CSS Overview

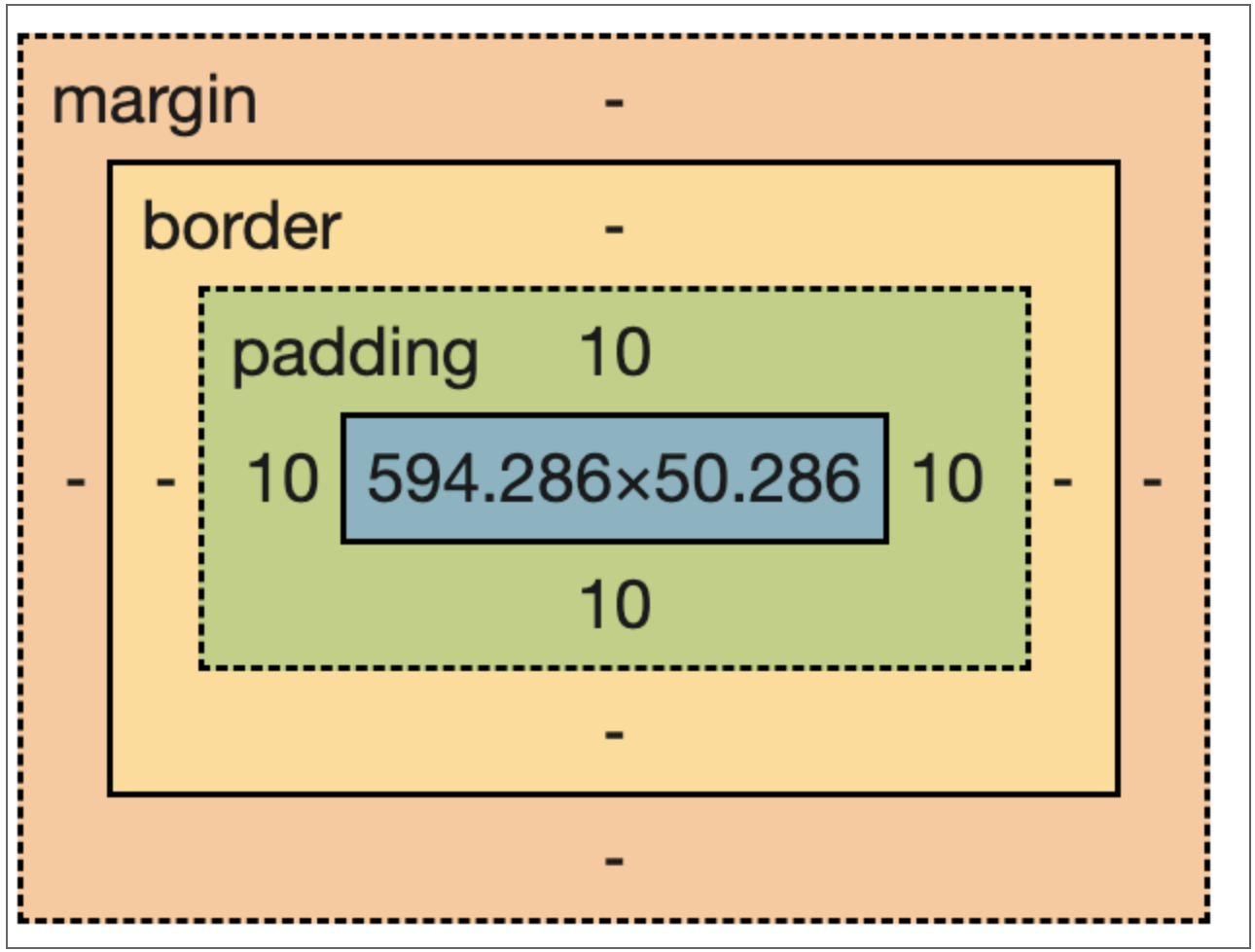
CSS provides

- Rules for appearance of HTML
- Based on structure

CSS Box Model

Every rendered element is a "**box**" of boxes:

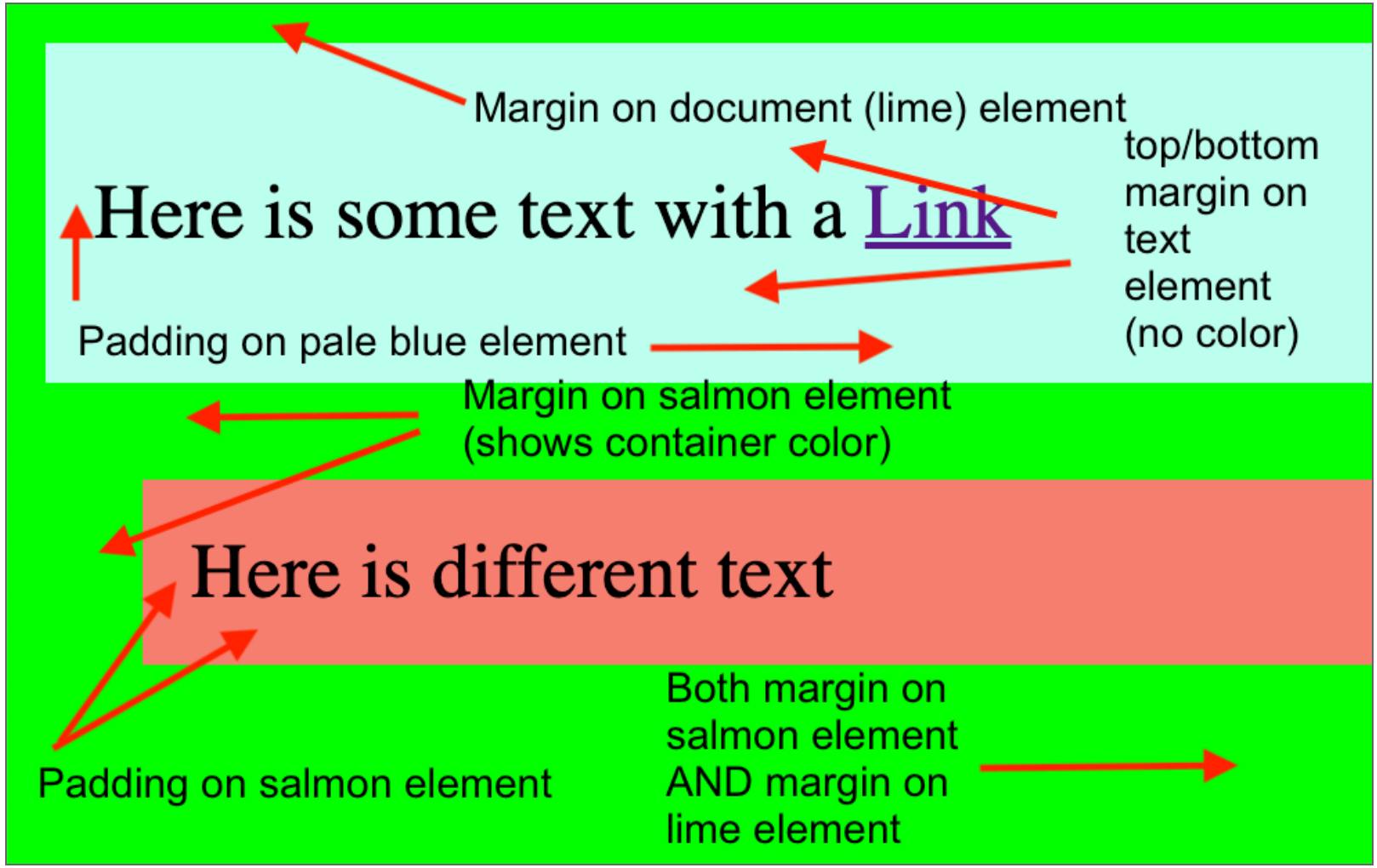
- content has **height** and **width**
- **padding** around it
- **border** has a width
- **margins** between border and adjacent boxes


margin
-
border
-
padding
10
-
-
10
594.286×50.286
10
-
-
10
-
-

Controlling Spacing

- Using the Box Model is an essential skill
    - **margin** - Space around box and other boxes
    - **border** - Frame of box
    - **padding** - Space INSIDE box, around content
    - **width/height** - size of content itself
        - Usually automatically determined
        - Do NOT overuse setting fixed sizes!
- "Visual Space" is likely most important UI control
    - Don't smash content together

Box Model in Use



Box Sizing

How wide is the below element?

```
p {
  width: 100px;
  padding: 10px;
}
```

- With `box-sizing: content-box;` (default) = 120px
- With `box-sizing: border-box;` = 100px;

Common to see:

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

Stylesheets

There are a few ways to apply CSS to HTML

- Inline CSS on element (**don't do**)
- `<style>` element (**don't do**)
- A stylesheet file linked via `<link>` element

Inline CSS

CSS can be applied to an element as an attribute

```
<div style="color: red;">Example</div>
```

Example

- Generally: Don't do this
- For this course: **DO NOT DO THIS**
    - Force to learn alternatives

Why not use Inline CSS?

- Hard to override
- Impossible to reuse
- Really annoying to edit
- Frustrating to debug
- Difficult to maintain

Using a style element

```
<head>
  <style>
    #demo {
      color: red;
    }
  </style>
</head>
<body>
  <div id="demo">Example</div>
</body>
```

Example

- Generally: Don't do this
- For this course: **do not do this**

Why not use style element?

- Makes for big files
- Impossible to reuse between files
- Annoying to edit

Using a stylesheet file

```
<link rel="stylesheet" href="example.css"/>
// in example.css

#demo {
  color: red;
}

.selected {
  color: black;
  background-color: red;
}
```

How many stylesheets?

Varies, but typical to have:

- 1 file for site-wide standards
- 1 file for page-specific css

Sites might have 1 stylesheet, might have 5

- All about levels of abstraction and reuse

Exceptions

Okay to use `<style>` element

- If tools build it for you
    - You don't suffer any of the downsides
    - Fewer requests

Okay to use inline CSS

- If assigned with JS *and*
- Values can't be defined by class names
    - Such as changing position by dragging

Remember: HTML is made up of nested elements

If element A has element B in element A's **content**...

- Element A is the **parent** of element B
- Element B is the **child** of element A

This relationship description applies to many levels

- A **descendant** element is a child, grand-child, etc.
- Elements with the same parent are **siblings**
- An **ancestor** element has descendants

CSS makes heavy use of these relationships!

- Understanding the relationships of HTML elements is essential
- CSS will often decide the appearance of an element based on relationships
    - Defined by **selectors**
- Many CSS properties assigned to an ancestor will apply to descendants (inheritance)

CSS Rules

CSS is made up of **rules**

- A rule is **selector(s)** and **declarations**

```
p {
  color: #C0FFEE;
}

li {
  border: 1px solid black;
  padding: 0px;
}
```

Invalid rules/declarations are skipped

- Next rule/declaration tried
- No error message!

Selectors

A rule has one or more comma separated **selectors**
https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors

```
p, li {
  background-color: #BADA55;
}
```

- Tag name: `p {...}`
- "id" `#demo {...}`
- A class `.example {...}` (**most common**)
- Descendants `div .wrong {...}`
- Direct children `div > .wrong {...}`
- Many other options (read on MDN)

Quick Note: Classes are most common selector

- We will discuss why later, but take note now
- Default to using class selectors
    - Unless you have a reason why
    - This assignment WON'T let you use classes
        - To force you to learn relationships
    - But in future, use class names for selectors

Declarations

The "body" of a CSS rule is declarations.

```
{
  css-property: value;
  another-property: value;
}
```

If a property doesn't exist, the next will be tried

Browsers have specific properties with "prefixes"

- Example: `--webkit-transform-style: flat;`
- Generally should avoid these in modern CSS
    - A few historical ones still exist

Shorthand properties

Some properties accept multiple values to apply to multiple properties:

```
p {
  border: 1px solid black;
}

p {
  border-width: 1px;
  border-style: solid;
  border-color: black;
}
```

Use these where the meaning is understood

Nothing wrong with being more explicit for clarity

CSS colors

- A named color **https://drafts.csswg.org/css-color/#named-colors**
- a hexadecimal RGB color (e.g. `#BADA55`)
    - 3, 4, 6, and 8 character varieties
    - 3 or 4 have hex chars doubled
        - e.g. `#639` is `#663399`
    - 4 or 8 include alpha aka opacity
- `rgb()` or `rgba()` passing 3 RGB vals and an alpha
    - passed RGB values are decimal
    - alpha is 0-1 or 0%-100%
- non-RGB systems like `hsl()` or `hwb()`

Property Inheritance

Some properties are inherited by descendants

- Unless overridden
- Some other properties are not inherited
- Ex: "color" is inherited
- Ex: "width" is not inherited
- Most colors and typography are inherited
- Sizes and positioning are not

Casing is used to communicate

- Previously said **indentation** is used for humans
- So too is **casing**
    - When uppercase/lowercase letters
    - How multiple words are separated

A very common mistake

- New coders often treat as unimportant
- Your future team will reject your work
- Your future self will hate you

Different Casing Conventions (Part 1)

- `CONSTANT_CASE`
    - All uppercase
    - Words separated with `_`
    - Used to indicate "constants" in JS/Java/Python/etc
- `snake_case`
    - All lowercase
    - Words separated with `_`
    - Used in Python
    - NOT used in this course

Different Casing Conventions (Part 2)

- `MixedCase` / `PascalCase`
    - First letter of words capitalized
    - Words squished together/no separation
    - Used in some traditional coding languages
    - Used for components in Javascript (JS)
        - Also JS classes, distinct from CSS classes

Different Casing Conventions (Part 3)

- `camelCase`
  - First letter of words capitalized, except first
  - Words squished together/no separation
  - Used in many traditional coding languages
  - Used in Javascript (JS)
- `kebab-case`
  - All lowercase
  - Words separated with `-`
  - Traditionally used for HTML/CSS class names
  - Used for HTML attributes

Casing systems we use in 6150

- `CONSTANT_CASE`
  - Used in Javascript (JS) for specific **constants**
- `camelCase`
  - Used in Javascript (JS) for **variables**
- `MixedCase`
  - Used in Javascript (JS) for **components**
- `kebab-case`
  - Used for HTML **attributes**
  - Used for CSS/HTML **class names**
  - HTML allows for non-kebab-case class names
    - We will NOT use this outside of BEM

Using Box Model

- We do not yet know how to *layout* a page
- One step at a time
- Focus on styling element boxes right now

Basic Box Example - HTML

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Box Model Sample</title>
  <link href="/styles.css" rel="stylesheet"/>
</head>
<body>
  <p>Text with <a href="./index.html">link</a></p>
  <div>Here is more content</div>
</body>
</html>
```

Box Model CSS Starting Point

styles.css

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

Basic Box Properties

Make the element box visible:

```
p {
    background-color: burlywood;
    border: 1px solid black;
}
```

Dimensions:

```
p {
    background-color: burlywood;
    border: 1px solid black;
    height: 50px;
    width: 300px;
}
```

Padding is Between Border and Content

```
p {
    background-color: burlywood;
    border: 1px solid black;
    height: 50px;
    width: 300px;

    padding: 5px;
}
```

Try increasing/decreasing padding in DevTools

Margin is between border and neighbor elements

Make neighbor box visible

```
div {
  border: 1px solid black;
}
```

Notice the `<p>` has a DEFAULT margin!

```
p {
    background-color: burlywood;
    border: 1px solid black;
    height: 50px;
    width: 300px;
    padding: 5px;

    margin: 0px;
}
```

Try increasing/decreasing margin in DevTools

Quick Interruption: Let's Revisit DevTools

- Because previous semesters didn't use well
- You should use DevTools ALL THE TIME
    - "Where is this space coming from?"
    - "What styles are on this element"?
        - Browser has some DEFAULT styles
            - Styles you didn't set!
            - Ex: `<p>` has top/bottom margin!
- Coding should minimize "guessing"
- Use DevTools to *know* what is happening
    - Core Job skill, practice it now!

Most HTML elements are Inline or Block

- `display: inline;`
  - Take up size **based on content**
  - CSS **resizing highly limited**
  - **Does not break the "flow" of text**
- `display: block;`
  - Fill **width of container**
  - **Height as needed by content**
  - CSS **resizing fully available**
  - **Break text flow** before and after

Notes about inline elements

- Do not break flow
    - Means some sizing properties don't do anything

Notes about Block elements

Take up full-width of container by default

- AND break flow

Breaking flow means changing the size alone won't stop it

inline Example

```
a {
    background-color: aqua;
    border: 1px solid red;
    height: 30px;
    width: 50px;
}
```

height/width don't work!

- because `<a>` is `display: inline;` by default

Notes about inline block elements

```
display: inline-block;
```

- Does not break flow
- Does allow for resizing

If you are changing `display`, it will tend to be to `inline-block` or one of the layout options

inline-block Example

```
a {
    background-color: aqua;
    border: 1px solid red;
    height: 30px;
    width: 50px;

    display: inline-block;
}
```

Now height/width take effect!

Notes about floating

`float: left;` (etc)

Used to have inline elements flow around it

- Ex: paragraph of text wrapping around a small image

Do not use `float` for layout

- Was a common fix before flexbox/grids
- Only use to wrap text around an image
- A lot of outdated online advice

What If?

If an element matches different selectors?

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

Resolve via **specificity**

CSS Specificity

- `!important` is the most specific (overrules all)
    - *Only* use this to override an external library
- Inline CSS is the next most specific
    - You should also not be doing this
- id selectors (`#example`) are next
- class selectors (`.example`) are next
- element selectors (`p`) are next

Selectors can combine to increase specificity

- `.example.wrong` is more specific than `.example`
    - still less specific than `#example`

Same Specificity?

If two selectors have the same specificity

- the winner will be the "most recent"
    - later in the file or page

Avoid Specificity War

If you have multiple sources of CSS

- Different sources may use specificity to override
- This can lead to "specificity wars":
    - One source makes a selector more specific
    - But that breaks another place
    - So other source raises THEIR specificity
- There is only pain and tears in a specificity war

Scoping on a shared page

A semi-common pattern:

- Your content container has an `id`
- Use classes (not ids) for lower levels
- Use `#YOUR-ID .YOUR-CLASS` as your CSS pattern
    - That's a **descendant selector**

Only have to have one unique id per source of content

- Everyone otherwise uses classes

Emmet

- Editor may have "snippets"
    - Define expansions of known content
- Emmet is a generic standard
    - For HTML and CSS (and lorem ipsum text)

https://docs.emmet.io/

Lorem Ipsum

Fake text

- Taken randomly from an old latin speech
- See how a layout looks with "text-like" content
- Real content is always better
    - But rarely available at design time
- Many tools to generate "lorem text"
    - In-editor or websites to cut/paste

CSS Units

- `%` of container
- `vh` and `vw`
    - "viewport"
- `px` vs `rem` vs `em`
    - `px` is (mostly) fixed
        - fixed is often bad
    - `em` causes inheritance problem
    - Sizes based off of "root" font "em" width
        - "root" is `<html>` element
    - **https://css-tricks.com/html-vs-body-in-css/**
    - `rem` useful with browser text settings

So what units to use?

Users may have different text settings

- `px` for parts that don't change based on text size
- `rem` for parts that DO change based on text size

Do border sizes change based on text size?

- It Depends - you have to decide

Margin Collapse - A common source of confusion

Imagine the following code:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title></title>
  <link rel="stylesheet" href="styles.css"/>
</head>
<body>
  <header><h1>This is a top heading</h1></header>
  <main>
    <p>Paragraph 1</p>
    <p>Paragragh 2</p>
  </main>
  <footer>This is a footer</footer>
</body>
</html>
```

Sample CSS for Margin Collapse demo

```css
body {
  margin: 0;
  background-color: lime;
}

header, footer {
  background-color: #bada55;
}

main {
  background-color: #c0ffee;
}
```

Margin Collapse in action

- Paragraphs (`<p>`) are children of `<main>`
- Top and bottom of those paragraphs do NOT show `<main>` background color

Exploring with DevTools increases confusion

- `<header>` contains `<h1>`
    - But `<h1>` margin extends OUTSIDE `<header>`
- `<p>` are inside `<main>`
    - But `<p>` margins extend OUTSIDE `<main>`
- `<h1>` margin and top `<p>` margin OVERLAP

This is all due to **margin collapse**

What is Margin Collapse?

General rule of Box Model:

- The box contains the contents
- When `height` and `width` are `auto;` (the default)`
    - Box will size to fit the contents

Margins with **margin collapse** can violate this

- Collapse **upwards** (top) and **outwards** (parent)
- Only when margin collapse happens!
    - Requires a **block formatting context**
    - Never with `display: flex;` or `display: grid;`

Why does Margin Collapse exist?

Remember the original context of the web

- Sharing big linking text documents
    - Like Wikipedia

Margin Collapse makes a lot of things more convenient

- Paragraphs have top/bottom margins
    - But 2 `<p>` in a row won't get double margin

Margin Collapse makes OTHER things LESS convenient

- Like teaching/learning the box model

What do we do with this knowledge?

When debugging with DevTools

- If margins aren't included in parent content box
    - Margin collapse is to blame
- This is a rare spot DevTools doesn't help you

You can avoid Margin Collapse

- Switching to `display` of `flex`/`grid`
- By having `padding`
- By having a border on parent

CSS Custom Properties

Often we have values that we want to reuse

- Height/widths of elements interacted with (nav?)
- Colors (background, accent, highlight, etc)

Technically these are **custom properties**

- Sometimes called "CSS Variables"
- But they act like CSS properties
- Follow the normal cascading/precedence rules

Outside CSS

CSS took a long time to add "variables"

- Can't work everywhere even still

We will talk about SASS later in semester

- Has own solution for "variables"
- But SASS isn't actual CSS

This is the pure (but limited) CSS solution

Using a CSS Custom Property

Assign:

```
.some-selector {
  --my-var: black;
  --another: 5rem;
}
```

Use:

```
p {
  color: var(--my-var);
}
```

"Global" assign:

```
:root {  /* same as `html` */
  --main-bg-color: #BADA55;
}
```

Real World Example of CSS Custom Properties

Taken from **http://washingtonpost.com/**

```
a {
    color: var(--link-color);
    text-decoration: none
}

:root {
    --color-brand-blue-normal: #1955a5;
    --color-brand-blue-dark: #172a52;
    --color-ui-white: #fff;
    --color-ui-offwhite: #f7f7f7;
    --color-ui-gray-light: #d5d5d5;
    /* Cut ~100 lines */
    --primary-background: var(--color-ui-black);
    --secondary-background: var(--color-ui-gray-darkest);
    --primary-fill: var(--color-ui-white);
    --secondary-text: var(--color-ui-gray-light);
    --link-color: var(--color-brand-blue-normal)
}
```

Pseudo-classes

Added to a selector to indicate a state

- `:hover`
- `:focus` and `:focus-within`
- `:active`
- `:not()`
- `:first-child`
- `:nth-child()`

Pseudo-elements

Not elements, but allow you to style them like one

- `::selection`
- `::first-line` and `::first-letter`
- `::before` and `::after`
    - These require a `content` property

CSS Properties

- `filter`
    - `filter: brightness()`
- `opacity`
- `font-family`
- `visibility`
    - Hides without removing from layout
    - Can be good/bad for accessibility
        - More on accessibility in a later class

CSS Functions

- `calc()`
- `max()` and `min()`
- `clamp()`
    - 3 args, preferred should be a value that changes

Media Queries

- Wraps CSS Rules
- Rules applied or not based on query
- Says if the rules are matched

Screen Width

If CONDITION, apply CSS rules

```
@media (min-width: 1000px) {
  body {
    background-color: red;
  }
}
```

Reduced Motion

- Options are `no-preference` or `reduce`
- Which involves less work?
- Which is "safer"?

```
@media (prefers-reduced-motion: no-preference) {
  .my-element {
    animation: flashy-zoom-in-out 1s;
  }
}
```

Orientation

- If you care past width…

```
@media (orientation: portrait) {
  body {
    display: flex;
    flex-direction: column;
  }
}
```

Printing

A deep rabbithole

- Alternative to generating PDFs
- Not always the best alternative

```
@media print {
  h3 {
    page-break-before: always;
  }
}
```

Summary - CSS Purpose

CSS provides **rules for appearance**

- Based on structure
- When structure matches rules:
    - Appearance applies

Summary - Box Model

Every element is a "box" of "boxes"

- Content width and height
- Padding width and height
- Border width and height
- Margin width and height

`box-sizing` property

- `content-box`: `width` and `height` are content
- `border-box`: w + h are content+padding+border

Summary - Stylesheets

CSS added to your page:

- Inline in elements
    - Rare except for specific needs
    - Can't reuse
- In `<style>` element
    - Rare without tools
    - Can't reuse
- As a separate `.css` file
    - Via `<link>` element with `href` attribute
    - Common
    - Multiple CSS files when different reuse cases

Summary - Rules

- Rules are selector(s) + declarations
- Invalid rules skipped over
    - No error messages

Summary - Selectors

- Comma separated
- If any selectors match, declarations applied
- Symbols indicate type of selector
    - No symbol = element selector
- Connected symbols = must match all:
    - `div#root.active`
        - "`<div>` with id `root` and class of `active`"
- Space = descendant, easiest to read backwards
    - `div .wrong`
        - "Element with a class of `wrong` that is a descendant of a `<div>`"

Summary - Declarations

- **kebab-case**, each ends in semicolon
- **Prefixes** (`--webkit-*`) mostly retired
- **Shorthand properties** set multiple properties
    - Use where understandable
    - Avoid where confusing (be explicit then)
- Color values can be
    - RGB 3,4,6,8 hex characters starting with `#`
    - `rgb()` or `rgba()` (decimal values)
    - `hsl()` or `hwb()`
    - Transparency/opacity is "alpha"
        - 0.0-1.0 or 0%-100%

Summary - Cascade

All matching rules are applied

- Some properties inherited from parent element
  - Most Text and color related properties
  - Not size, display, or layout related properties

Summary - Specificity

If property gets different values, which takes effect?

- All applied, but some override others
- Selector Specificity
    - Which rules have properties overridden
    - `!important` > inline > id > classes > element
- Same specificity:
    - Most "recent" overrides
    - Order of file loading
    - Place in file
    - Part of why class selectors most common
        - All the same specificity