

Styling

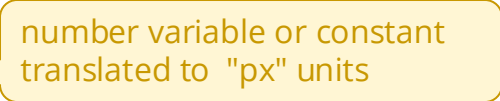
- CSS Style for Components
- Tailwind

Approaches to CSS Style for Components

- Separate CSS file for each component
(and be careful about how you choose element ids and selectors)
- Inline CSS attributes in component render
- Import *component specific* CSS module file
- Utility classes (e.g. using Tailwind)
- (for React) styled-components package

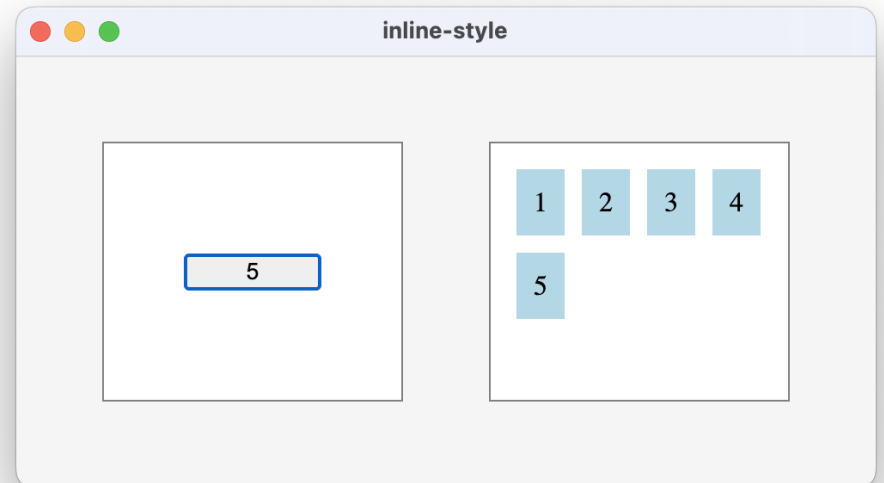
not covered
in lecture

Inline CSS

- Style attribute accepts a JavaScript object to specify CSS properties
 - e.g.
 - `"padding: 10px" → { padding: "10px" }`
 - `"padding: 10px" → { padding: 10 }` 
 - `"flex: 1 1 auto" → { flex: "1 1 auto" }`
- JavaScript variable names can't have "-" character, so CSS properties with "-" are written in camelCase
 - e.g.
 - `"flex-flow: row nowrap" → { flexFlow: "row nowrap" }`
- Convention is to create const object and assign in render
- Inline styles do not provide flexibility with rules, selectors, etc.

style-inline

- Using inline styles for counter app
- Note still using global css for "reset"



CSS Modules

- CSS file with ".module.css" extension
 - convention is to name like "MyComponent.module.css"
- import module.css into the component tsx file

```
import style from './MyComponent.module.css';
```

- Assign classes in render

```
<div class={style.root}>
```

- Class names are locally scoped to component

this is pretty
amazing

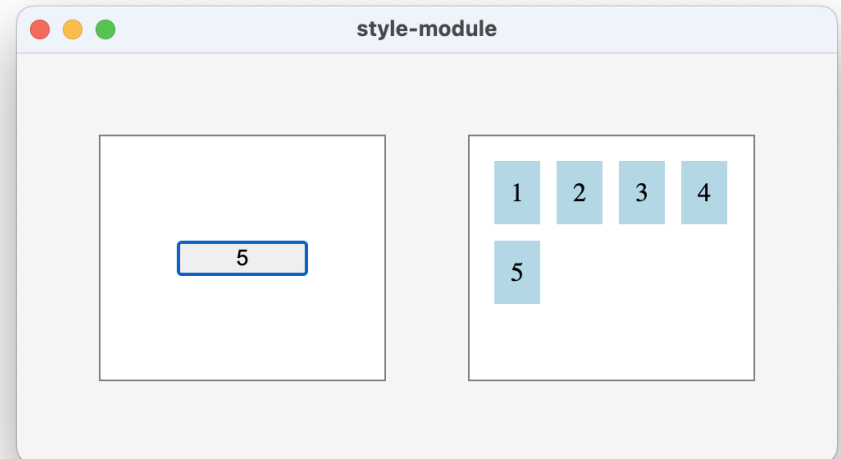
- unique class names are generated, e.g.

```
<div class="root_dsfsdf_1">
```

- try inspecting styles in DevTools

style-module

- Using style modules for counter app
- Note how styles classes are local to component
- Note, still global reset style.css



Tailwind

- A utility-first CSS framework
- class names are like style properties (called "utility classes")
- No need for media queries, use a size prefix instead
- Same strategy for hover, focus, etc.
- bundler removes all unused CSS for production
 - most Tailwind projects ship less than 10kB of CSS to the client
- Approach works best with declarative UI programming
 - e.g. components in Preact

“Best practices” don’t actually work.

I’ve written **a few thousand words** on why traditional “semantic class names” are the reason CSS is hard to maintain, but the truth is you’re never going to believe me until you actually try it. If you can suppress the urge to retch long enough to give it a chance, I really think you’ll wonder how you ever worked with CSS any other way.



Adam Wathan

Creator of Tailwind CSS

Tailwind Setup

1. Follow “Tailwind with Vite” installation steps:

- <https://tailwindcss.com/docs/guides/vite>
- Needed for compiling Tailwind CSS

2. Install official Tailwind VS Code plug-in:

- <https://tailwindcss.com/docs/editor-setup#intelli-sense-for-vs-code>
- Tailwind autocomplete, linting, hover preview of actual CSS

Tailwind Philosophy

No predesigned components like buttons, cards, alerts, etc.

- other CSS libraries (like Bootstrap) provide predesigned components that you "fight to override"
- Tailwind uses a pretty extreme CSS reset (called "preflight")
<https://tailwindcss.com/docs/preflight>
- **Implication:** you need to add basic styling for *everything*
- Can disable the Tailwind CSS reset and use browser defaults
 - <https://tailwindcss.com/docs/preflight#disabling-preflight>

Example of Basic Styling with Tailwind

- Default style

```
<button>Button</button>
```

renders:

Button

just clickable text

- Style with *utility classes*

```
<button class="py-0.5 px-2 bg-gray-200  
border border-gray-600 rounded  
hover:bg-gray-300 active:bg-gray-200"  
>Button</button>
```

renders:

Button

- Style with *inline utility classes in standard CSS rule*

- in style.css:

usually in **@layer base** (a Tailwind defined CS layer)

```
button {  
  @apply py-0.5 px-2 bg-gray-200  
  border border-gray-600 rounded  
  hover:bg-gray-300 active:bg-gray-200;}  
then:
```

```
<button>Button</button>
```

renders:

Button

style-tailwind-test

- Test Tailwind in a simple html file
- Setup: must add html files to types for processing in tailwind.config.js

```
content: [ "./index.html",  
            "./src/**/*.{js,ts,jsx,tsx,html}" ],
```
- Demos:
 - Experiment with inline class and @apply in style.css
 - Button (including hover, active, disabled)
 - Padding values, including custom unit like p-[10px]
 - Screen sizes (next slide)

Tailwind is Mobile-first Responsive

- Provides a {screen:} prefix to use instead of media queries/rules
- Screen prefix identifies *when* to use utility class
 - default screen is mobile (no prefix)



- Example:

```
<div class="... flex flex-col sm:flex-row ...">
```

column for mobile

row for tablet and larger

style-tailwind

Counter demo styled with Tailwind

- Button style component class in "styles.css"

- Responsive screen prefix

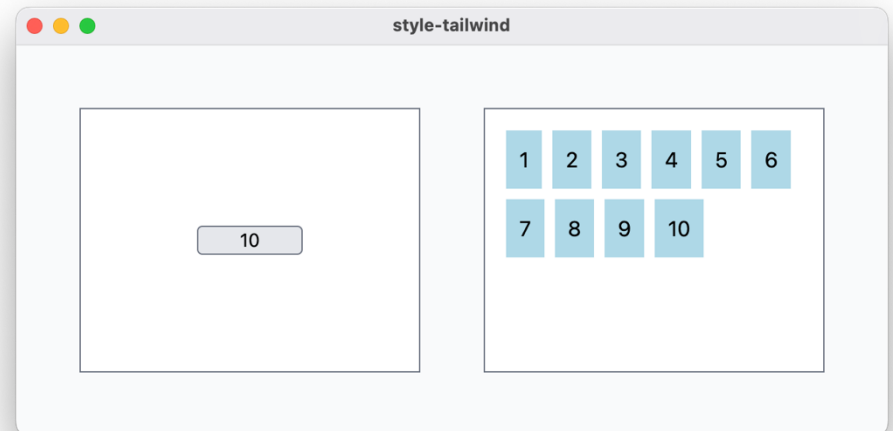
```
<div class="... flex flex-col sm:flex-row ...">
```

- Set custom value

```
<div class="... p-[10px] ...">
```

- Pseudo class prefix

```
<div class="... text-black hover:text-red-500 ...">
```



Tailwind Plug-ins

- Plug-ins let you register new styles for Tailwind to inject into the user's stylesheet (using JavaScript instead of CSS)
- **Typography** plug-in
 - <https://github.com/tailwindlabs/tailwindcss-typography>
 - Adds `prose` class, mainly for styling text like `h1`, `p`, `quote`, etc.
- **Forms** plug-in
 - <https://github.com/tailwindlabs/tailwindcss-forms>

Check assignment specification,
using `npm` to install Tailwind plug-ins may not be allowed.



Tailwind CSS is the worst...

spoiler: it isn't the worst

- <https://youtu.be/IHZwlzOUOZ4?si=TQyNWJkxIBxghyss>



How Tailwind CSS came to be feat. Adam Wathan
- https://youtu.be/1x7HlvSfW6s?si=tQ7bEsqDM-Fs7_aK

todo with Tailwind



- Install TailwindCSS and use it to style the todo app from last week's lecture
 - Only a single style.css needed (don't forget to add 3 @tailwind declarations)
 - Suggest defining a standard button style using @apply
 - Strike through text in TodoItem is best handled with declarative rendering of "line-through" class
 - slate colour looks nicer than gray
 - I used built-in tailwind sizes since it was faster for this demo (e.g. p-8 gives about 8px padding, rather than p-[10px] to get exactly 10px padding)

