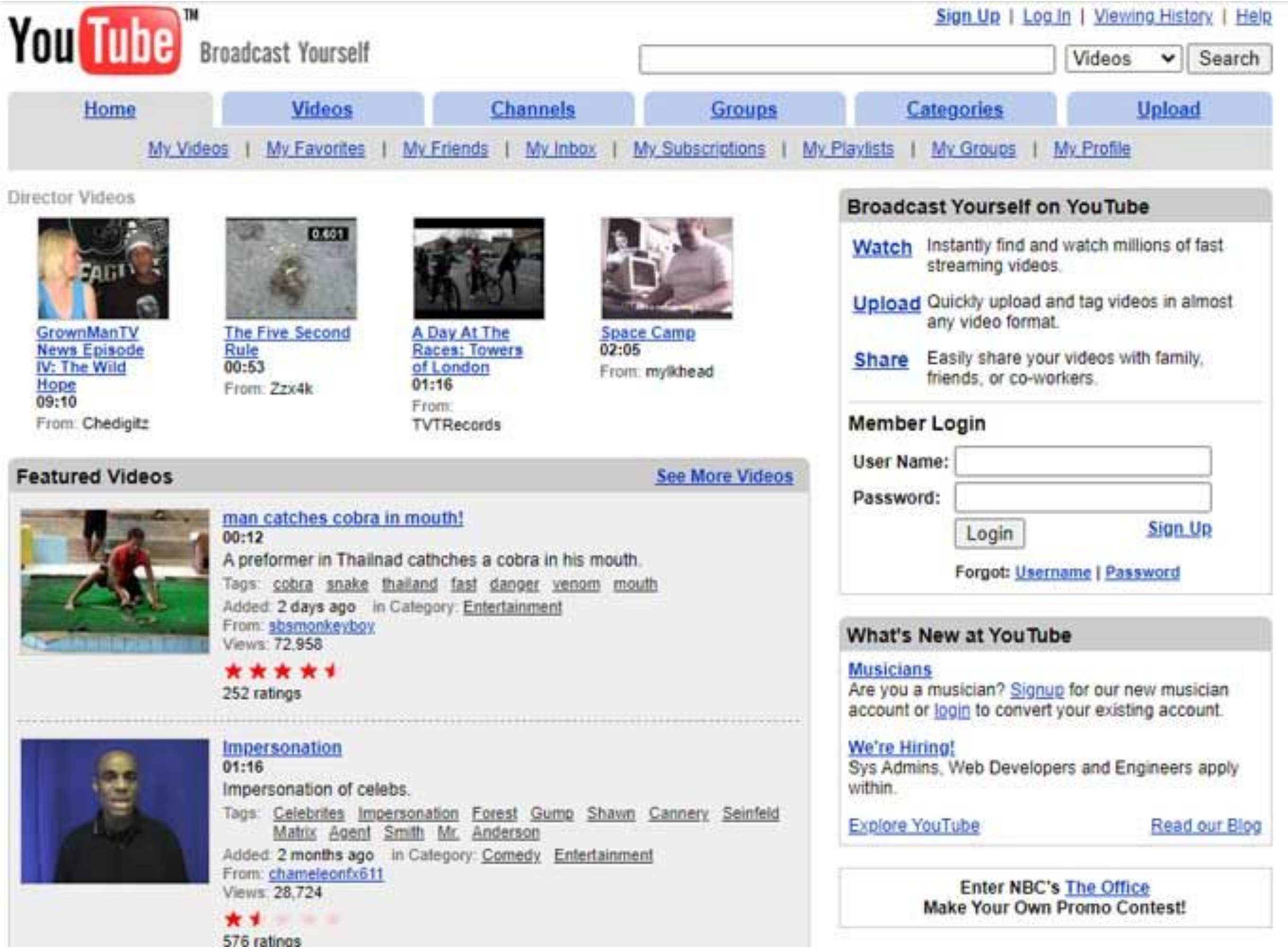

YouTube™ Broadcast Yourself
Sign Up | Log In | Viewing History | Help
Videos
Search
Home
Videos
Channels
Groups
Categories
Upload
My Videos | My Favorites | My Friends | My Inbox | My Subscriptions | My Playlists | My Groups | My Profile
Login
0.601

Director Videos

GrownManTV News Episode IV: The Wild Hope
09:10
From: Chedigitz

The Five Second Rule
00:53
From: Zzx4k

A Day At The Races: Towers of London
01:16
From: TVTRecords

Space Camp
02:05
From: mylkhead

Featured Videos See More Videos

man catches cobra in mouth!
00:12
A preformer in Thailnad cathches a cobra in his mouth.
Tags: cobra snake thailand fast danger venom mouth
Added: 2 days ago in Category: Entertainment
From: absmonkeyboy
Views: 72,958
★★★★½
252 ratings

Impersonation
01:16
Impersonation of celebs.
Tags: Celebrites Impersonation Forest Gump Shawn Cannery Seinfeld Matrix Agent Smith Mr. Anderson
Added: 2 months ago in Category: Comedy Entertainment
From: chameleonfx611
Views: 28,724
★½☆☆☆
576 ratings

Broadcast Yourself on YouTube

Watch Instantly find and watch millions of fast streaming videos.

Upload Quickly upload and tag videos in almost any video format.

Share Easily share your videos with family, friends, or co-workers.

Member Login

User Name:
Password:
Sign Up
Forgot: Username | Password

What's New at YouTube

Musicians
Are you a musician? Signup for our new musician account or login to convert your existing account.

We're Hiring!
Sys Admins, Web Developers and Engineers apply within.

Explore YouTube Read our Blog

Enter NBC's The Office
Make Your Own Promo Contest!

mediaire CSS Workshop

Enhancing our Styling Practices for Maintainability & UX

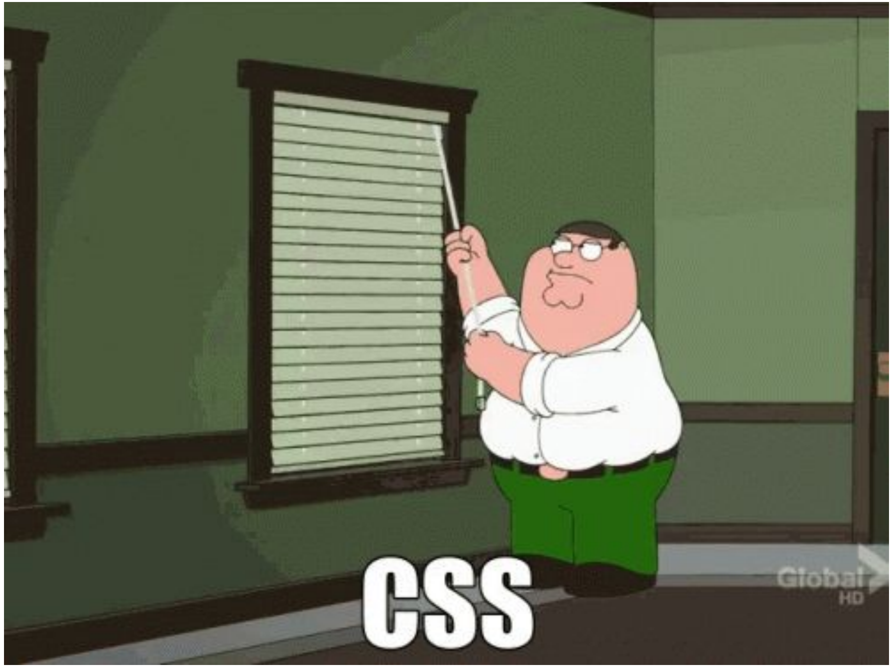
Christopher Nagy - Software Team - mediaire

23.01.2025

Workshop Goals

- Review **CSS basics** and their application
- Understand how **SCSS** enhances CSS maintainability and discuss its downsides
- Explore **Tailwind CSS** as a utility-first alternative to traditional CSS
- Establish shared knowledge about maintainable styling and its importance

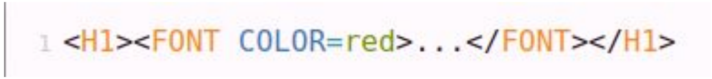
Source: ChatGPT


CSS
Global HD

What is CSS?

Cascading **S**tyle**s**heets: one of the 3 backbones of the web

- selective styling of HTML elements
- Before: styling directly in HTML → cumbersome, hard to maintain



```
<H1><FONT COLOR=red>...</FONT></H1>
```

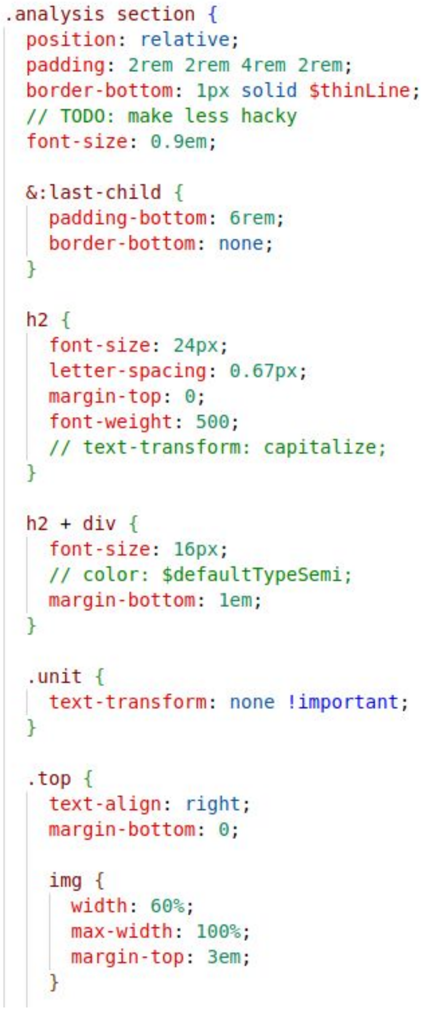
Benefits

- Separation of style and content
- Efficiency: Reusable stylesheets
- Customization & branding
- Improved UX & Accessibility

Why focus on Styling?

It just makes up a huge part of the frontend code.

- **Consistency** in design improves the overall user experience
- **Maintainable** and well-structured styles reduce time spent fixing issues and enhance productivity
- **Adaptability** to new technologies / products ensures scalability as the company grows



```
.analysis section {
  position: relative;
  padding: 2rem 2rem 4rem 2rem;
  border-bottom: 1px solid $thinLine;
  // TODO: make less hacky
  font-size: 0.9em;

  &:last-child {
    padding-bottom: 6rem;
    border-bottom: none;
  }

  h2 {
    font-size: 24px;
    letter-spacing: 0.67px;
    margin-top: 0;
    font-weight: 500;
    // text-transform: capitalize;
  }

  h2 + div {
    font-size: 16px;
    // color: $defaultTypeSemi;
    margin-bottom: 1em;
  }

  .unit {
    text-transform: none !important;
  }

  .top {
    text-align: right;
    margin-bottom: 0;

    img {
      width: 60%;
      max-width: 100%;
      margin-top: 3em;
    }
```

Part 1

CSS Basics

The CSS Box Model

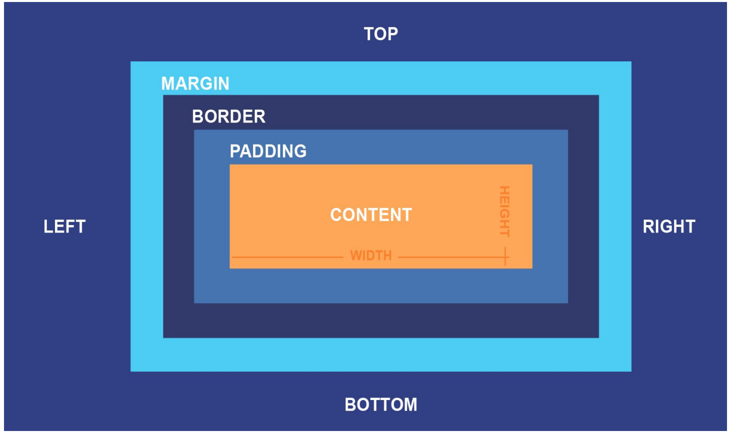
Every element is a rectangular box made of:

Content: The text or image inside the box.

Padding: Space between the content and the border.

Border: Surrounds the padding (optional).

Margin: Space outside the border, separating the box from others.



`box-sizing` CSS property: determines how total width/height of the box is calculated (includes padding/border or not)

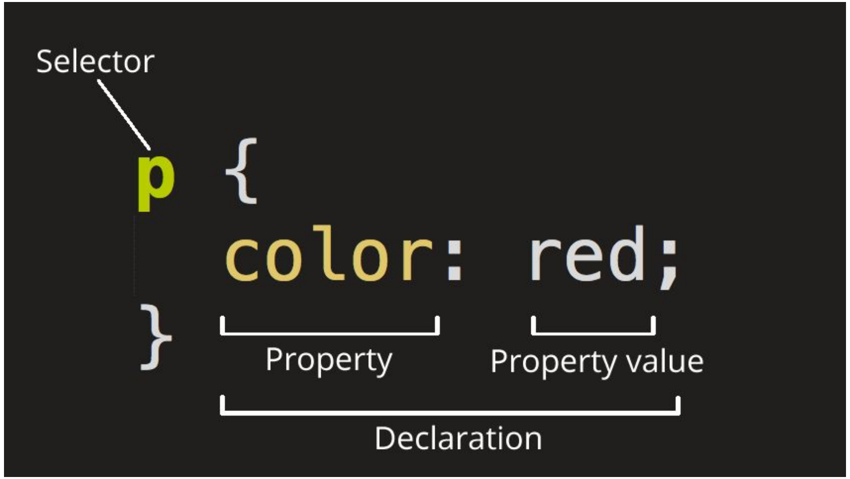
CSS Rules

CSS is a rule-based language for styling HTML.
Rules consist of:

Selector: Targets an element or group of elements (e.g. via ID or class)

Declaration: Maps a **property** to a value, changing the element's styling

Property: keyword that can take on a range of values (e.g. color)



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

Note: Sometimes rule & selector are used as synonyms.

A

- abs()
- accent-color
- acos()
- :active
- additive-symbols (@counter-style)
- ::after (:after)
- align-content
- align-items
- align-self
- align-tracks
- all
- <an-plus-b>
- anchor()
- anchor-name
- anchor-scope
- anchor-size()
- <angle>
- <angle-percentage>
- animation
- animation-composition
- animation-delay
- animation-direction
- animation-duration
- animation-fill-mode
- animation-iteration-count
- animation-name
- animation-play-state
- animation-range
- animation-range-end
- animation-range-start
- animation-timeline
- animation-timing-function
- @annotation
- :any-link
- appearance
- ascent-override (@font-face)
- asin()
- aspect-ratio
- atan()
- atan2()
- attr()

B

- ::backdrop
- backdrop-filter
- backface-visibility
- background
- background-attachment
- background-blend-mode
- background-clip
- background-color
- background-image
- background-origin
- background-position
- background-position-x
- background-position-y
- background-repeat
- background-size
- base-palette (@font-palette-values)
- <blend-mode>
- block-size
- blur()
- border
- border-block
- border-block-color
- border-block-end
- border-block-end-color
- border-block-end-style
- border-block-end-width
- border-block-start
- border-block-start-color
- border-block-start-style
- border-block-start-width
- border-block-style
- border-block-width
- border-bottom
- border-bottom-color
- border-bottom-left-radius
- border-bottom-right-radius
- border-bottom-style
- border-bottom-width
- border-collapse
- border-color
- border-end-end-radius
- border-end-start-radius
- border-image
- border-image-outset
- border-inline
- border-inline-color
- border-inline-end
- border-inline-end-color
- border-inline-end-style
- border-inline-end-width
- border-inline-start
- border-inline-start-color
- border-inline-start-style
- border-inline-start-width
- border-inline-style
- border-inline-width
- border-left
- border-left-color
- border-left-style
- border-left-width
- border-radius
- border-right
- border-right-color
- border-right-style
- border-right-width
- border-spacing
- border-start-end-radius
- border-start-start-radius
- border-style
- border-top
- border-top-color
- border-width
- bottom
- @bottom-left-corner
- box-decoration-break
- box-shadow
- box-sizing
- break-after
- break-before
- break-inside
- brightness()

C

- calc()
- calc-size()
- caption-side
- caret
- caret-color
- caret-shape
- @character-variant
- @charset
- :checked
- circle()
- clamp()
- clear
- clip-path
- clip-rule
- <color>
- color
- color-interpolation-filters
- color-scheme
- column-count
- column-fill
- column-gap
- column-rule
- column-rule-color
- column-rule-style
- column-rule-width
- column-span
- column-width
- columns
- conic-gradient()
- contain
- contain-intrinsic-block-size
- contain-intrinsic-height
- contain-intrinsic-inline-size
- contain-intrinsic-size
- contain-intrinsic-width
- container
- container-name
- container-type
- content
- content-visibility
- contrast()
- cos()
- <counter>
- counter-increment
- counter-reset
- counter-set
- @counter-style
- counters()
- cross-fade()
- cubic-bezier()
- ::cue
- ::cue-region
- :current
- cursor
- <custom-ident>
- cx
- cy
- length#cap
- length#ch
- length#cm

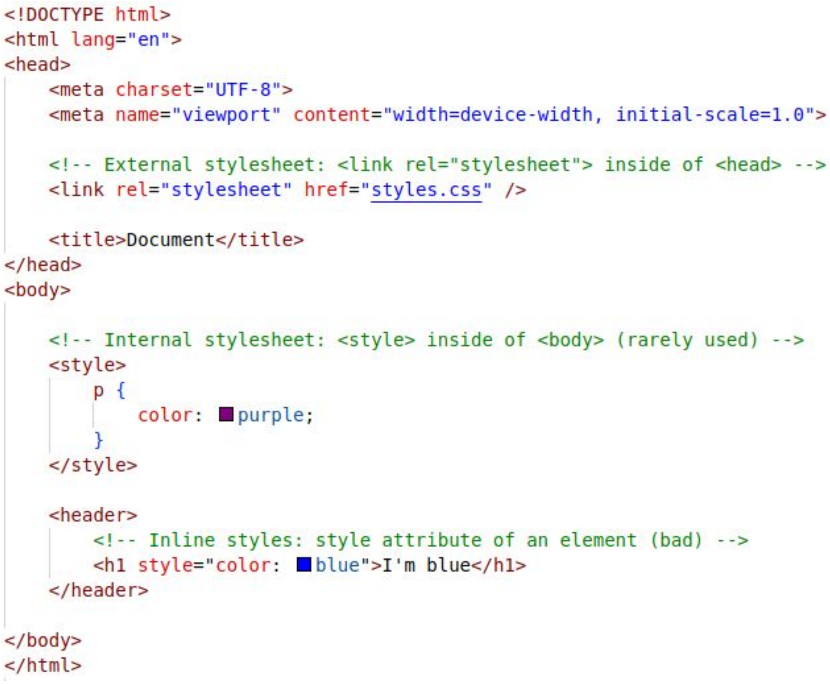
How to apply CSS

1. **External** stylesheet (most common)

2. **Internal** stylesheet

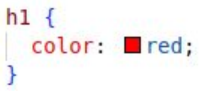
3. **Inline** styles (to avoid - why?)

index.html



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!-- External stylesheet: <link rel="stylesheet"> inside of <head> -->
    <link rel="stylesheet" href="styles.css" />

    <title>Document</title>
</head>
<body>

    <!-- Internal stylesheet: <style> inside of <body> (rarely used) -->
    <style>
        p {
            color: purple;
        }
    </style>

    <header>
        <!-- Inline styles: style attribute of an element (bad) -->
        <h1 style="color: blue">I'm blue</h1>
    </header>

</body>
</html>
```

styles.css



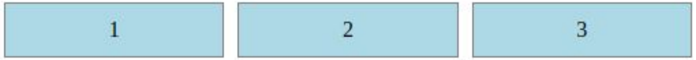
```css
h1 {
  color: red;
}
```

CSS Basics: Flexbox

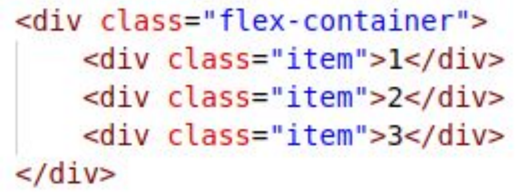
One-dimensional layouts: horizontal/vertical

→ simple and easy to use for most layouts
→ frequently used

Result

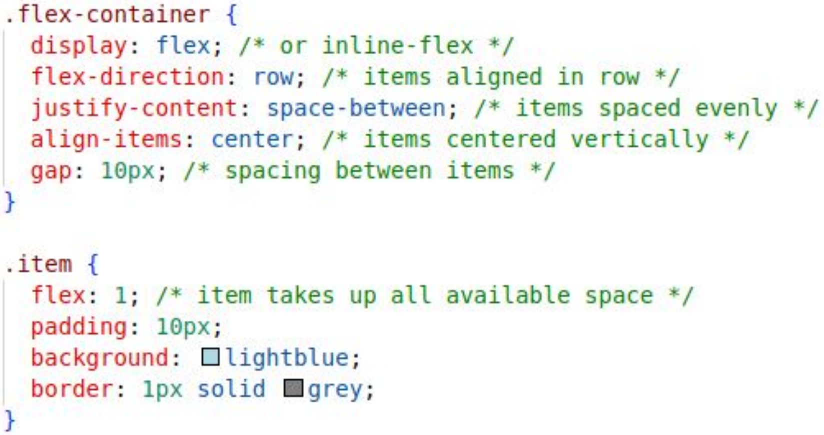


Index.html (excerpt)



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

styles.css



```
.flex-container {
  display: flex; /* or inline-flex */
  flex-direction: row; /* items aligned in row */
  justify-content: space-between; /* items spaced evenly */
  align-items: center; /* items centered vertically */
  gap: 10px; /* spacing between items */
}

.item {
  flex: 1; /* item takes up all available space */
  padding: 10px;
  background: □lightblue;
  border: 1px solid ■grey;
}
```

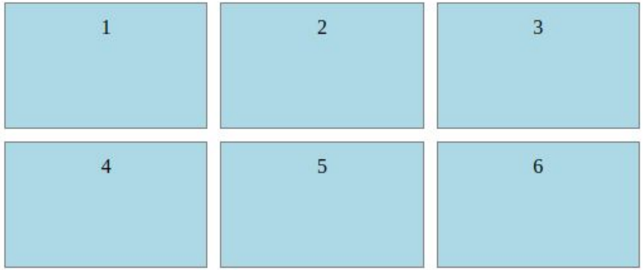
Cool visual guide: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

CSS Basics: Grid

Two-dimensional: rows & columns

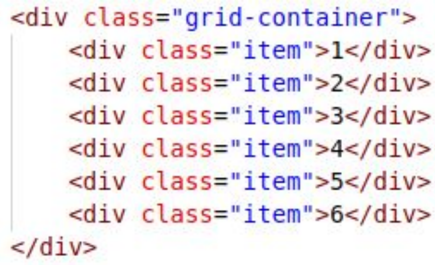
→ for more complex layouts
→ less used, but has its use cases

Result



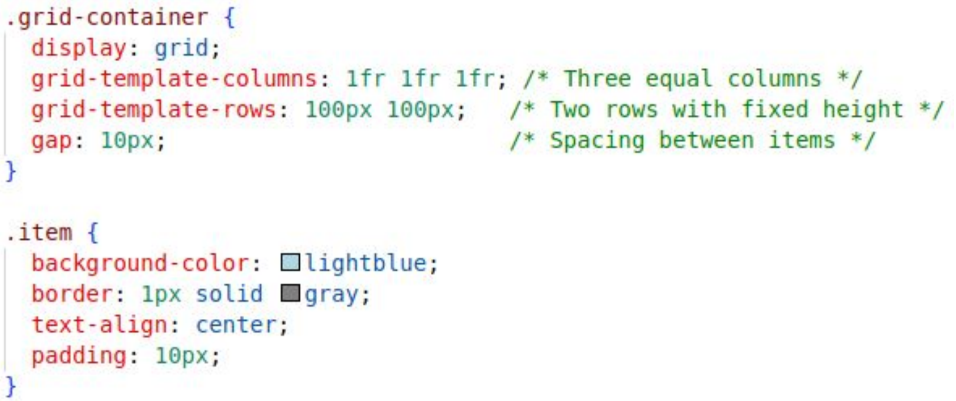
1	2	3
4	5	6

Index.html (excerpt)



```html
<div class="grid-container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
</div>
```

styles.css



```css
.grid-container {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr; /* Three equal columns */
  grid-template-rows: 100px 100px;   /* Two rows with fixed height */
  gap: 10px;                         /* Spacing between items */
}

.item {
  background-color: □lightblue;
  border: 1px solid ■gray;
  text-align: center;
  padding: 10px;
}
```

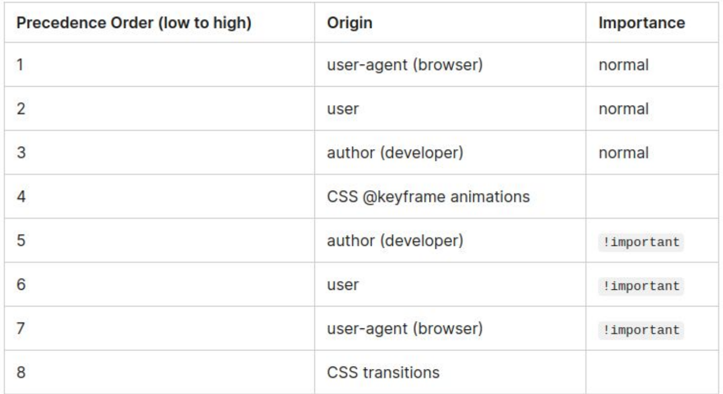
Why is my CSS not applied?

Common problem: multiple CSS rules target the same element → which properties to apply?

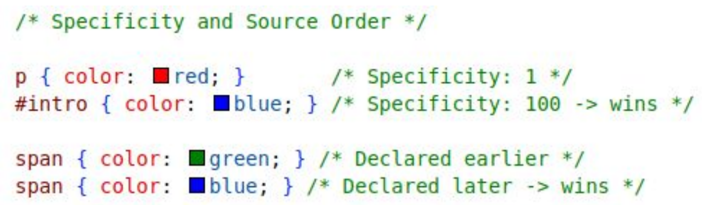
→ **Cascade** Algorithm: “origin and order matter”

Properties are applied based on:

1. **Origin (Precedence)**: Where the style comes from (browser < user < developer)
2. **Specificity (“priority”)**: more specific selectors override less specific ones
3. **Source Order**: Later rules in the CSS override earlier rules



Precedence 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	



```
/* Specificity and Source Order */

p { color: 🟥red; }       /* Specificity: 1 */
#intro { color: 🟦blue; } /* Specificity: 100 -> wins */

span { color: 🟩green; } /* Declared earlier */
span { color: 🟦blue; } /* Declared later -> wins */
```

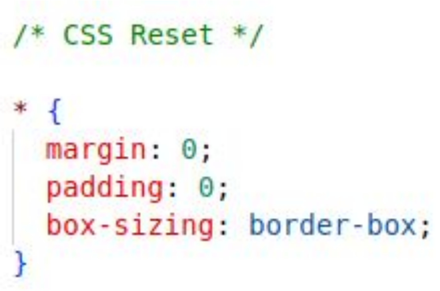
Browser Defaults & CSS Reset

Browsers add default styles to every HTML they render (User Agent Styles)

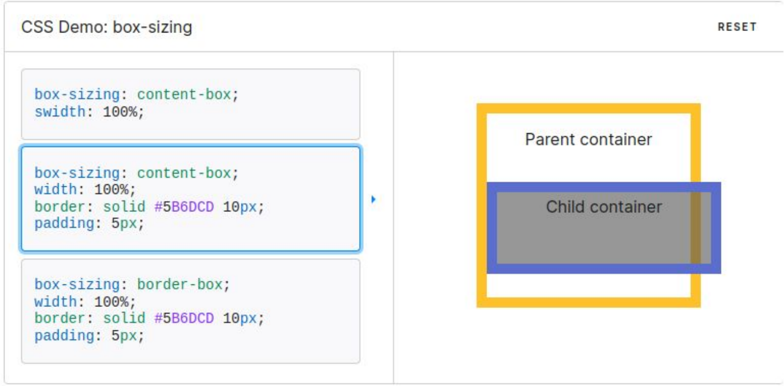
→ Readability, Accessibility, Consistency

However: may clash with custom CSS or introduce unwanted side-effects (e.g. `box-sizing: content-box`)

→ Usually new projects should use a “CSS Reset” stylesheet or a library (like normalize.css or Tailwind)



```
/* CSS Reset */

* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}
```



CSS Demo: box-sizing

```
box-sizing: content-box;
swidth: 100%;
```

```
box-sizing: content-box;
width: 100%;
border: solid #5B6DCD 10px;
padding: 5px;
```

```
box-sizing: border-box;
width: 100%;
border: solid #5B6DCD 10px;
padding: 5px;
```

Task 1

Basic CSS

Clone repo: css-workshop

→ task1-css

(~15 min)

Task 1

Basic CSS

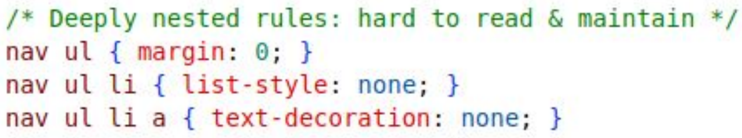
Reflection Questions

- How was the experience, were there any issues?
- What would change for bigger code bases?
- Would you use native CSS in our frontend projects? Why or why not?
- What issues can you imagine in larger projects?

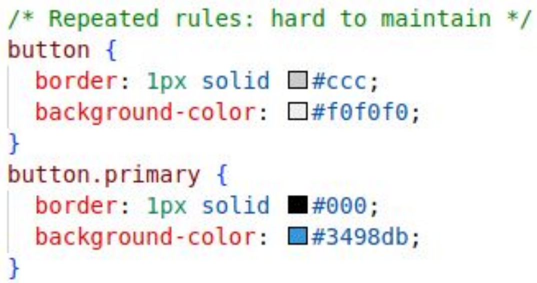
Challenges using Pure CSS

- Deeper HTML nesting → complex CSS
- No mixins: reusable rules with slight value variations need to be duplicated
- No modularity: no built-in way to organize code into reusable modules
- Limited calculation functionalities
- No variables → doesn't apply anymore (since 2017)

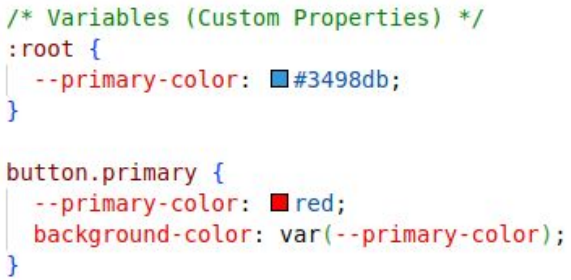
→ **Hard to maintain,** especially for large teams & projects with a lot of style changes / complex HTML



```css
/* Deeply nested rules: hard to read & maintain */
nav ul { margin: 0; }
nav ul li { list-style: none; }
nav ul li a { text-decoration: none; }
```



```css
/* Repeated rules: hard to maintain */
button {
  border: 1px solid ▨#ccc;
  background-color: ⬜#f0f0f0;
}
button.primary {
  border: 1px solid ⬛#000;
  background-color: 🟦#3498db;
}
```



```css
/* Variables (Custom Properties) */
:root {
  --primary-color: 🟦#3498db;
}

button.primary {
  --primary-color: 🟥red;
  background-color: var(--primary-color);
}
```

Part II

Preprocessors & Sass

CSS Preprocessors

Idea: extend CSS features to improve maintainability

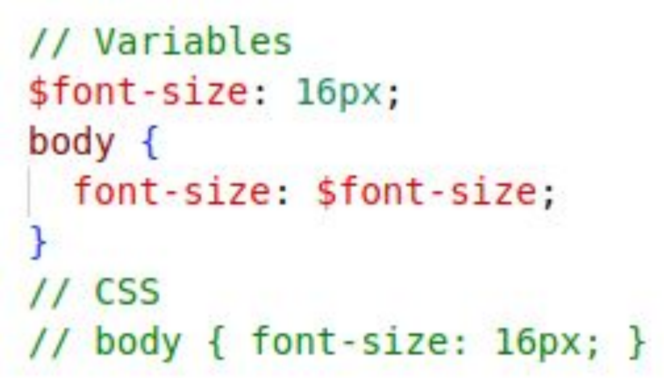
- Provide features like variables, mixins, nested rules
- Custom language is compiled to CSS before runtime

Popular examples:

1. **Sass** (Syntactically Awesome Stylesheets)
2. Less
3. Stylus

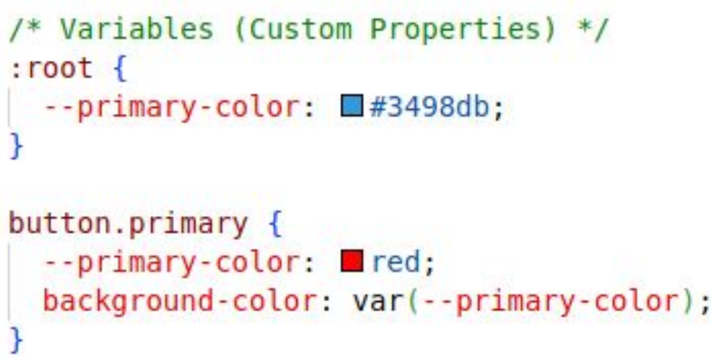
SCSS: Variables

SCSS



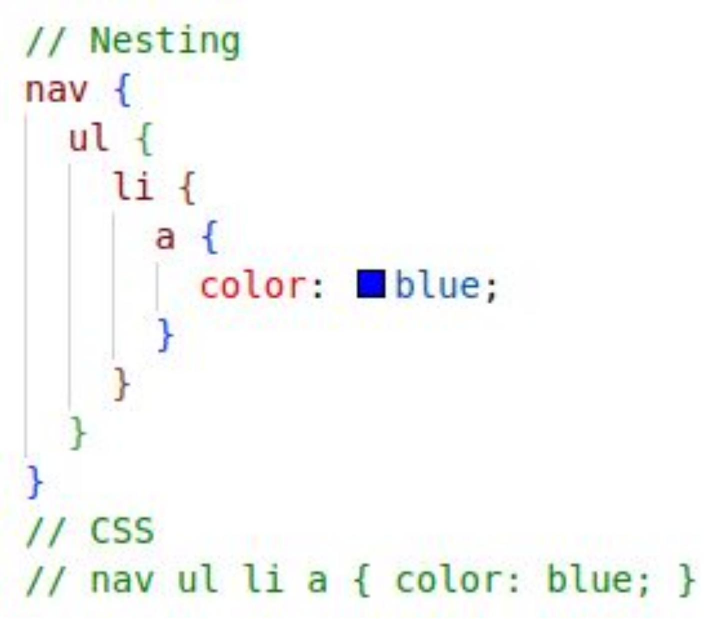
```scss
// Variables
$font-size: 16px;
body {
  font-size: $font-size;
}
// CSS
// body { font-size: 16px; }
```

CSS



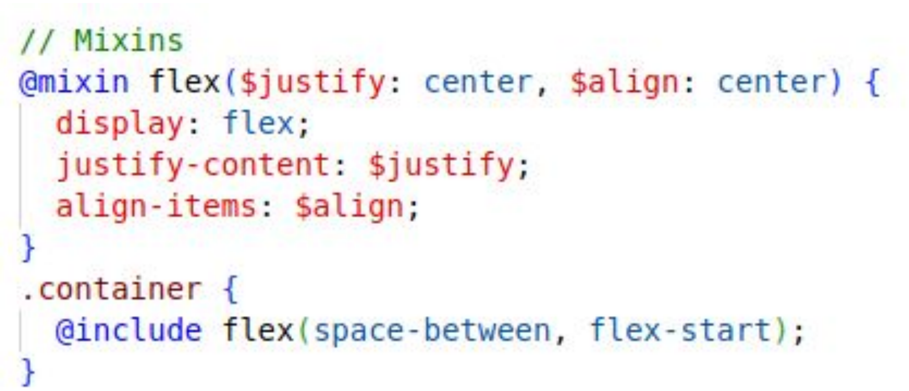
```css
/* Variables (Custom Properties) */
:root {
  --primary-color: 🟦#3498db;
}

button.primary {
  --primary-color: 🟥red;
  background-color: var(--primary-color);
}
```

SCSS: Nesting

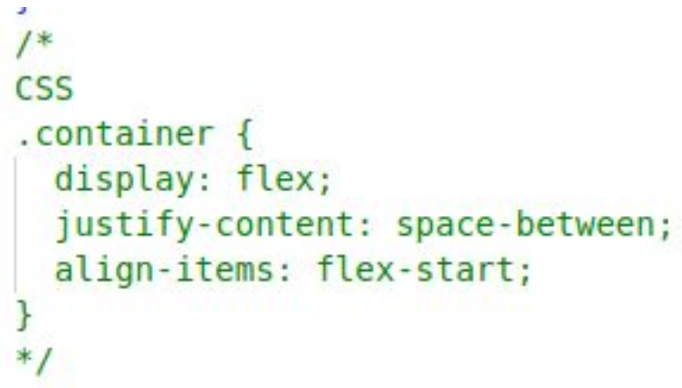


```
// Nesting
nav {
  ul {
    li {
      a {
        color: blue;
      }
    }
  }
}
// CSS
// nav ul li a { color: blue; }
```

SCSS: Mixins

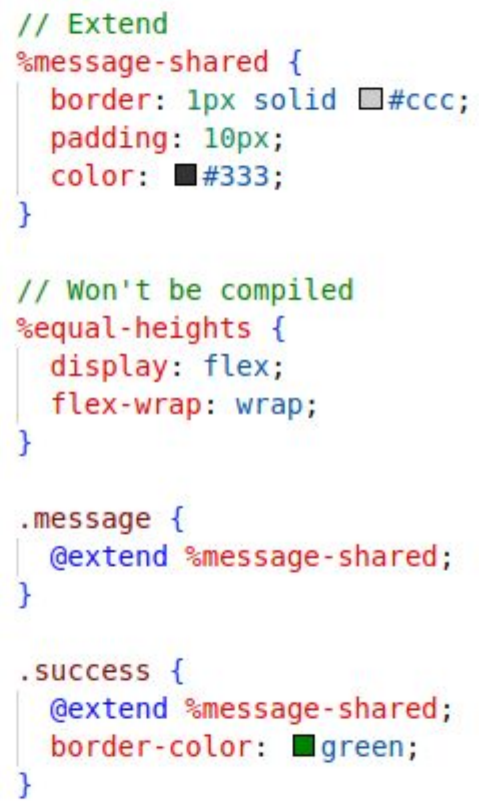


```
// Mixins
@mixin flex($justify: center, $align: center) {
  display: flex;
  justify-content: $justify;
  align-items: $align;
}
.container {
  @include flex(space-between, flex-start);
}
```

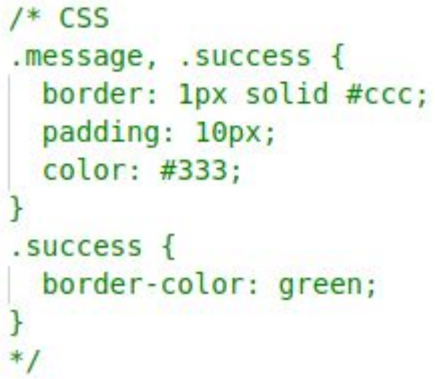


```
/*
CSS
.container {
  display: flex;
  justify-content: space-between;
  align-items: flex-start;
}
*/
```

SCSS: Extends

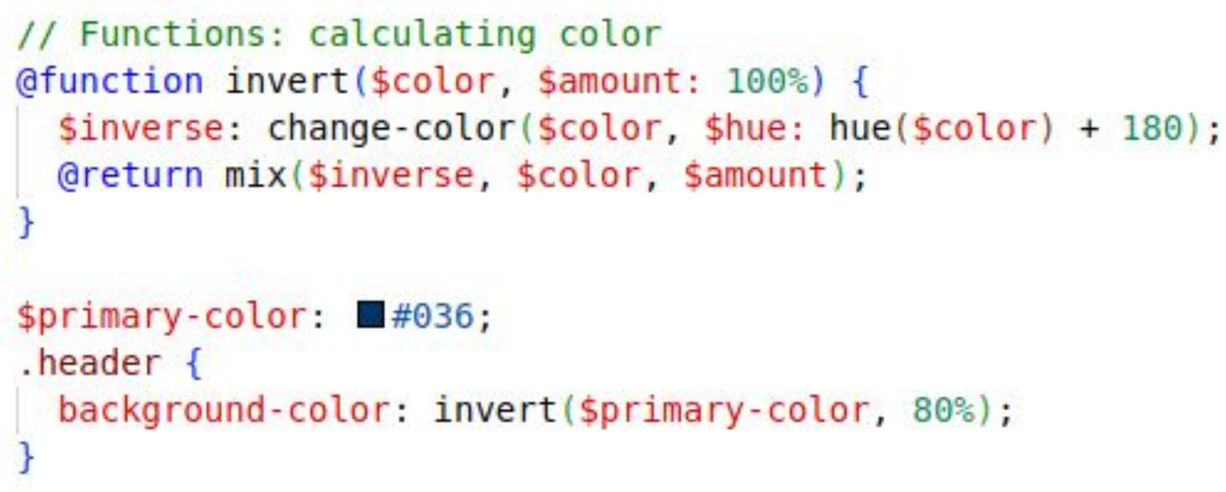


```scss
// Extend
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}

// Won't be compiled
%equal-heights {
  display: flex;
  flex-wrap: wrap;
}

.message {
  @extend %message-shared;
}

.success {
  @extend %message-shared;
  border-color: green;
}
```



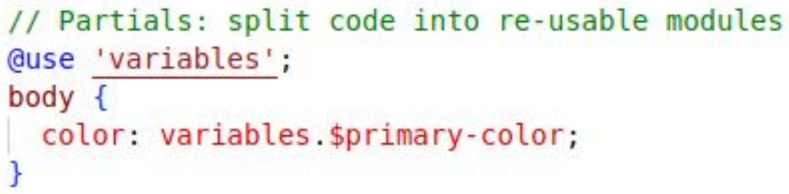
```css
/* CSS
.message, .success {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}
.success {
  border-color: green;
}
*/
```

SCSS: Functions

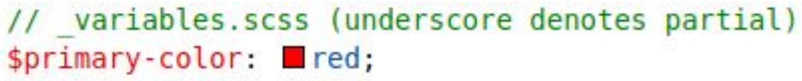


```scss
// Functions: calculating color
@function invert($color, $amount: 100%) {
  $inverse: change-color($color, $hue: hue($color) + 180);
  @return mix($inverse, $color, $amount);
}

$primary-color: ■#036;
.header {
  background-color: invert($primary-color, 80%);
}
```

SCSS: Partials

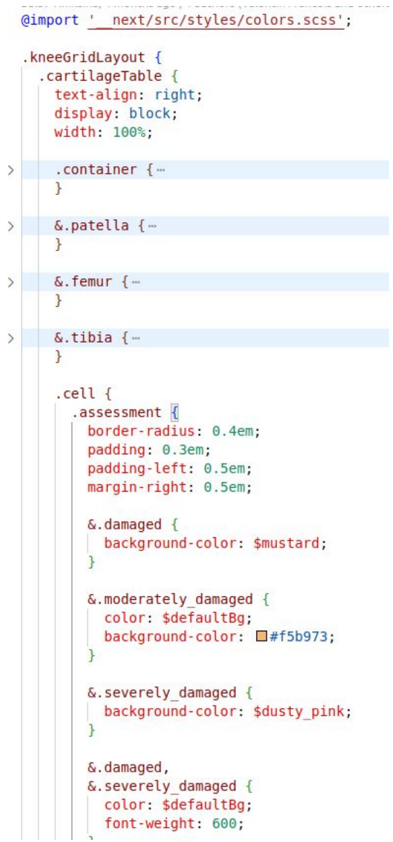


```scss
// Partials: split code into re-usable modules
@use 'variables';
body {
  color: variables.$primary-color;
}
```

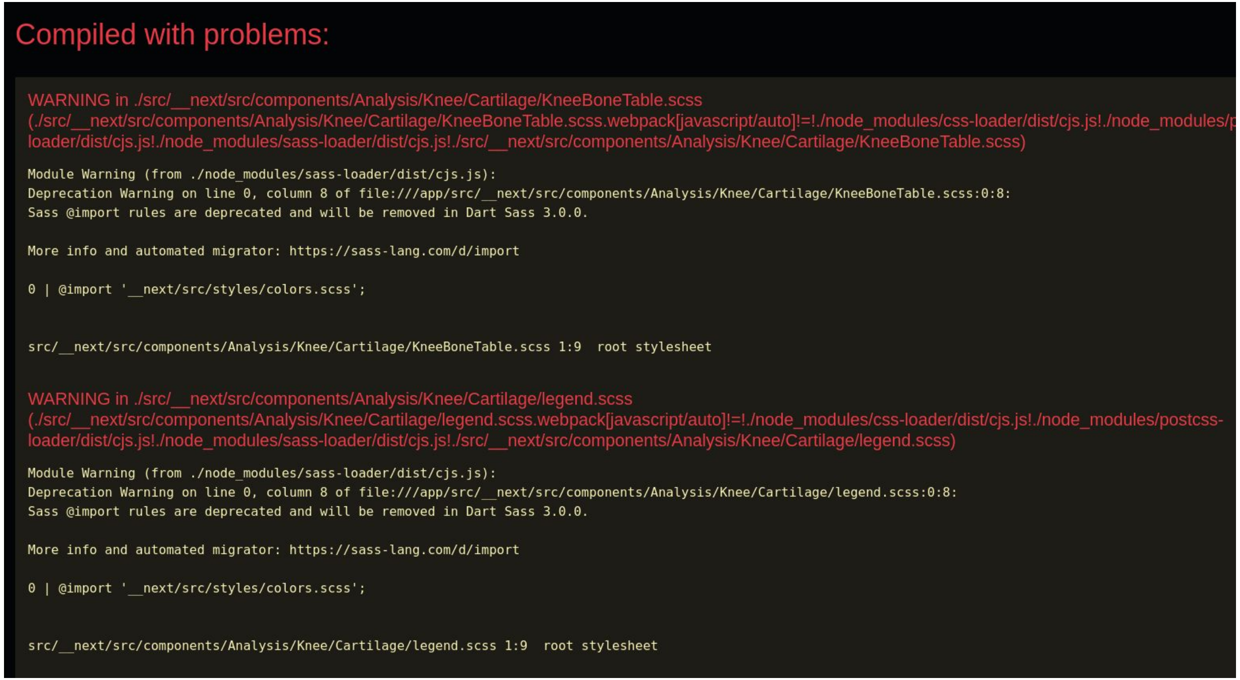


```scss
// _variables.scss (underscore denotes partial)
$primary-color: 🟥 red;
```

SCSS: Partials



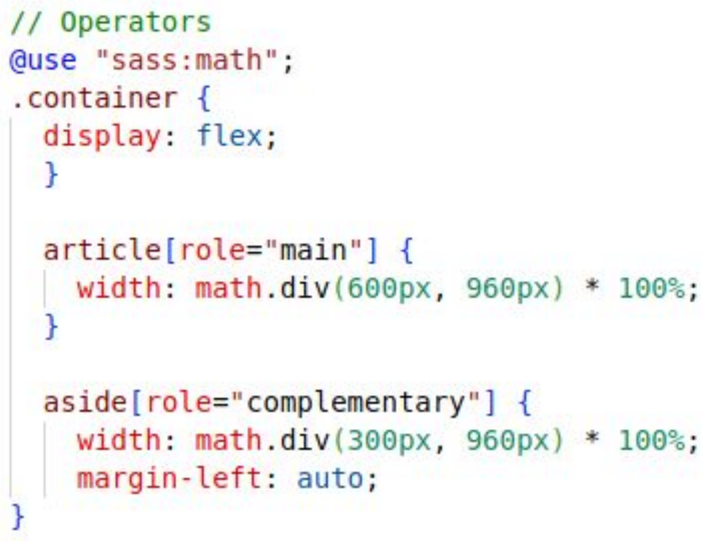
```
@import '__next/src/styles/colors.scss';

.kneeGridLayout {
  .cartilageTable {
    text-align: right;
    display: block;
    width: 100%;

    .container {⋯
    }

    &.patella {⋯
    }

    &.femur {⋯
    }

    &.tibia {⋯
    }

    .cell {
      .assessment {
        border-radius: 0.4em;
        padding: 0.3em;
        padding-left: 0.5em;
        margin-right: 0.5em;

        &.damaged {
          background-color: $mustard;
        }

        &.moderately_damaged {
          color: $defaultBg;
          background-color: #f5b973;
        }

        &.severely_damaged {
          background-color: $dusty_pink;
        }

        &.damaged,
        &.severely_damaged {
          color: $defaultBg;
          font-weight: 600;
```



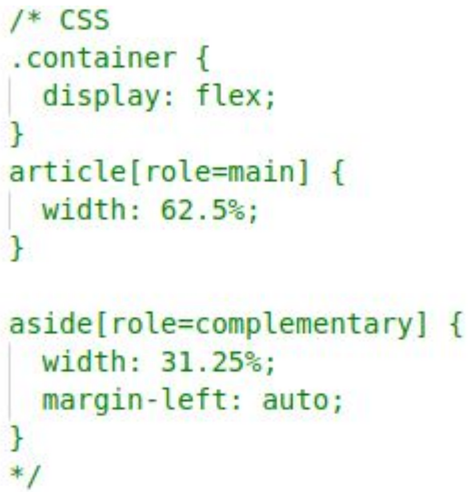
Compiled with problems:

```
WARNING in ./src/__next/src/components/Analysis/Knee/Cartilage/KneeBoneTable.scss
(./src/__next/src/components/Analysis/Knee/Cartilage/KneeBoneTable.scss.webpack[javascript/auto]!=!./node_modules/css-loader/dist/cjs.js!./node_modules/p
loader/dist/cjs.js!./node_modules/sass-loader/dist/cjs.js!./src/__next/src/components/Analysis/Knee/Cartilage/KneeBoneTable.scss)

Module Warning (from ./node_modules/sass-loader/dist/cjs.js):
Deprecation Warning on line 0, column 8 of file:///app/src/__next/src/components/Analysis/Knee/Cartilage/KneeBoneTable.scss:0:8:
Sass @import rules are deprecated and will be removed in Dart Sass 3.0.0.

More info and automated migrator: https://sass-lang.com/d/import

0 | @import '__next/src/styles/colors.scss';


src/__next/src/components/Analysis/Knee/Cartilage/KneeBoneTable.scss 1:9  root stylesheet

WARNING in ./src/__next/src/components/Analysis/Knee/Cartilage/legend.scss
(./src/__next/src/components/Analysis/Knee/Cartilage/legend.scss.webpack[javascript/auto]!=!./node_modules/css-loader/dist/cjs.js!./node_modules/postcss-
loader/dist/cjs.js!./node_modules/sass-loader/dist/cjs.js!./src/__next/src/components/Analysis/Knee/Cartilage/legend.scss)

Module Warning (from ./node_modules/sass-loader/dist/cjs.js):
Deprecation Warning on line 0, column 8 of file:///app/src/__next/src/components/Analysis/Knee/Cartilage/legend.scss:0:8:
Sass @import rules are deprecated and will be removed in Dart Sass 3.0.0.

More info and automated migrator: https://sass-lang.com/d/import

0 | @import '__next/src/styles/colors.scss';


src/__next/src/components/Analysis/Knee/Cartilage/legend.scss 1:9  root stylesheet
```

SCSS: Math Operators



```scss
// Operators
@use "sass:math";
.container {
  display: flex;
  }

  article[role="main"] {
    width: math.div(600px, 960px) * 100%;
  }

  aside[role="complementary"] {
    width: math.div(300px, 960px) * 100%;
    margin-left: auto;
}
```



```css
/* CSS
.container {
  display: flex;
}
article[role=main] {
  width: 62.5%;
}

aside[role=complementary] {
  width: 31.25%;
  margin-left: auto;
}
*/
```

Sass for the win…?

- **Reusability**: variables, mixins
- **Readability**: nested rules
- **Modularity**: split code into partials
- Programming features: if, for, each, functions
- Backwards compatibility (CSS)
- Integration into libraries like Vite / Webpack

→ fosters **maintainability**

→ made for large projects

Sass Downsides

- Preprocessing step → +build time & complexity
- Learning curve: specific syntax and paradigms (e.g. @use and namespaces)
- Risk of overuse/”unintended” use → bad maintainability
- Debugging styles can be challenging due to additional step
- Native CSS is evolving
- Dependency on Sass compiler

Conclusion: When (not) to use Sass?

- Useful tool for big projects and teams
- Prerequisite: developer needs to understand its paradigms and syntax
- Sass should not be the default choice, but depend on project requirements & size
- Don't use SCSS for simple projects. Native CSS fulfills most use cases

Task 2
Refactoring w/ Sass

task2-sass

(~20 min)

Task 2

Refactoring w/ Sass

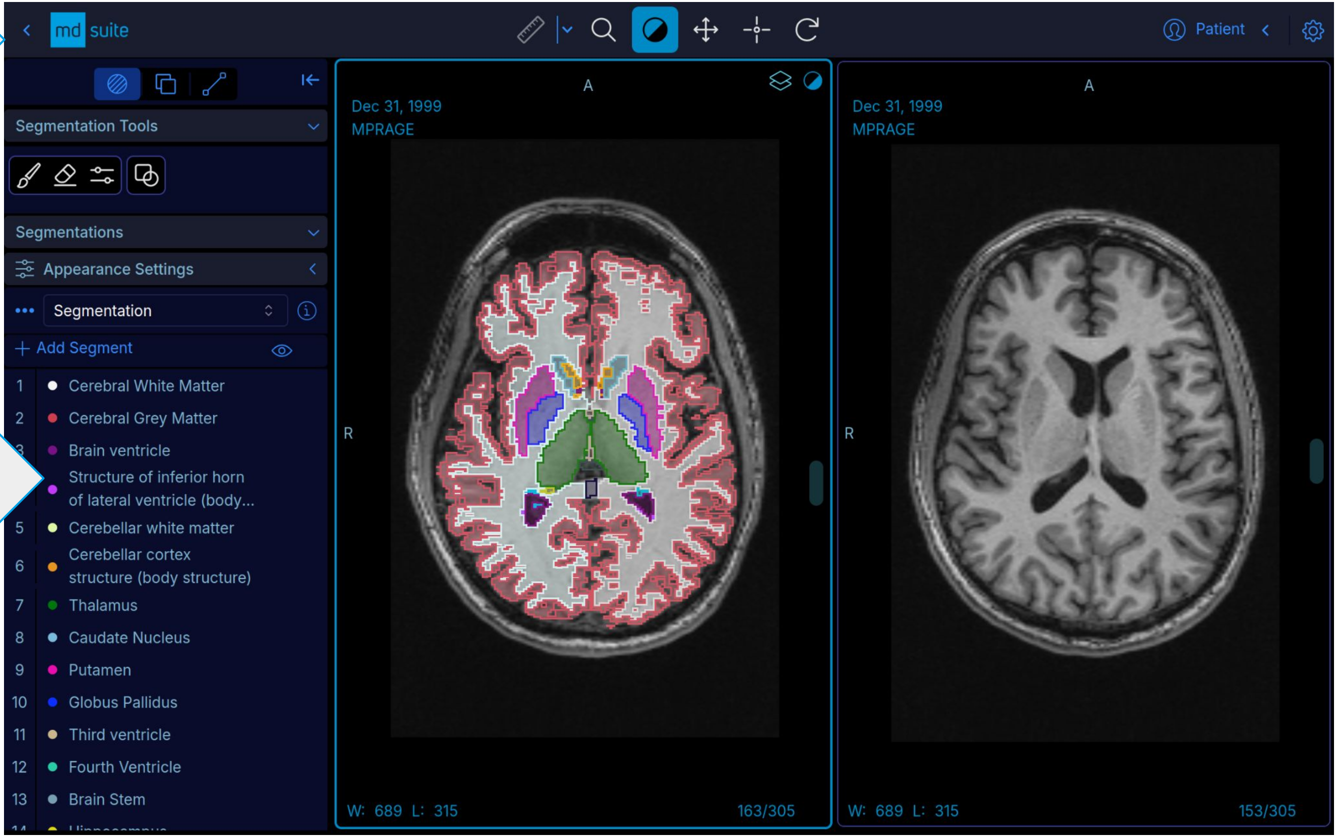
Reflection Questions

- How easy/difficult was the refactoring?
- Personally, what are the main benefits of Sass?
- When would you choose Sass over pure CSS? Why?
- Did you learn anything we can apply for our projects?

Part III

Utility-first & Tailwind CSS

Business Case: Viewer Rewrite



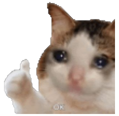
1 Cerebral White Matter
2 Cerebral Grey Matter
3 Brain ventricle
4 Structure of inferior horn of lateral ventricle (body...
5 Cerebellar white matter
6 Cerebellar cortex structure (body structure)
7 Thalamus
8 Caudate Nucleus
9 Putamen
10 Globus Pallidus
11 Third ventricle
12 Fourth Ventricle
13 Brain Stem
14 Hippocampus

How we usually write CSS

Semantic CSS: define HTML and CSS based on element structure & meaning

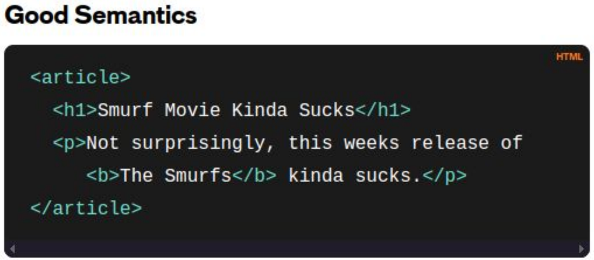
Separate files for structure and styles

Separation of concerns, maintainability, collaboration



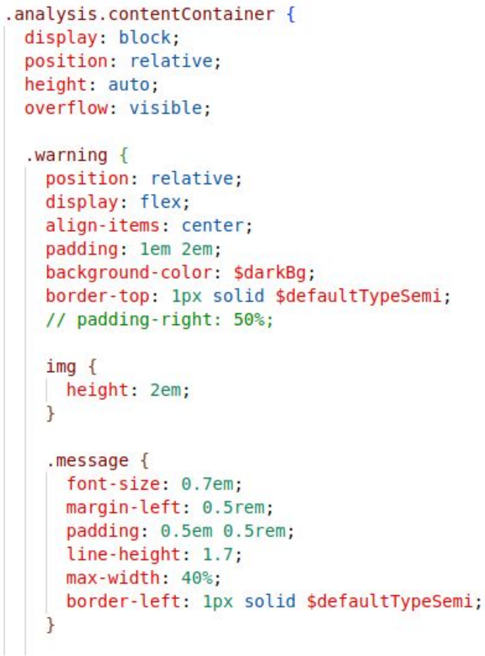
Problems

- Lacks scalability, unwanted side-effects (global scope)
- Naming conventions required
- No real separation between HTML and CSS
- New styles -> more CSS (re-using requires effort)
- Slower development from scratch



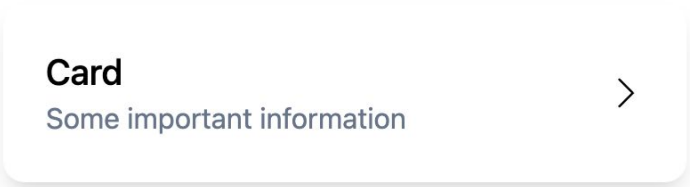
Good Semantics

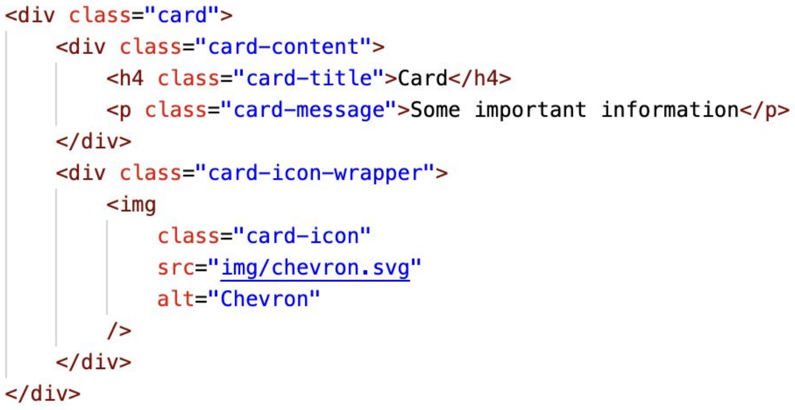
```html
<article>
  <h1>Smurf Movie Kinda Sucks</h1>
  <p>Not surprisingly, this weeks release of
     <b>The Smurfs</b> kinda sucks.</p>
</article>
```



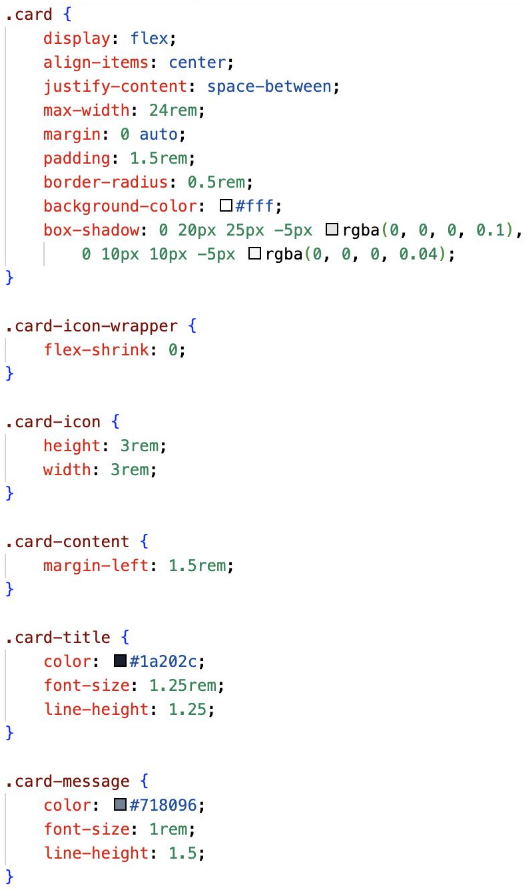
```css
.analysis.contentContainer {
  display: block;
  position: relative;
  height: auto;
  overflow: visible;

  .warning {
    position: relative;
    display: flex;
    align-items: center;
    padding: 1em 2em;
    background-color: $darkBg;
    border-top: 1px solid $defaultTypeSemi;
    // padding-right: 50%;

    img {
      height: 2em;
    }

    .message {
      font-size: 0.7em;
      margin-left: 0.5rem;
      padding: 0.5em 0.5rem;
      line-height: 1.7;
      max-width: 40%;
      border-left: 1px solid $defaultTypeSemi;
    }
```

Traditional Workflow





```html
<div class="card">
    <div class="card-content">
        <h4 class="card-title">Card</h4>
        <p class="card-message">Some important information</p>
    </div>
    <div class="card-icon-wrapper">
        <img
            class="card-icon"
            src="img/chevron.svg"
            alt="Chevron"
        />
    </div>
</div>
```



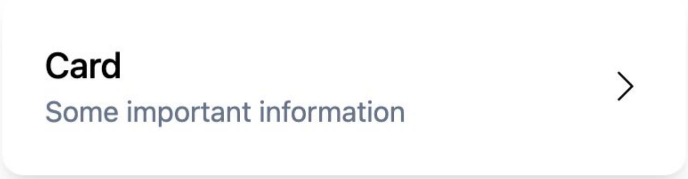
```css
.card {
    display: flex;
    align-items: center;
    justify-content: space-between;
    max-width: 24rem;
    margin: 0 auto;
    padding: 1.5rem;
    border-radius: 0.5rem;
    background-color: □#fff;
    box-shadow: 0 20px 25px -5px □rgba(0, 0, 0, 0.1),
        0 10px 10px -5px □rgba(0, 0, 0, 0.04);
}

.card-icon-wrapper {
    flex-shrink: 0;
}

.card-icon {
    height: 3rem;
    width: 3rem;
}

.card-content {
    margin-left: 1.5rem;
}

.card-title {
    color: ■#1a202c;
    font-size: 1.25rem;
    line-height: 1.25;
}

.card-message {
    color: ■#718096;
    font-size: 1rem;
    line-height: 1.5;
}
```

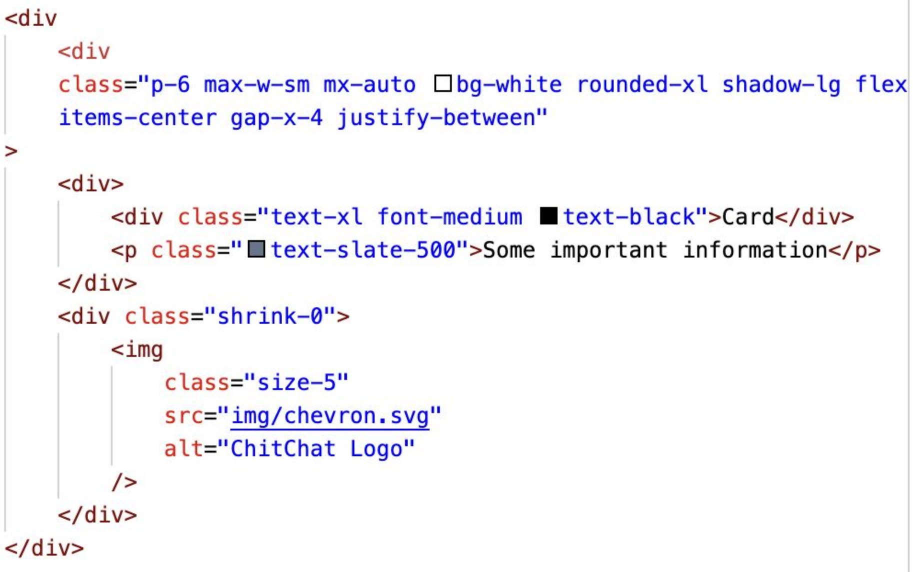
Alternative: Utility-first Approach

Idea: don't write CSS; instead apply atomic utility classes to HTML

Atomic: each class represents a single CSS property

No custom CSS needed for most solutions!





```
<div
    <div
    class="p-6 max-w-sm mx-auto □bg-white rounded-xl shadow-lg flex
    items-center gap-x-4 justify-between"
>
    <div>
        <div class="text-xl font-medium ■text-black">Card</div>
        <p class="■text-slate-500">Some important information</p>
    </div>
    <div class="shrink-0">
        <img
            class="size-5"
            src="img/chevron.svg"
            alt="ChitChat Logo"
        />
    </div>
</div>
```

connect
KEYNOTE 2023

Tailwind CSS

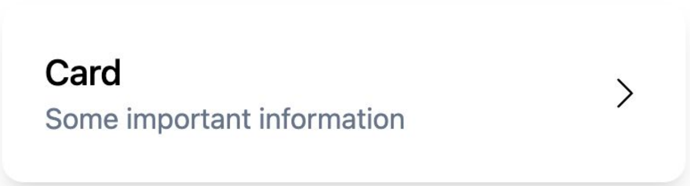
Utility-first CSS framework: pre-defined utility-classes

Idea: Instead of writing CSS, you use the classes like building blocks

Benefits

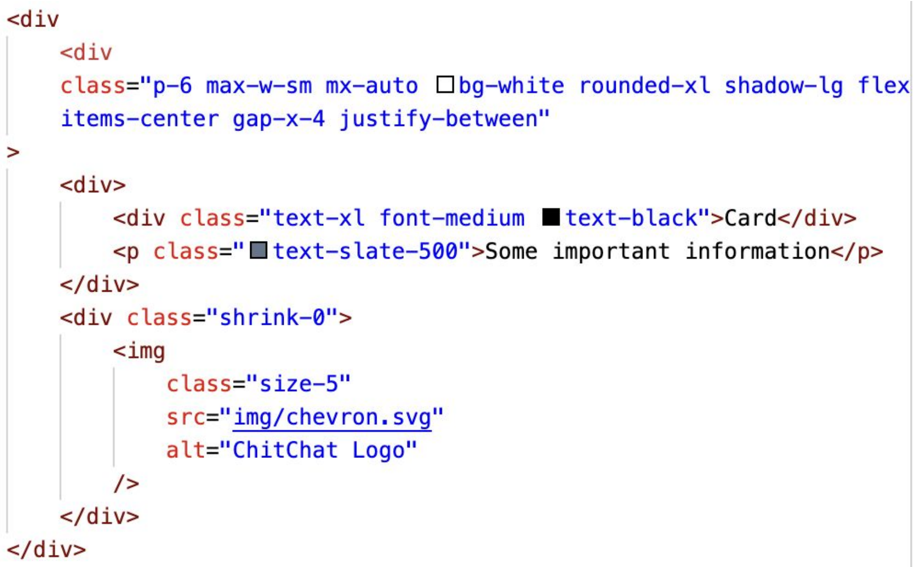
- Pre-defined class names: no name guessing + consistency
- Maintainability: no more growing CSS
- No global changes on local edits (remember Peter Griffin)
- Most popular CSS framework (used by Netflix, OpenAI, Shopify, …)
- Customizable: allows to change colors or overwrite/define classes
- UI libraries and toolkits ready to be used or draw inspiration from (TailwindUI, shadcn/ui)

Tailwind CSS: Example



Card

Some important information

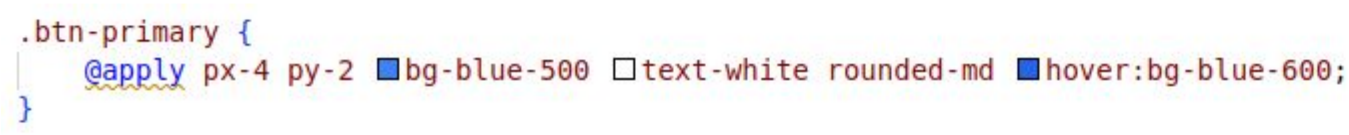


```
<div
    <div
    class="p-6 max-w-sm mx-auto □bg-white rounded-xl shadow-lg flex
    items-center gap-x-4 justify-between"
>
    <div>
        <div class="text-xl font-medium ■text-black">Card</div>
        <p class="■text-slate-500">Some important information</p>
    </div>
    <div class="shrink-0">
        <img
            class="size-5"
            src="img/chevron.svg"
            alt="ChitChat Logo"
        />
    </div>
</div>
```

Tailwind CSS: @apply

@apply allows to combine different utility-classes into one

→ reuse common patterns



```css
.btn-primary {
    @apply px-4 py-2 bg-blue-500 text-white rounded-md hover:bg-blue-600;
}
```

However: you need to write CSS again…

Task 3
Tailwind CSS

task3-tailwindcss

(~20 min)

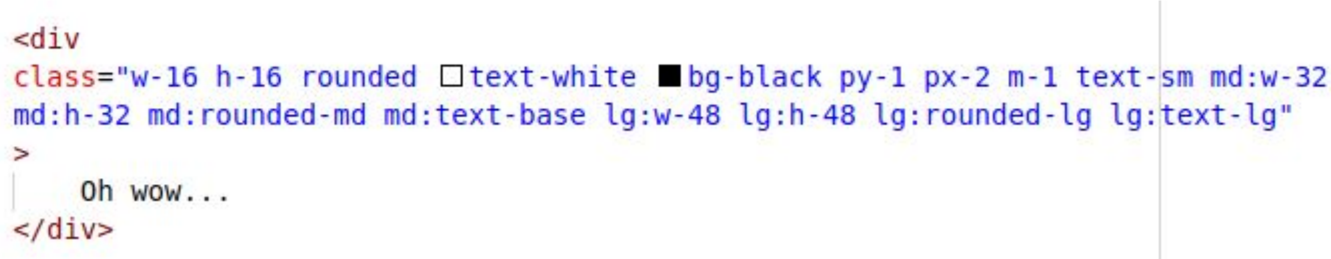
Task 3
Tailwind CSS

Reflection:

- How was the experience of styling using utility classes instead of CSS rules?
- Why do you think Tailwind is so popular?
- What were drawbacks you noticed or what can you imagine when the code grows?
- Could you imagine using it in our projects?

Tailwind CSS Drawbacks

- Cluttered HTML / difficult to read
- Learning curve (abbreviated class names like `mt-3`)
- Adds complexity compared to CSS (check which classes are printed in the final bundle)



```
<div
class="w-16 h-16 rounded □text-white ■bg-black py-1 px-2 m-1 text-sm md:w-32
md:h-32 md:rounded-md md:text-base lg:w-48 lg:h-48 lg:rounded-lg lg:text-lg"
>
    Oh wow...
</div>
```

However: This fosters using components / @apply to encapsulate reusable parts as much as possible.

Tooling exists to help organize the classes.

Part IV
Discussion & Conclusion

Discussion

- What were your main learnings today?
- Do you have any ideas how / where we improve our current styles?
  - Web Interface tables (party done via SCSS, and partly copied from some JS file as inline styles)
  - In general: consolidate duplicate styles → reduce bundle size
- What paradigm & framework should we use for the future?
- What are the most important factors for you personally when it comes to styling?
- Do you have further suggestions how we can improve our styles (maintainability & UX)?
  - Mediaire UI Library (next slide)

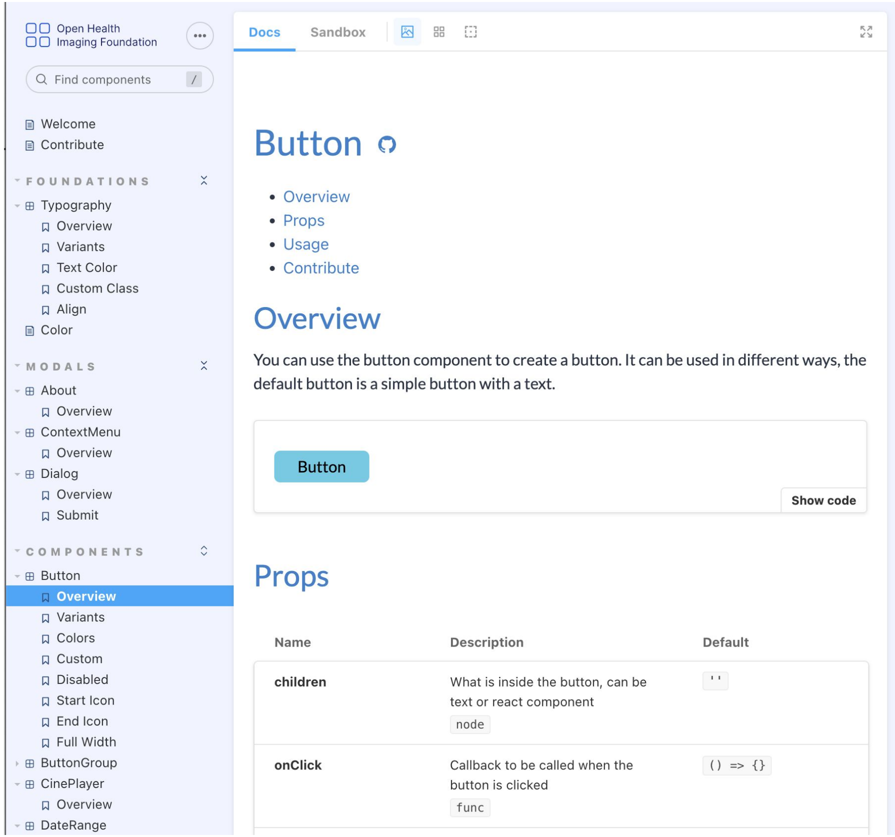
Idea Pitch: Mediaire UI Library

- Extract commonly used components & styles into a shared repo
- Use TailwindCSS for quick implementation & high maintainability
- Document components using Storybook
  - “Single source of truth” for our company styles
  - easy to browse components, colors, typography
  - collaborative tool for designers, devs, QA (test in isolation)

→ Collection of re-usable components that are pre-styled according to our mediaire brand (e.g. buttons, nav bar, modal dialogs, …)

→ **no duplicate styles, quick development of new features or products**

Example: OHIF UI



Button

- Overview
- Props
- Usage
- Contribute

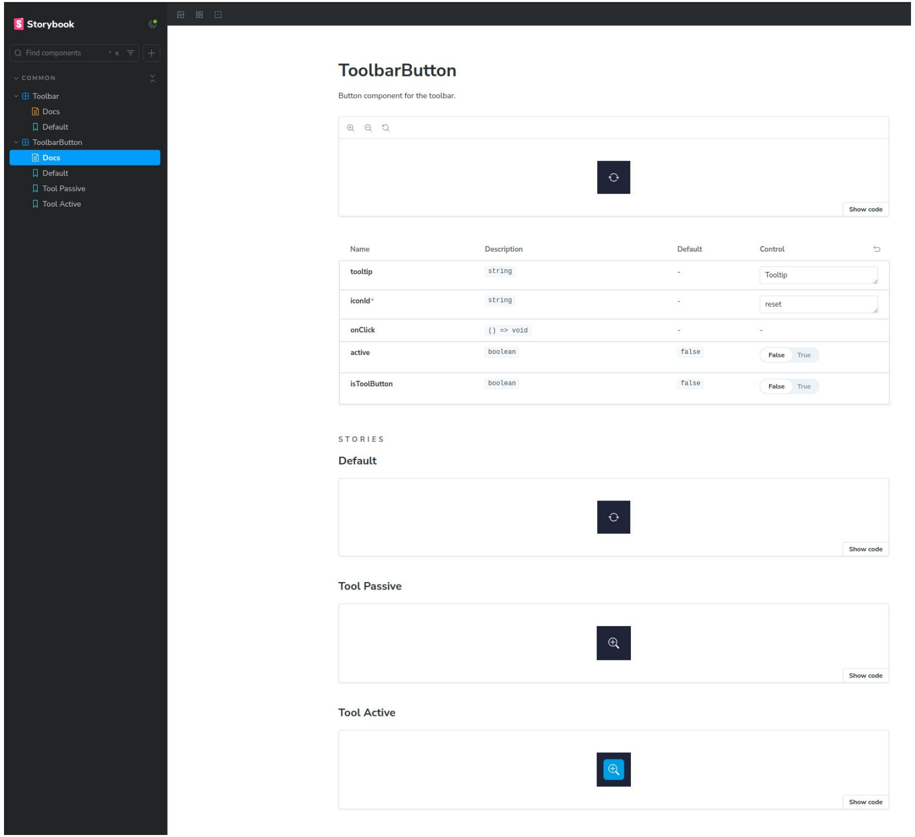
Overview

You can use the button component to create a button. It can be used in different ways, the default button is a simple button with a text.

Props

Name	Description	Default
children	What is inside the button, can be text or react component `node`	`''`
onClick	Callback to be called when the button is clicked `func`	`() => {}`

First Draft: Viewer Components



ToolbarButton

Button component for the toolbar.

Name	Description	Default	Control
tooltip	`string`	-	Tooltip
iconId*	`string`	-	reset
onClick	`() => void`	-	-
active	`boolean`	`false`	False True
isToolButton	`boolean`	`false`	False True

STORIES

Default

Tool Passive

Tool Active

Conclusion

- Basic CSS solves a lot of problems with reduced complexity
- Frameworks / preprocessors should have a clear use case
- Two major paradigms of CSS
    - **Semantic CSS**: great for readability, collaboration, and long-term maintainability in simple or medium-complexity projects
    - **Utility-first CSS:** excels in fast prototyping, scalability, and consistency, making it ideal for larger or fast-paced projects

Next steps:

- First draft of mediaire component library (based on viewer components)
- Tailwind CSS: great opportunity for our projects to have more maintainable styles and code in general

Q&A / Feedback

I’m happy to hear your opinions on this workshop!
Was it helpful? Did you like the format? …

Sources

1. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Getting_started
2. https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Styling_basics/Handling_conflicts
3. https://sass-lang.com/guide/
4. https://css-tricks.com/semantic-class-names/
5. https://www.infoq.com/articles/no-need-css-framework/
6. https://adamwathan.me/css-utility-classes-and-separation-of-concerns/
7. https://www.aleksandrhovhannisyan.com/blog/why-i-dont-like-tailwind-css
8. https://eev.ee/blog/2020/02/01/old-css-new-css/
9. https://raygun.com/blog/css-preprocessors-examples/