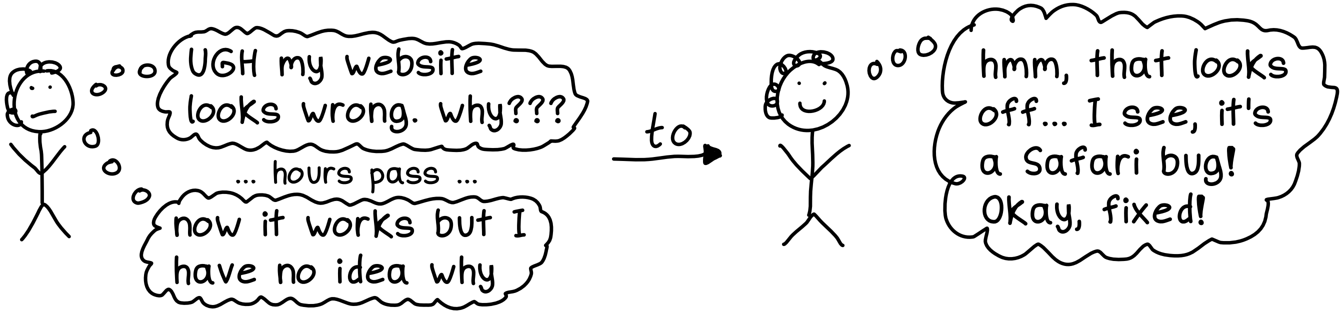
HELL YES! CSS!

by Julia Evans

what's this?

I wrote this zine because, after 15 years of being confused about CSS, I realized I was still missing a lot of basic CSS knowledge. Learning the facts in this zine helped me go from:


UGH my website looks wrong. why???
... hours pass ...
now it works but I have no idea why
to
hmm, that looks off... I see, it's a Safari bug! Okay, fixed!

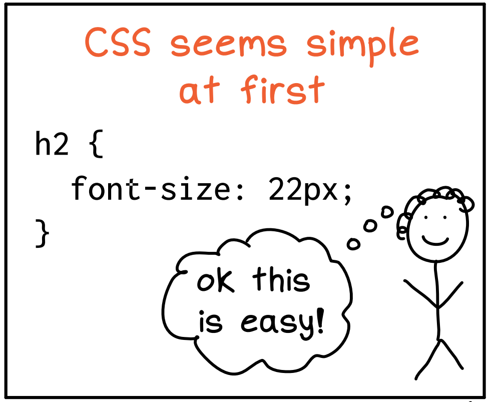
This zine also comes with examples for you to try out. They're at:

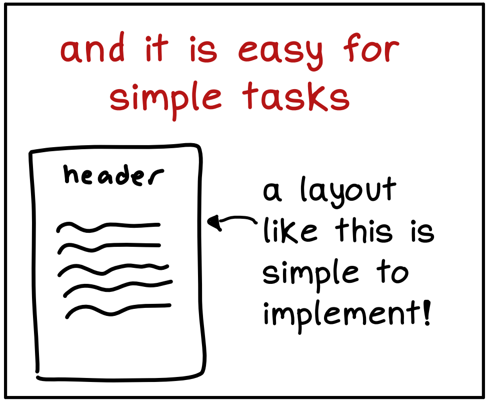
https://css-examples.wizardzines.com

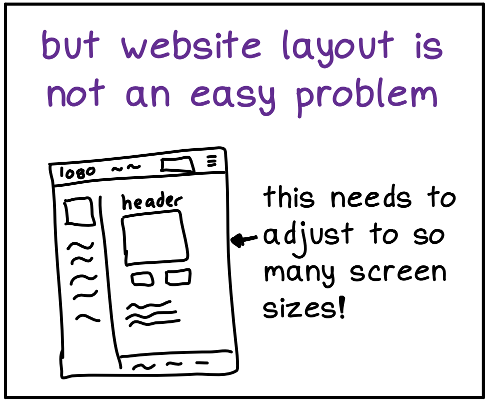
Panels which have examples you can try are labelled TRY ME!

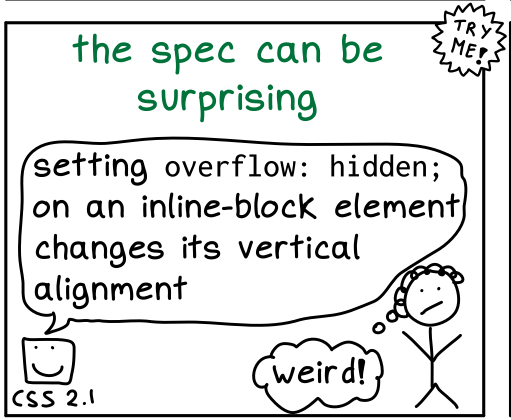
table of contents

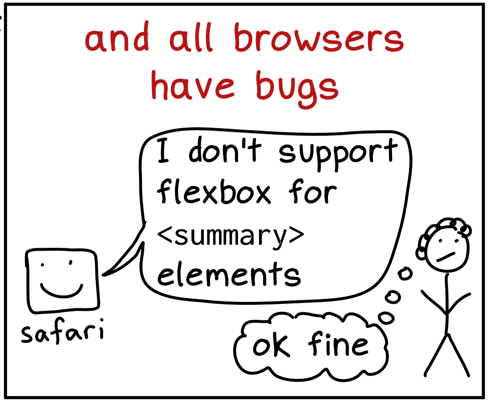
CSS isn't easy

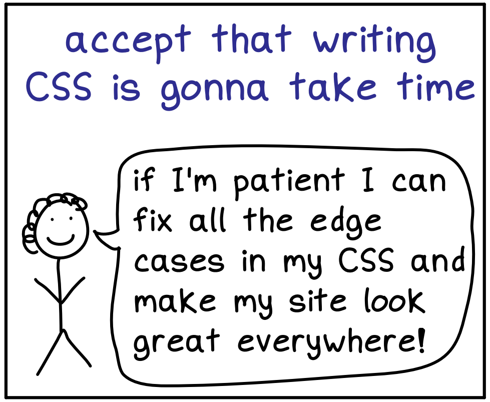



and it is easy for simple tasks
header
a layout like this is simple to implement!


but website layout is not an easy problem
logo
header
this needs to adjust to so many screen sizes!

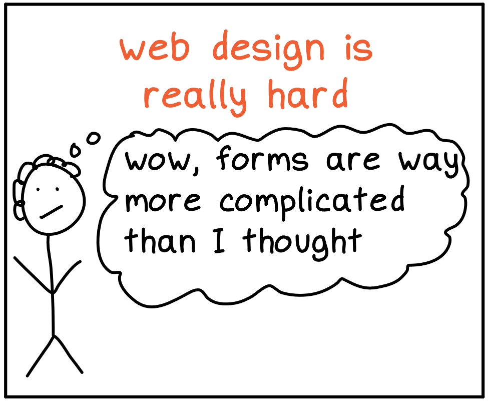

the spec can be surprising
TRY ME!
setting overflow: hidden; on an inline-block element changes its vertical alignment
CSS 2.1
weird!


and all browsers have bugs
I don't support flexbox for <summary> elements
safari
ok fine

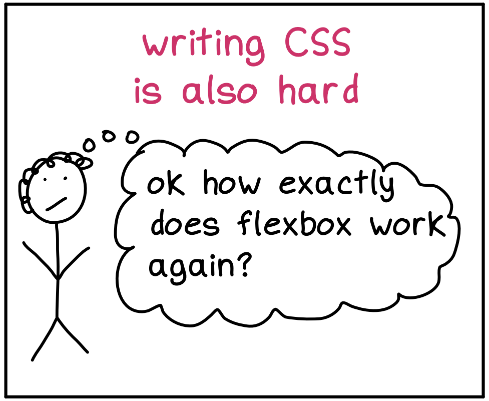

accept that writing CSS is gonna take time
if I'm patient I can fix all the edge cases in my CSS and make my site look great everywhere!

CSS != design

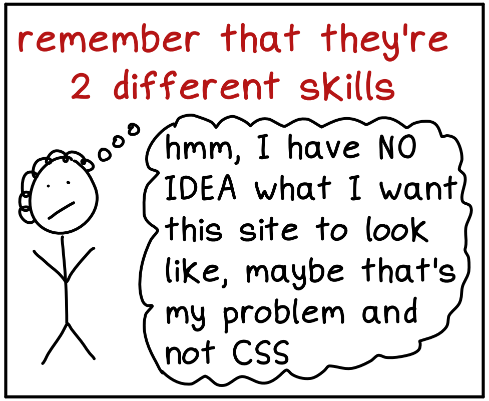
web design is really hard



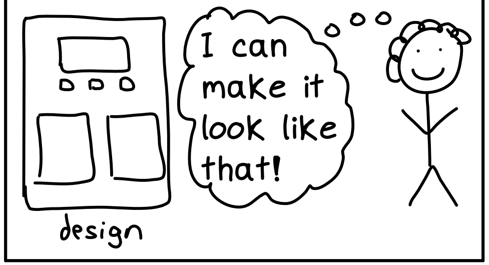
writing CSS is also hard



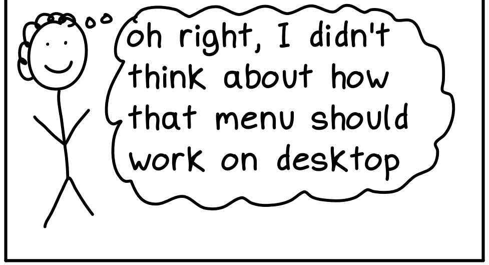
remember that they're 2 different skills


hmm, I have NO IDEA what I want this site to look like, maybe that's my problem and not CSS

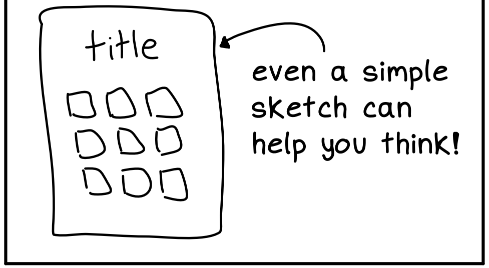
CSS is easier when you have a good design



usually you have to adjust the design


oh right, I didn't think about how that menu should work on desktop

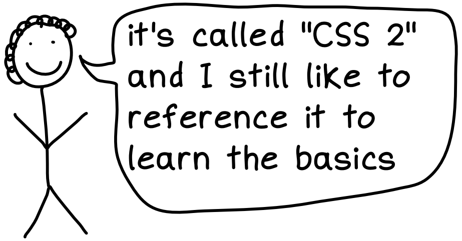
sketching a design in advance can help!



CSS specifications

CSS has specifications

there used to be just one specification


it's called "CSS 2" and I still like to reference it to learn the basics

today, every CSS feature has its own specification

you can find them all at https://www.w3.org/TR/CSS/

there are dozens of specs, for example: colors, flexbox, and transforms

major browsers usually obey the spec

but sometimes they have bugs

levels

CSS versions are called "levels".

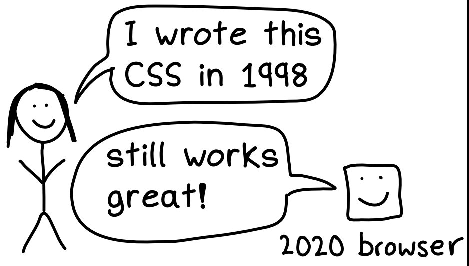
new levels only add new features. They don't change the behaviour of existing CSS code

new features take time to implement

https://caniuse.com can tell you which browser versions support a CSS feature

backwards compatibility

browsers support old HTML + CSS forever



this makes CSS hard to write...

why are CSS units so weird?
grey hair
let me tell you a story from 20 years ago...

... but it means it's worth the investment

I spent DAYS getting this CSS to work
I'll make sure it keeps working forever!
browser

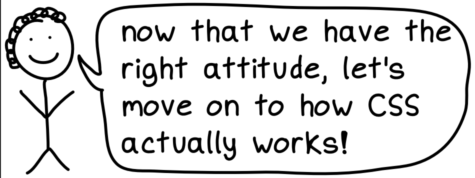
your CSS <u>doesn't</u> have to support browsers from 1998

just test that your CSS works on the browsers that your users are using!

newer features are often easier to use

what people expect from a website has changed a LOT since 1998. Newer CSS features make responsive design easy

a few CSS selectors


now that we have the right attitude, let's move on to how CSS actually works!

div

matches div elements

`<div>`

#welcome

matches elements by id

`<div id="welcome">`

.btn

. matches elements by class

`<a class="btn">`

div .btn

match every .btn element that's a descendent of a div

div.btn

match divs with class "btn"

`<div class="btn">`

div > .btn

match every .btn element that's a direct child of a div

.btn, #welcome

matches both .btn and #welcome elements

[href^="http"]

match a elements with a href attribute starting with http

:checked

matches if a checkbox or radio button is checked

a:hover

matches a elements that the cursor is hovering over

tr:nth-child(odd)

match alternating tr elements (make a striped table!)

specificity

different rules can set the same property

```
a:visited {
  color: purple;
  font-size: 1.2em;
}
#start-link {
  color: orange;
}
```

which one gets chosen?

CSS uses the "most specific" selector that matches an element

In our example, the browser will use `color: orange` because IDs (like `#start-link`) are more specific than pseudoclasses (like `:visited`)

CSS can mix properties from different rules

TRY ME!

```
a:visited {
  color: purple;
  font-size: 1.2em;
}
#start-link {
  color: orange;
}
```

it'll use this font-size

but use this color because #start-link is more specific

how CSS picks the "most specific" rule

a selector with element names

```
body div span a {
    color: red;
}
```

loses to

a selector with .classes or :pseudoclasses

```
.sidebar .link {
    color: orange;
}
```

loses to

a selector with an #id

```
#header a {
    color: purple;
}
```

loses to

an inline style

```
style="color: green;"
```

loses to

an !important rule*

```
color: blue !important;
```

*!important is very hard to override, which makes life hard for your future self!

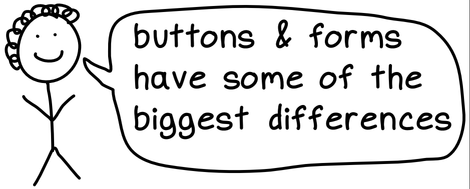
default stylesheets

every browser has a default stylesheet
(aka "user agent stylesheet")

a small sample from the Firefox default stylesheet:

```
h1 {
  font-size: 2em;
  font-weight: bold;
}
```

different browsers have different defaults



you can read the default stylesheet

Firefox's default stylesheets are at:

`resource://gre-resources/`

every property also has a default "initial value"

the initial value (defined in the spec) is what's used if no stylesheet has set anything. For example, `background-color`'s initial value is `transparent`

a CSS property can be set in 5 ways

lowest priority

① the initial value

② the browser's default stylesheet

③ the website's stylesheets and user stylesheets

④ inline styles set with HTML/JS

highest priority

units

CSS has 2 kinds of units: absolute & relative

absolute:	px, pt, pc, in, cm, mm
relative:	em, rem, vw, vh, %

rem

the root element's font size

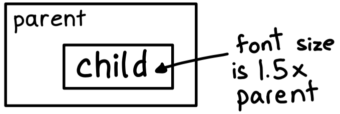
1rem is the same everywhere in the document. rem is a good unit for setting font sizes!

em

TRY ME!

the parent element's font size

```
.child {
    font-size: 1.5em;
}
```



0 is the same in all units

```
.btn {
    margin: 0;
}
```

also, 0 is different from none. border: 0 sets the border width and border: none sets the style

1 inch = 96 px

on a screen, 1 CSS "inch" isn't really an inch, and 1 CSS "pixel" isn't really a screen pixel. look up "device pixel ratio" for more.

rem & em help with accessibility

```
.modal {
    width: 20rem;
}
```

this scales nicely if the user increases their browser's default font size

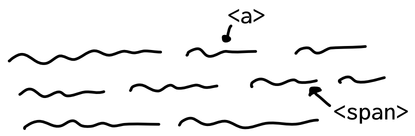
inline vs block

HTML elements default to inline or block

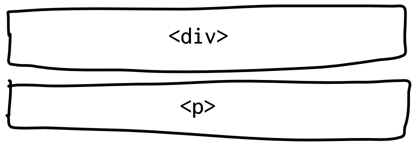
example inline elements:
`<a>` `<span>` `<strong>` `<i>` `<small>` `<abbr>` `<img>` `<q>` `<code>`

example block elements:
`<p>` `<div>` `<ol>` `<ul>` `<li>` `<h1>` - `<h6>` `<blockquote>` `<pre>`

inline elements are laid out horizontally



block elements are laid out vertically by default



to get a different layout, use `display: flex` or `display: grid`

inline elements ignore width & height*

Setting the width is impossible, but in some situations, you can use `line-height` to change the height

*`img` is an exception to this: look up "replaced elements" for more

display can force an element to be inline or block

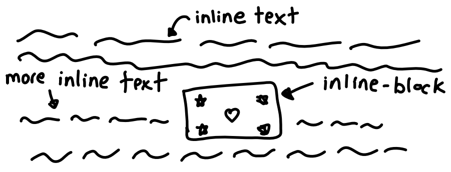
`display` determines 2 things:

1. whether the element itself is `inline`, `block`, `inline-block`, etc
2. how child elements are laid out (`grid`, `flex`, `table`, default, etc)

`display: inline-block;`

TRY ME!

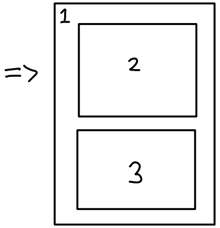
`inline-block` makes a block element be laid out horizontally like an inline element



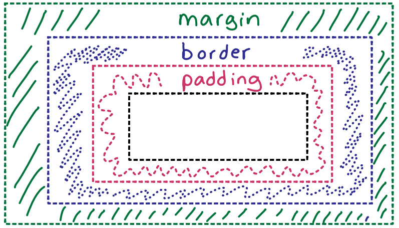
the box model

every HTML element is in a box

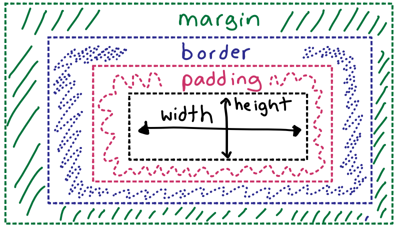
```
<div class="1">
  <div class="2" />
  <div class="3" />
</div>
```



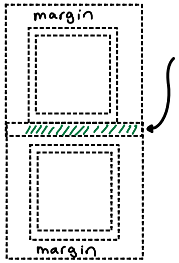
boxes have padding, borders, and a margin



width & height don't include any of those



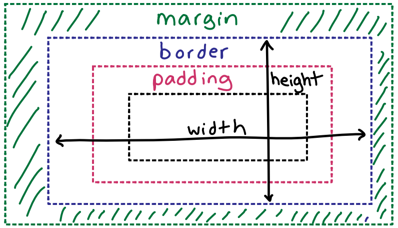
margins are allowed to overlap sometimes



the browser combines these top/bottom margins

look up "margin collapse" to learn more

box-sizing: border-box; includes border + padding in the width/height



padding & border are inside the element, margin is outside

For example, clicking on an element's border/padding triggers its onclick event, but clicking on the margin doesn't.

padding + margin syntax

there are 4 ways to set padding

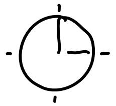
```
                        all sides
padding: 1em;
          vertical   horizontal
padding: 1em 2em;
          top    horizontal  bottom
padding: 1em 2em 3em;
          top   right  bottom  left
padding: 1em 2em 3em 4em;
```

tricks to remember the order

① **tr**ou**bl**e

```
 trouble
 ↑↑   ↑↑
top right bottom left
```

② it's clockwise



you can also set padding on just 1 side

```
padding-top: 1em;
padding-right: 10px;
padding-bottom: 3em;
padding-left: 4em;
```

differences between padding & margin

TRY ME!

→ padding is "inside" an element: the background color covers the padding, you can click padding to click an element, etc. Margin is "outside".

→ you can center with `margin: auto`, but not with padding

→ margins can be negative, padding can't

margin syntax is the same as padding

`border-width` also uses the same order: top, right, bottom, left

borders

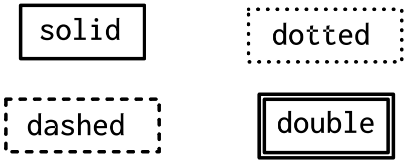
border has 3 components

```
border: 2px solid black;
```

is the same as

```
border-width: 2px;
border-style: solid;
border-color: black;
```

border-style options



+ lots more
(inset, groove, etc)

border-{side}

you can set each side's border separately:

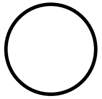
```
border-bottom:
        2px solid black;
```

border-radius

border-radius lets you have rounded corners

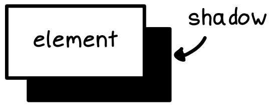
```
border-radius: 10px;
```

```
border-radius: 50%;
```
will make a square into a circle!



box-shadow

lets you add a shadow to any element

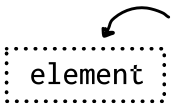


```
box-shadow: 5px 5px 8px black;
```
x offset · y offset · blur radius · color

outline

TRY ME!

outline is like border, but it doesn't change an element's size when you add it



outlines on :hover/ :active help with accessibility: with keyboard navigation, you need an outline to see what's focused

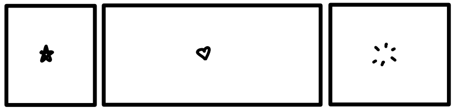
flexbox basics

`display: flex;`

set on a parent element to lay out its children with a flexbox layout.

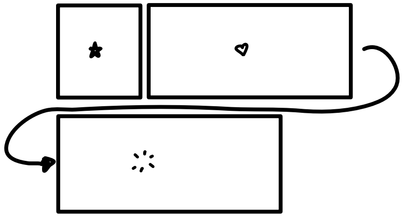
by default, it sets `flex-direction: row;`

`flex-direction: row;`



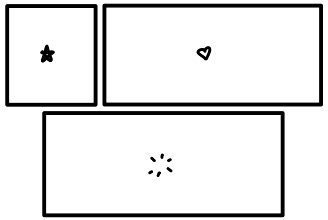
by default, children are laid out in a single row. the other option is `flex-direction: column`

`flex-wrap: wrap;`



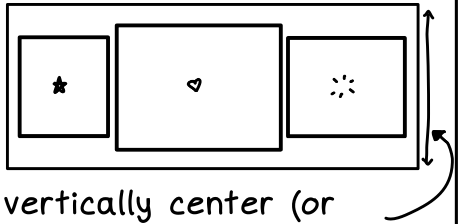
will wrap instead of shrinking everything to fit on one line

`justify-content: center;`



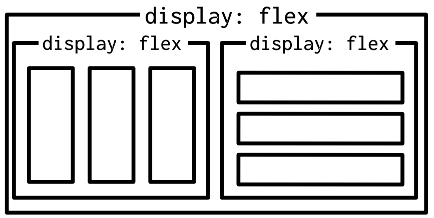
horizontally center (or vertically if you've set `flex-direction: column`)

`align-items: center;`

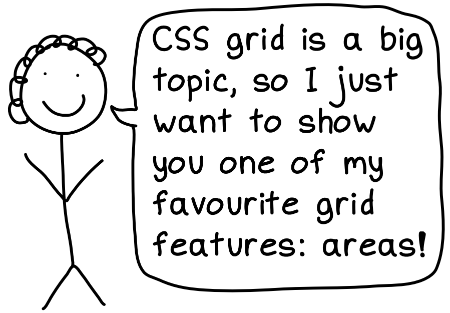


vertically center (or horizontally if you've set `flex-direction: column`)

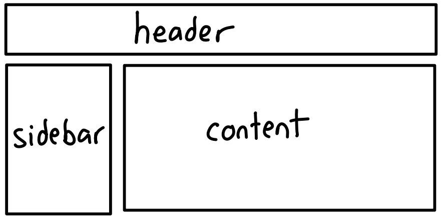
you can nest flexboxes



CSS grid: areas!


CSS grid is a big topic, so I just want to show you one of my favourite grid features: areas!

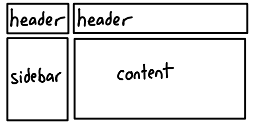
let's say you want to build a layout



grid-template-areas lets you define your layout in an almost visual way

```
grid-template-areas:
   "header header"
   "sidebar content";
```

I think of it like this:



1. write your HTML

```
<div class="grid">
  <div class="top"></div>
  <div class="side"></div>
  <div class="main"></div>
</div>
```

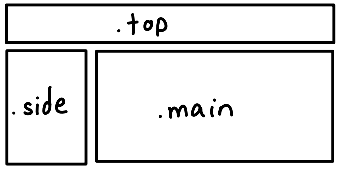
2. define the areas

```
.grid {
  display: grid;
  grid-template-columns:
      200px 800px;
  grid-template-areas:
    "header header"
    "sidebar content";
}
```

3. set grid-area

```
.top {grid-area: header}
.side {grid-area: sidebar}
.main {grid-area: content}
```

result:



centering

center text with text-align

```
h2 {
  text-align: center;
}
```

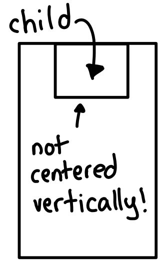
center block elements with margin: auto

example HTML:

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

margin: auto only centers horizontally

```
.child {
  width: 400px;
  margin: auto;
}
```



vertical centering is easy with flexbox or grid

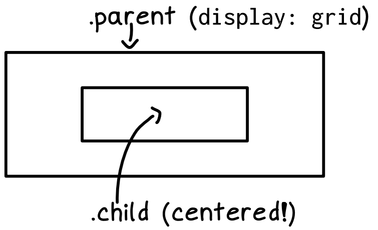
here's how with grid:

```
.parent {
  display: grid;
  place-items: center;
}
```

and with flexbox:

```
.parent {
  display: flex;
}
.child {
  margin: auto;
}
```

it's ok to use a flexbox or grid just to center one thing



position: absolute

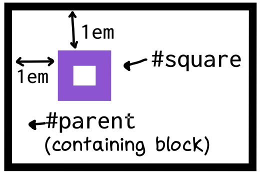
position: absolute;
doesn't mean absolutely positioned on the page: it's relative to the "containing block"

the "containing block" is the closest ancestor with a position that isn't set to static (the default value), or the body if there's no such ancestor.

Here's some typical CSS:

```
#square {
   position: absolute;
   top: 1em; left: 1em;
}
#parent {
   position: relative;
}
```

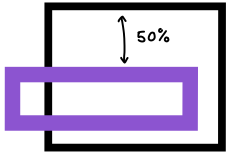
this makes #parent the containing block



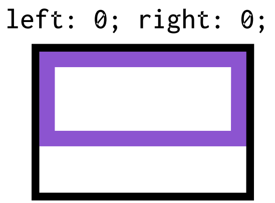
top, bottom, left, right will place an absolutely positioned element

```
top: 50%;
bottom: 2em;
right: 30px;
left: -3em;
```

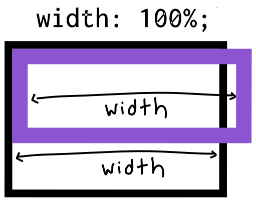
negative works too



left: 0; right: 0; ≠ width: 100%;

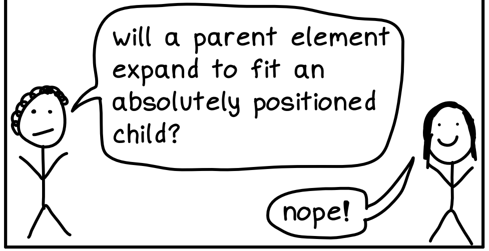


left and right borders are both 0px away from containing block



the box sticks out because width doesn't include borders by default

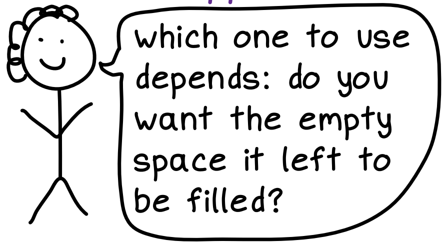
absolutely positioned elements are taken out of the normal flow



will a parent element expand to fit an absolutely positioned child?
nope!

hiding elements

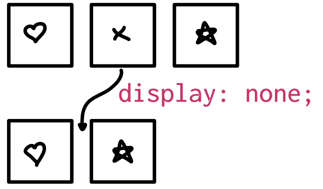
there are many ways to make an element disappear


which one to use depends: do you want the empty space it left to be filled?

display: none;

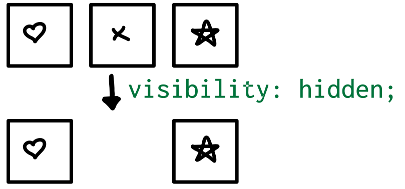
TRY ME!

other elements will move to fill the empty space



visibility: hidden;

the empty space will stay empty



opacity: 0;

like `visibility: hidden`, but you can still click on the element & it'll still be visible to screen readers. Usually `visibility: hidden` is better.

how to slowly fade out

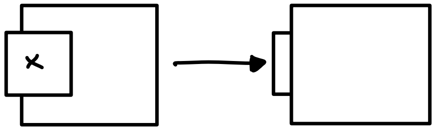
```
#fade:hover {
    transition: all 1s ease;
    visibility: hidden;
    opacity: 0;
}
```

set the opacity just so that the transition works

z-index

TRY ME!

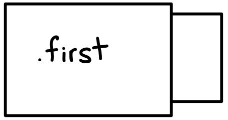
z-index sets the order of overlapping positioned elements



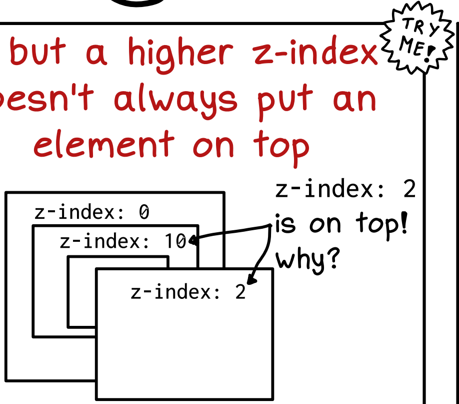
stacking contexts

a z-index can push an element up/down...

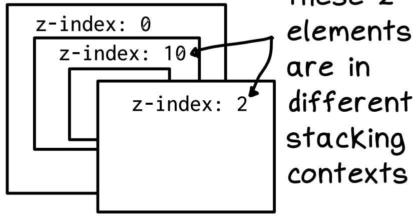
```
.first {
  z-index: 3;
}
.second {
  z-index: 0;
}
```



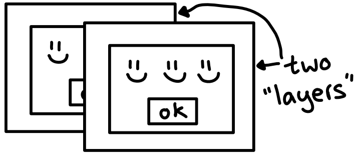
... but a higher z-index doesn't always put an element on top



every element is in a stacking context



a stacking context is like a Photoshop layer



by default, an element's children share its stacking context

setting z-index creates a stacking context

```
#modal {
    z-index: 5;
    position: absolute;
}
```

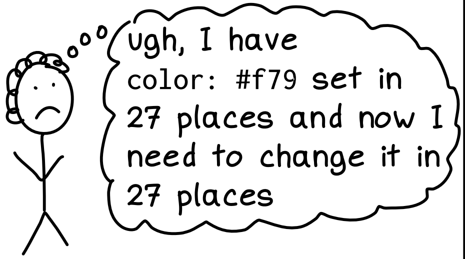
this is a common way to create a stacking context

stacking contexts are confusing

You can do a lot without understanding them at all. But if z-index ever isn't working the way you expect, that's the day to learn about stacking contexts :)

CSS variables

duplication is annoying


ugh, I have color: #f79 set in 27 places and now I need to change it in 27 places

define variables in any selector

```
body {
   --text-color: #f79;
}            applies to
             everything
#header {
   --text-color: #c50;
}          applies to children
           of #header
```

use variables with var()

```
body {
   color: var(--text-color);
}
```

variables always start with --

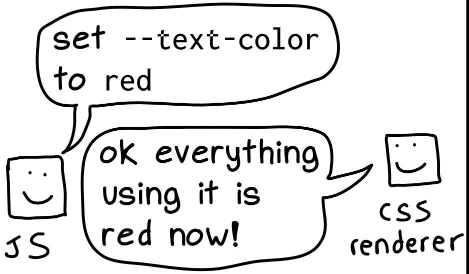
do math on them with calc()

```
#sidebar {
   width: calc(
       var(--my-var) + 1em
   );
}
```

you can change a variable's value in Javascript

```
let root =
  document.documentElement;
root.style.setProperty(
   '--text-color', 'black');
```

changes to variables apply immediately



transitions

an element's computed style can change

2 ways this can happen:

① pseudo-classes (like :hover)

② Javascript code
```
el.classList.add('x')
```

new styles change the element instantly...

```
a:hover {
  color: red;
}
```

the element will turn red right away

... unless you set the transition property

TRY ME!

```
a {
  color: blue;
  transition: all 2s;
}
a:hover {
  color: red;
}
```

will fade from blue to red over 2s

transition has 3 parts

```
transition: color 1s ease;
```

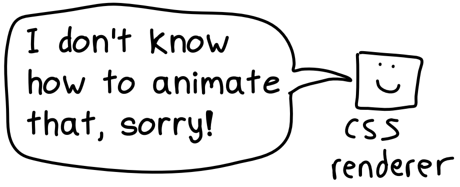
which CSS properties to animate

duration

timing function

not all property changes can be animated....

```
list-style-type: square;
```



...but there are dozens of properties that can

if it's a number or color, it can probably be animated!

```
font-size: 14px;
rotate: 90deg;
width: 20em;
```

media queries

media queries let you use different CSS in different situations

```
@media (print) {
  #footer {
    display: none;
  }
}
```

media query

CSS to apply

max-width & min-width

```
@media (max-width: 500px) {
   // CSS for small screens
}

@media (min-width: 950px) {
   // CSS for large screens
}
```

print and screen

screen is for computer/ mobile screens
print is used when printing a webpage

there are more: tv, tty, speech, braille, etc

accessibility queries

you can sometimes find out a user's preferences with media queries

examples:

```
prefers-reduced-motion: reduce
prefers-color-scheme: dark
```

you can combine media queries

it's very common to write something like this:

```
@media screen and
    (max-width: 1024px)
```

the viewport meta tag

```
<meta name="viewport"
content="width=device-width,
initial-scale=1">
```

Your site <u>will</u> look bad on mobile if you don't add a tag like this to the <head> in your HTML. Look it up to learn more!

the CSS inspector

all major browsers have a CSS inspector

usually you can get to it by right clicking on an element and then "inspect element, but sometimes there are extra steps

see overridden properties

```
button {
   display: inline-block;
   ~~color: var(--orange);~~
}
```

edit CSS properties

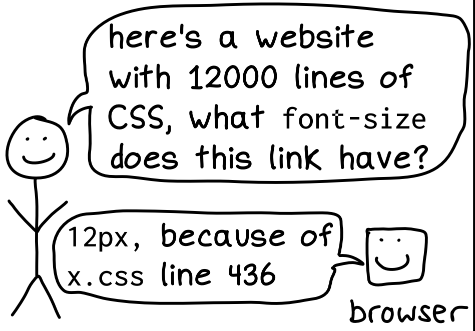
```
element {
}
```

lets you change this element's properties

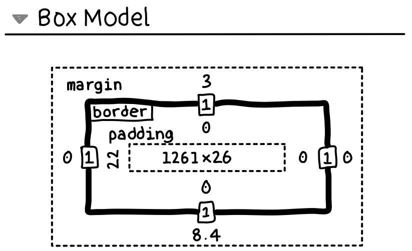
```
button {
   display: inline-block;
   border: 1px solid black;
}
```

this lets you change the border of every <button>!

see computed styles


here's a website with 12000 lines of CSS, what font-size does this link have?
12px, because of x.css line 436
browser

look at margin & padding



... and LOTS more

different browsers have different tools! For example, Firefox has special tools for debugging grid/flexbox

testing checklist

Finally, it's important to test your site with different browsers, screen sizes, and accessibility evaluation tools.

browsers

- ☐ Chrome
- ☐ Safari
- ☐ Firefox
- ☐ maybe others!

sizes

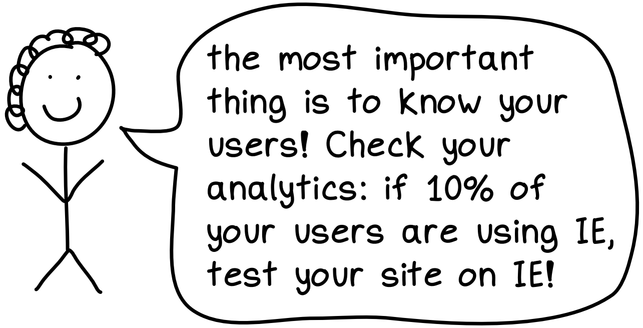
- ☐ small phone (300px wide)
- ☐ tablet (~700px)
- ☐ desktop (~1200px)

accessibility

- ☐ colour contrast
- ☐ text size
- ☐ keyboard navigation
- ☐ works with a screen reader

performance

- ☐ fake a slow/high latency network connection!


the most important thing is to know your users! Check your analytics: if 10% of your users are using IE, test your site on IE!

thanks for reading

CSS is a HUGE topic and there's a lot more to learn than what's in this zine. Here are some of my favourite CSS resources:

♡ **CSS Tricks** (css-tricks.com)

Hundreds of helpful blog posts and incredible guides, like their guides to centering & flexbox.

♡ **Mozilla Developer Network** (developer.mozilla.org)

My favourite reference for CSS, JS, HTML, and HTTP

♡ **Can I use...** (caniuse.com)

Tells you which browser versions (and what likely % of your users) have support for each CSS feature.

♡ **W3** (w3.org/TR/CSS)

The CSS specifications. Can be useful as a reference too!

credits

Cover art: Vladimir Kašiković
Editing: Dolly Lanuza, Kamal Marhubi
Technical review: Melody Starling
and thanks to all the beta readers ♡