

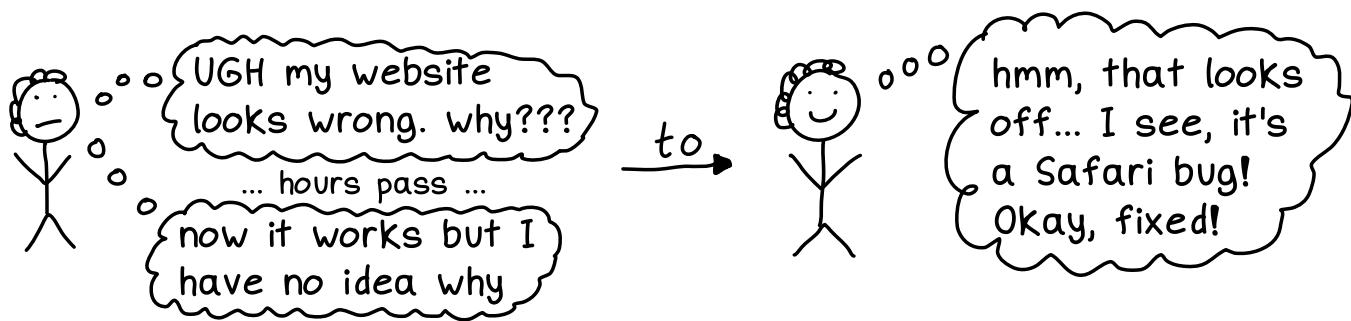
HELL YES! CSS!



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:



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



table of contents

ATTITUDE

CSS isn't easy.....	4
CSS isn't design	5
CSS specifications.....	6
backwards compatibility.....	7

BASICS

selectors.....	8
specificity	9
default stylesheets.....	10
units.....	11

LAYOUT

inline vs block.....	12
the box model.....	13
padding & margin....	14
borders.....	15
flexbox basics	16
CSS grid: areas!.....	17
centering	18
position: absolute.....	19

GETTING FANCY

hiding elements	20
stacking contexts.....	21
CSS variables.....	22
transitions.....	23

MAKING IT WORK

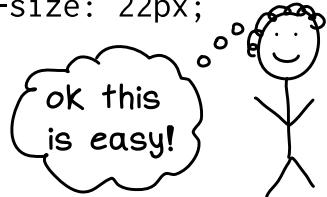
media queries.....	24
the CSS inspector.....	25
testing checklist.....	26

CSS isn't easy

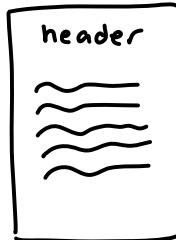
4

CSS seems simple at first

```
h2 {  
    font-size: 22px;  
}
```

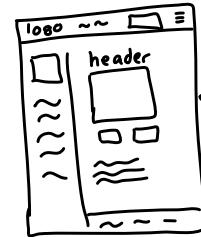


and it is easy for simple tasks



a layout like this is simple to implement!

but website layout is not an easy problem



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



weird!

and all browsers have bugs



I don't support flexbox for <summary> elements

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

5

web design is
really hard

wow, forms are way
more complicated
than I thought

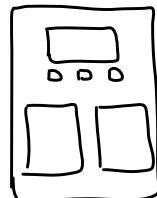
writing CSS
is also hard

ok how exactly
does flexbox work
again?

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



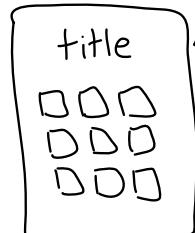
I can
make it
look like
that!

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!



even a simple
sketch can
help you think!

CSS specifications

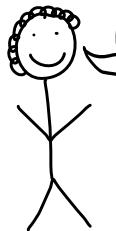
6

CSS has specifications



hello, this is how
max-width works in
excruciating detail

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 



browser

oops, I didn't quite implement that right...

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

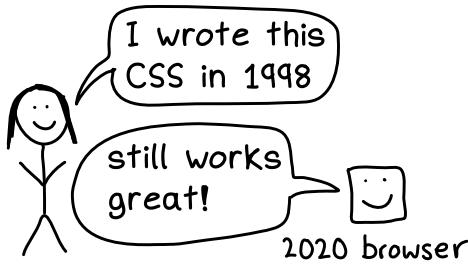


can tell you which browser versions support a CSS feature

backwards compatibility

7

browsers support old
HTML + CSS forever



this makes CSS hard
to write...



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



if you don't follow the
standards, you're not
guaranteed backwards
compatibility



your CSS doesn't
have to support
browsers from 1998



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

8



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">`

`.button`

`.` matches elements by class

``

`div .button`

match every `.button` element that's a descendent of a `div`

`div.button`

match `div`s with class "button"

`<div class="button">`

`div > .button`

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

`.button, #welcome`

matches both `.button` 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

9



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

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

it'll use this
font-size: 1.2em; 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

10

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



buttons & forms have some of the biggest differences

you can read the default stylesheet

Firefox's default stylesheets are at:

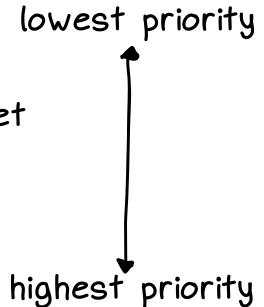
`resource://gre-resources/`

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

- ① the initial value
- ② the browser's default stylesheet
- ③ the website's stylesheets
- ④ a user stylesheet (least common)
- ⑤ inline styles set with HTML/JS



units

11

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!

TRY
ME!

em

the parent element's
font size

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

font size
is 1.5x
parent

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

12

HTML elements default to **inline** or **block**

example inline elements:

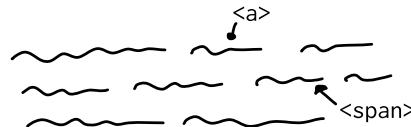
<a>
 <i>
<small> <abbr>
 <q>
<code>

example block elements:

<p> <div>

<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

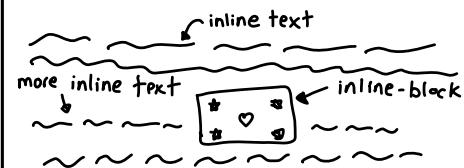
display can force an element to be **inline** or **block**

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;

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

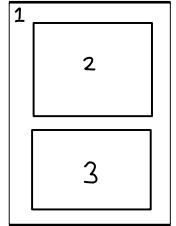


TRY
ME!

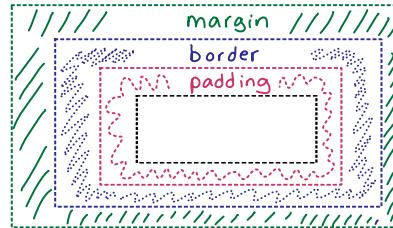
the box model

every HTML element
is in a box

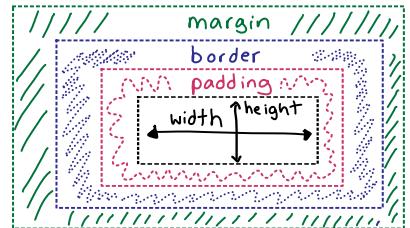
```
<div class="1">  
  <div class="2" />  
  <div class="3" />  
</div>
```



boxes have **padding**,
borders, and a **margin**

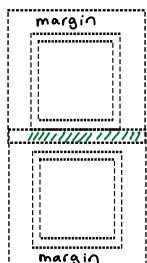


width & height don't
include any of those



margins are allowed
to overlap sometimes

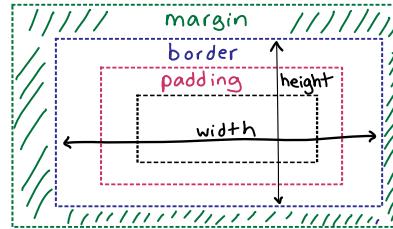
TRY
ME!



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

14

there are 4 ways
to set padding

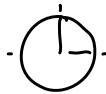
padding: 1em;
vertical horizontal
padding: 1em 2em;
top horizontal bottom
padding: 1em 2em 3em;
top right bottom left
padding: 1em 2em 3em 4em;

tricks to remember
the order

① 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



→ 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

15

border has 3 components

`border: 2px solid black;`

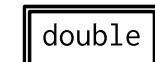
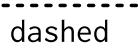
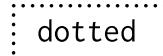
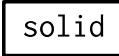
is the same as

`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:`

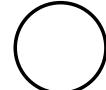
`2px solid black;`

border-radius

`border-radius` lets you have rounded corners

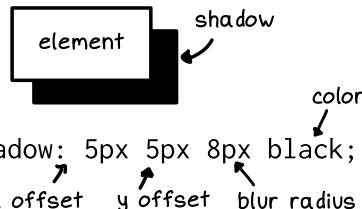
`border-radius: 10px;`

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



box-shadow

lets you add a shadow to any element



outline

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

TRY ME!

flexbox basics

16

`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;`



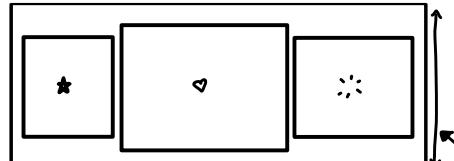
by default, children are
laid out in a single row.
the other option is
`flex-direction: column`

`justify-content: center;`



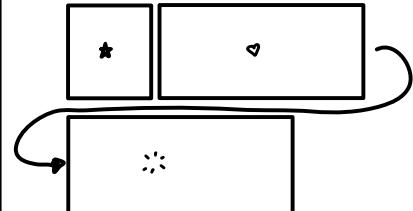
horizontally center
(or vertically if you've set
`flex-direction: column`)

`align-items: center;`



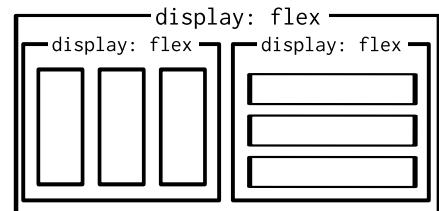
vertically center (or
horizontally if you've set
`flex-direction: column`)

`flex-wrap: wrap;`



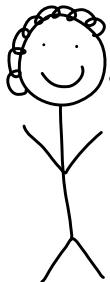
will wrap instead of shrinking
everything to fit on one line

`you can nest flexboxes`



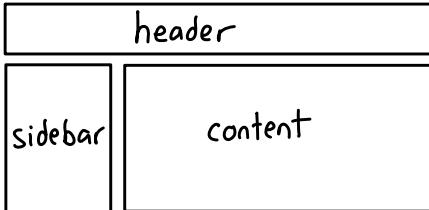
CSS grid: areas!

17



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

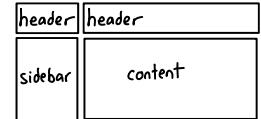
let's say you want to build a layout



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>
```

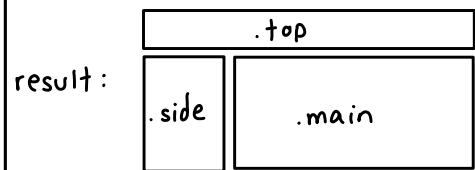
TRY
ME!

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}
```



centering

center text with text-align

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

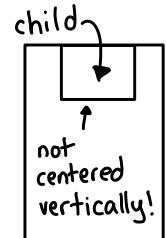
center block elements with margin: auto

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

```
.parent (display: grid)
```



.child (centered!)

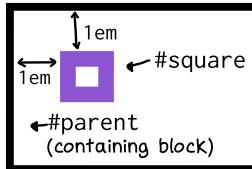
position: absolute

`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
```



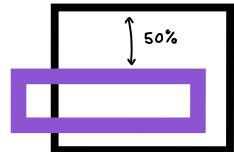
top, bottom, left, right
 will place an absolutely
 positioned element

```
top: 50%;  

bottom: 2em;  

right: 30px;  

left: -3em;
```



negative
 works too

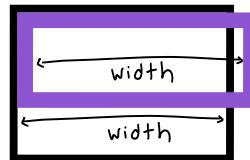
`left: 0; right: 0; ≠ width: 100%;`

`left: 0; right: 0;`



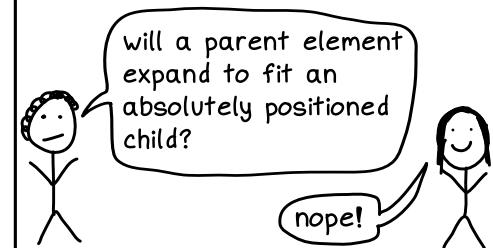
left and right borders
 are both 0px away
 from containing block

`width: 100%;`



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

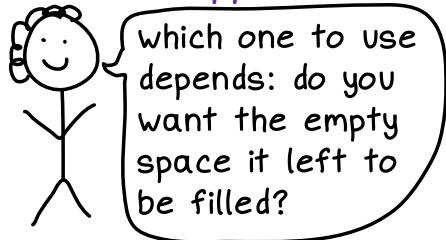
absolutely positioned
 elements are taken out
 of the normal flow



hiding elements

20

there are many ways
to make an element
disappear

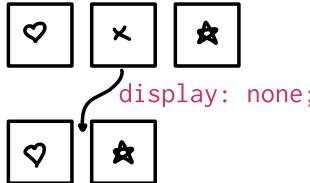


which one to use
depends: do you
want the empty
space it left to
be filled?

`display: none;`

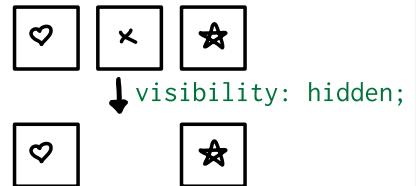


other elements will move
to fill the empty space



`visibility: hidden;`

the empty space will
stay empty



`opacity: 0;`

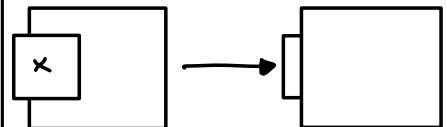
like `visibility: hidden`,
but you can still click on
the element & it'll still be
visible to screen readers.
Usually `visibility: hidden`
is better.

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`

`z-index` sets the order
of overlapping positioned
elements



stacking contexts

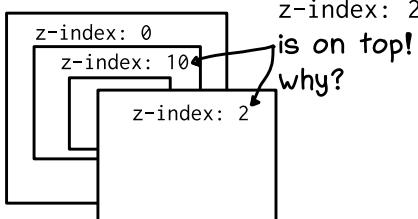
21

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

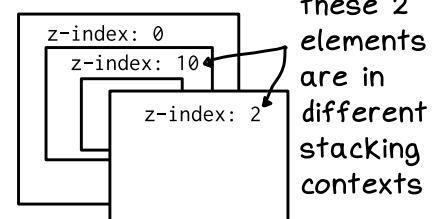
```
.first {  
  z-index: 3;  
}  
  
.second {  
  z-index: 0;  
}
```



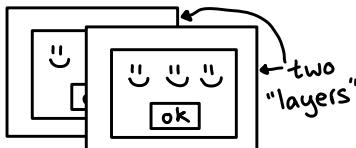
... but a higher z-index doesn't always put an element on top



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



define variables in any selector

```
body {  
    --text-color: #f79;  
}  
#header {  
    --text-color: #c50;  
}
```

applies to everything
applies to children of #header

use variables with var()

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

variables always start with --

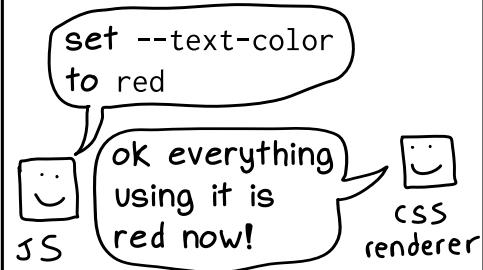
do math on them with calc()

```
#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

```
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;
```

I don't know how to animate that, sorry!



...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

24

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 will 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 {  
}
```

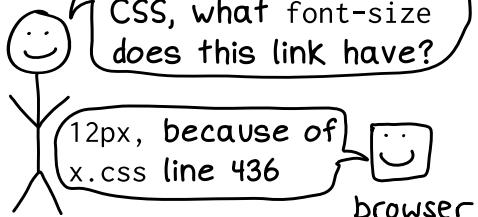
lets you change this element's properties

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

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

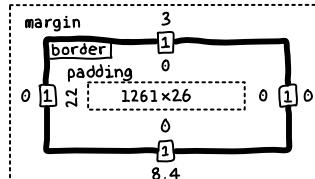
see computed styles

here's a website with 12000 lines of CSS, what font-size does this link have?



look at margin & padding

Box Model



... and LOTS more

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

testing checklist

26

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



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!

performance

- fake a slow/high latency network connection!

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.

♡ Can I use... (caniuse.com)

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

♡ Mozilla Developer Network

(developer.mozilla.org)

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

♡ 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 ♡

♡ this?
more at
★wizardzines.com★