(0,0,0,.1), rgba(0,0,0,1)),  
    url(https://picsum.photos/400/400);  
}
```

Gradient is rendered on top
of the image



background-repeat

- The property `background-repeat` defines how to repeat a background(s) (example [here](#))
- Possible values
 - `repeat` (default)
 - `repeat-x` (only x-axis)
 - `repeat-y` (only y-axis)
 - `no-repeat`
 - `space` (the background is repeated with spacing, but no clipping)
 - `round` (the background image will stretch without gaps)

background-size

- The `background-size` property defines the size of the background image(s)
- A size can be defined for each dimension (width and height)

background-size

- `background-size` can be used with these two keywords
 - `contain` → scale image as large as possible without clipping
 - `cover` → scale the image to cover the whole element
- Both keyword values will not stretch the image to keep its proportions
- See [example](#)

background-size

- `background-size` also takes
 - the `auto` keyword (default value) → keep the original size
 - percentage values → relative to the background position area
 - length values
- The first given value refers to the width and the second value refers to the height

background-size

<code>

One value and two value syntax for `background-size`.

```
/* background width is 100px and background height is auto */  
background-size: 100px;
```

```
/* background width 100px and background height 200px */  
background-size: 100px 200px;
```

background-position

- `background-position` is a shorthand property for the `background-position-x` and `background-position-y` properties
- The default values is `0% 0%` (top left corner)

background-position

- Possible values are
 - Keyword values → `top`, `bottom`, `left`, `right`, `center`
 - Length values → i.e. `10px` or `20rem`
 - Percentage values → refers to the size of the background area

linear-gradient

- The `linear-gradient()` function will draw a gradient with for the given colors
- By the default the gradient is drawn from top to bottom (0deg)

Gradient direction

- The direction of the gradient can be defined with
 - Degrees → i.e. `0deg` (top to bottom) or `90deg` (left to right)
 - Turns → i.e. `.25turn` equals `90deg`
 - Keywords → i.e. `to right`, `to bottom` or `to left top`

linear-gradient direction

<code>

Examples for the direction of the linear-gradient.

```
/* default direction 0deg - from top to bottom */  
background-image: linear-gradient(salmon, hotpink);
```

```
/* 180deg = from bottom to top */  
background-image: linear-gradient(180deg, salmon, hotpink);
```

```
/* to right bottom equals 135deg */  
background-image: linear-gradient(to right bottom, salmon, hotpink);
```

linear-gradient color stops

- Color stops defined where a color inside a gradients starts (or ends)
- Color stops can be defined with
 - Percentage values → refer to the background size
 - Length values → i.e. **10px** or **2rem**

linear-gradient

<code>

When colors are given without color stops they will be distributed equally.

```
/* Both gradients look the same */
```

```
background-image: linear-gradient(salmon, dodgerblue, gold);
```

```
background-image: linear-gradient(salmon 0%, dodgerblue 50%, gold 100%);
```

linear-gradient color stops

<code>

Gradient colors can also have no gradient when the color stops use the same value ([example](#))

```
.gradient {  
  width: 100px;  
  height: 100px;  
  background-image:  
    linear-gradient(45deg, transparent 48%, black 48%, black 52%, transparent 52%),  
    linear-gradient(135deg, transparent 48%, black 48%, black 52%, transparent 52%);  
}
```



How will this look like? 🙋

Combining multiple backgrounds

<code>

If only one value for background-size, -repeat or -position is given, the values apply to all background-images.

```
div {  
  background-image:  
    linear-gradient(black, white),  
    linear-gradient(lime, dodgerblue);  
  background-size: 100px 100px;  
  background-repeat: no-repeat;  
  background-position: 0px 0px;  
}
```



The values will be applied to all background images.

More resources

- Of course there is a [tool](#) for gradients
- [Gallery](#) of gradients by Lea Verou
- And there is also [radial-gradient\(\)](#) and [conic-gradient\(\)](#)

Containing Block

The containing block impacts layout specific properties.

Width of the containing block

- The width of the containing block is used as reference, when using the following properties with percentage values
 - `width`
 - `left` and `right`
 - `padding`
 - `margin`

Height of the containing block

- The height of the containing block is used as reference, when the following properties are used with percentage values
 - `height`
 - `top` and `bottom`

Identifying the containing block

- The containing block for elements with `position static`, `relative` or `sticky` is the content box of the nearest block level ancestor element

Identifying the containing block

- The containing block for absolute positioned elements is *the nearest positioned parent element*
- While this above statement is usually true, [here](#) are some exceptions

Identifying the containing block

- The containing block for elements with `position: fixed` is the viewport

height: 100%

<code>

Will the child element have the same height as the container?

```
.container {}  
  
.child {  
  height: 100%;  
}
```

```
<div class="container">  
  <p>Foobar</p>  
  <div class="child"></div>  
</div>
```


height: 100%

- Percentage values are calculated based on the height of the containing block
- If the height of the containing block **is not** defined explicitly and the element itself is not positioned absolute, the height is set to auto

Overflow

Overflow



**CSS
IS
AWESOME**

Overflow

- If the content of an element is bigger than the content box of the element it will overflow
- Content will only overflow, if the element has an explicit width or height

Overflow

- The `overflow` property is a shorthand for `overflow-x` and `overflow-y`

Overflow

- Possible values are
 - **visible** (default) → the overflowing content is visible
 - **hidden** → the overflowing content is hidden
 - **scroll** → overflowing content can be scrolled, scrollbars are always visible
 - **auto** → overflowing content can be scrolled, scrollbars are only visible when content overflows

Overflow

- Overflowing content will not affect the flow of the page and the following elements

CSS Multi-columns

CSS Multi-columns allows to create
print-inspired layout with minimal markup and
CSS.

CSS Column

- CSS Columns makes it easy to have print-inspired layouts
- It allows to define the width and amount of columns for text content

CSS Columns

- There are three different ways to declare columns
 - `column-count` → number of columns
 - `column-width` → width of columns
 - `column` → property shorthand (which declares both, safest way)

column-count


- The `column-count` property defines the number of columns
- When a number value is used, it defines the number of columns
- When the `auto` keyword is used, the number of columns depend on the `column-width` property

column-count

<code>

Brief example of column-count property.

```
p {  
  column-count: 5;  
}
```



⚠ Paragraphs will always have 5 column, even when it does not make any sense.

column-width

- The `column-width` property defines the *ideal* column width
- When a length value is passed, it defines the ideal width of columns
 - i.e. `column-width: 300px`
- When the `auto` keyword is used, the number of columns depend on the `column-count` property

ideal column-width

- The browser will always try to fill the width of the container
- The actual column width can differ from the passed value
 - When the container can't fit all columns, the columns will shrink
 - If there is available space, the columns will grow

column-width

<code>

Brief example of column-width.

```
p {  
  column-width: 200px;  
}
```

👉 When column-width is used without column-count, the browser will always try to fill the width of the container with unlimited columns

columns

<code>

The `columns` property shorthand will set both column-width and column-count.

```
p {  
  columns: 200px 3;  
  
  /*  
    column-width: 200px;  
    column-count: 3;  
  */  
}
```

column-gap

- The `column-gap` property defines the gutter between the columns
- By default a `1em` gutter is used
- Can be set to a length or percentage value

column-rule

- The `column-rule` property is used to add vertical lines between each column
- `column-rule` a property shorthand for
 - `column-rule-width`
 - `column-rule-style`
 - `column-rule-border`

column-span

- The column-span property allows to span elements across all columns
- This is often used for headlines
- Possible values are
 - none → the element does not span multiple columns
 - all → the element will span across all columns

column-span

<code>

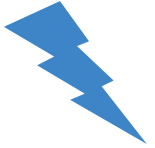
Minimal example for column-span, see demo [here](#).

```
article {  
  columns: 250px auto;  
}  
  
.article-headline {  
  column-span: all;  
}
```

Stacking Context and z-index

It's important to understand that the z-index
is **not global** to the document.

Why / What you'll learn



- The stacking context and z-index is closely coupled - you won't understand one without the other.
- Adding a high z-index to an element will not ensure that it will always be rendered on top of the document.

Works on my machine!



z-index:
100000000

Real World CSS

O RLY?

Captain Anonymous

By default all elements will be rendered by the order of their appearance in the HTML.

Stacking order (without z-index)

Elements are stacked (rendered) in the following order

1. Background and borders of (root) element
2. Non-positioned descendent elements in order of appearance
3. Positioned descendent elements in order of appearance

Stacking order (without z-index)

<code>

```
div:nth-child(1) {  
  position: relative;  
  top: 20px;  
  left: 20px;  
  background-color: dodgerblue;  
}
```

```
div:nth-child(2) {  
  /* position: relative; */  
  background-color: darksalmon;  
}
```

```
<main>  
  <div>First div</div>  
  <div>Second div</div>  
</main>
```

What happens if we toggle uncomment this line?

Stacking order and transform

- If an element is transformed (i.e. translated over a sibling element) it behaves like a positioned element

z-index

- The `z-index` property sets the position of an element along the z-axis
- Only works on *positioned* elements (or flex items or grid items)
- Possible values
 - `auto` keyword → default position on z-axis (0)
 - `number` → position on the z-axis

z-index

- Elements with the **same** z-index will be stacked according to the source order

z-index

- The `z-index` of an element is **NOT** global in the document
- The `z-index` is scoped within a stacking context

The **stacking context** is a three-dimensional conceptualization of HTML elements along 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.

- [MDN web docs](#)

A **stacking context** is an element that acts as a **z-index boundary** for its descendant elements.

Stacking context

- A stacking context is formed due to a couple of reasons, i.e.
 - The root element → `html`
 - Elements with position absolute / relative and a z-index value other than auto
 - Elements with position value fixed / sticky
 - Elements with opacity less than 1
 - Elements that are flex / grid items with a z-index value other than auto
- [Complete list](#) of reasons a stacking context is created

Stacking order with z-index

1. Background and border
2. Negative z-indexes
3. Block-level elements
4. Floated elements
5. Inline elements
6. z-index 0
7. Positive z-indexes

My (golden) Rules

Totally opinionated 🙋

My golden rules

- Global reset or normalize via tag selectors → low specificity
- Application styling only with classes → higher specificity
- Use pseudo classes and elements where it makes sense
- Use lower dash casing for all class names (even if you don't use BEM)

My golden rules

- Use BEM wherever it makes sense
- BEM should not reflect the HTML structure
- Modifiers should only add styling but not overwrite (no rule without exception)
- All rules should be lint-able



We teach.

workshops.de