

# Practical CSS for People who Avoids CSS

@albertalrisa

SODA Inc. 勉強会

Aug 4th, 2025

# Who is this session for?

- People who are not used to (or avoids) CSS
- Experienced FE devs might find how this slide is constructed more interesting

This session will be in **English** but a Japanese slide note will be provided.

(translated by  Gemini)

## Speaker notes

This session is aimed at people who are not used to CSS. For front-end developers, you might be already familiar with a lot of these CSS features.

However, for experienced front-end developers, you might be more interested in how this slide is constructed. This slide is fully written in HTML, CSS, and JS, using reveal.js with the addition of Tailwind and Lit for web components. Whether adding those libraries is a good idea remains to be seen. But it works.

このセッションは、CSSにあまり慣れていない方を対象としています。フロントエンド開発者の方にこつては、すでにご存知の内容が多く含まれているかもしません。

しかし、経験豊富なフロントエンド開発者の方であれば、むしろこのスライドが“どうのよう”に作られているかの方にご興味があるかもしません。このスライドは全体がHTML、CSS、JSで書かれており、reveal.jsをベースに、WebコンポーネントのためにTailwindとLitを追加しています。

これらのライブラリを追加するのが良いアイデアだったかはさておき、まあ、ちゃんと動いています。

# Layoutting

# Grid

Baseline  
widely available



# Grid template

```
.container {  
  display: grid;  
  grid-template-columns: repeat(4, 1fr);  
  grid-template-rows: 50px 1fr;  
}
```



# Column and row size

Flex

minmax

1fr

repeat(3, 10px)

minmax(100px, 1fr)

Repeat

Implicit size

auto

repeat(3, 10px)

## Speaker notes

Flex ('fr') allows us to define a flexible length in a grid container. The 'fr' unit represents a fraction of the leftover space. For example, a '3fr' div and '2fr' div placed in a 800px container will split the container to '3/5' (480px) and '2/5' (320px).

'minmax' allows us to specify the minimum and maximum size of a cell. 'minmax' works with both fixed and flexible size.

'repeat' allows us to define the same length multiple times.

Lastly, 'auto' will set the width automatically to fit the size of the content.  
fr単位は、グリッドコンテナー内でフレキシブルな（可変の）長さを定義することを可能にします。この単位は、利用可能な残りのスペースに対する割合を表します。例えば、幅800pxのコンテナーに3frと2frのdivを配置すると、コンテナーは 3/5 (480px) と 2/5 (320px) の割合で分割されます。  
minmax()関数は、セルの最小サイズと最大サイズを指定することを可能にします。minmax()は、固定サイズと可変サイズの両方で機能します。

repeat()関数は、同じ長さを複数回繰り返して定義することを可能にします。  
最後に、autoはコンテナのサイズに合わせて幅を自動的に設定します。

# Let's try!

A      B      C      D      E      F



## STYLES

```
.example {  
  display: grid;  
  grid-template-columns: auto,  
}
```

## Speaker notes

Using those sizing, we can easily layout multiple items into a neatly organized grid. Just like with flexbox, we can define the gap between the column and row.

```
.example {  
  display: grid;  
  grid-template-columns: 100px 1fr auto;  
  column-gap: 0.5rem;  
  row-gap: 0.25rem;  
}
```

While this is a more advanced use case, grid allows us to create more advanced layouts by using `grid-template-areas` and `grid-area` .

```
.example {  
  display: grid;  
  grid-template-columns: repeat(3, 1fr);  
  grid-template-rows: repeat(3, 1fr);  
  grid-template-areas:  
    "a b"  
    "c b"  
    "c d"  
  ;  
}  
  
.a {  
  grid-area: a;  
}  
.b {  
  grid-area: b;  
}  
.c {  
  grid-area: c;  
}  
.d {  
  grid-area: d;  
}  
  
.e, .f {  
  display: none;  
}
```

Grid allows us to overlay multiple elements in the same grid cell. If multiple elements are assigned the same grid-area, it will be overlaid on top of each others.

```
.example {  
  display: grid;
```

```
grid-template-columns: 1fr;
grid-template-rows: 1fr;
width: 200px;
height: 200px;
background: white;
}

.a, .b, .c, .d, .e, .f {
  opacity: 0.6;
  grid-area: 1 / 1;
}
```

We can use this method to center element in a container!

```
.a, .b, .c, .d, .e, .f {
  display: none;
}

.a {
  display: block;
  padding: 1rem;
  grid-area: 1 / 1;
  place-self: center;
}
```

これらのサイジング方法を使うと、複数のアイテムをきれいに整列したグリッドへ簡単にレイアウトできます。flexboxと同様に、column-gapやrow-gapで行と列の間隔を定義できます。

```
.example {
  display: grid;
  grid-template-columns: 100px 1fr auto;
  column-gap: 0.5rem;
  row-gap: 0.25rem;
}
```

これはより高度な使い方ですが、grid-template-areasとgrid-areaを作成できます。

```
.example {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 1fr);
  grid-template-areas:
    "a b c"
    "c b d"
    "a b d";
}

.a {
  grid-area: a;
}

.b {
  grid-area: b;
```

```
.c {  
  grid-area: c;  
}  
  
.d {  
  grid-area: d;  
}  
  
.e, .f {  
  display: none;  
}
```

Gridでは、同じセルに複数の要素を重ねて配置できます。複数の要素に同じgrid-areaを割り当てるごと、それらの要素は互いに重なり合います。

```
.example {  
  display: grid;  
  grid-template-columns: 1fr;  
  grid-template-rows: 1fr;  
  width: 200px;  
  height: 200px;  
  background: white;  
}  
  
.a, .b, .c, .d, .e, .f {  
  opacity: 0.6;  
  grid-area: 1 / 1;  
}
```

このテクニックを応用すれば、コンテナー内で要素を中央揃えにできます！

```
.a, .b, .c, .d, .e, .f {  
  display: none;  
}  
  
.a {  
  display: block;  
  padding: 1rem;  
  grid-area: 1 / 1;  
  place-self: center;  
}
```

**What about flexbox?**

Use both!

## Speaker notes

Grid and flexbox serve similar but different purpose. For a lot of cases, flexbox is enough. Just like grid, flexbox also allows flexible sizing.

The main differentiator between grid and flexbox is that with grid, you have a lot more control to the sizing and positioning. Flexbox works great if you need to manage items in one dimension. Grid makes it easier to align items in two dimensions. For example, you can align multiple rows of items using `grid-template-columns` . This is not possible with flexbox unless you specify explicit size of each cell. With grid, you can set the column size dynamically based on the content.

Grid also allows you to overlap elements. This is not possible with just flexbox. This overlap feature allows you to replace things like absolute positioning and it follows the alignment of the grid.

Gridとflexboxは、似ていますが異なる目的を果たします。多くの場合、flexboxだけでは十分です。Gridと同様に、flexboxもフレキシブルなサイズング（可変長）が可能です。  
Gridとflexboxの主な違いは、Gridの方がサイズングと配置に対しても、より多くの制御ができる点です。flexboxは、一次元（一方向）でアイテムを管理する場合に非常にうまく機能します。Gridは、二次元（縦横）でアイテムを整列させることうを容易にします。例えば、grid-template-columnsを使えば、複数行にわたるアイテムの整列が可能です。これは、各セルのサイズを明示的に指定しない限り、flexboxでは困難です。Gridなら、コントラストに基づいて列のサイズを動的に設定できます。

また、Gridでは要素を重ねることもできます。これはflexboxだけでは不可能です。この重ね合わせ機能により、position: absolute のような指定を代替でき、かつGridの配置ルールに従わせることができます。

# @container

Baseline  
from 2023



# You might be familiar with @media query

```
1 .product-list {  
2   display: grid;  
3   grid-template-columns: repeat(3, 1fr);  
4 }  
5  
6 @media (min-width: 768px) {  
7   .product-list {  
8     grid-template-columns: repeat(5, 1fr);  
9   }  
10 }
```

# Now we can apply it to any container!

```
1 .product-info {  
2   container: inline-size;  
3  
4   .name {  
5     font-size: 1.5rem;  
6   }  
7 }  
8  
9 @container (width > 512px) {  
10  .name {  
11    font-size: 2rem;  
12  }  
13 }
```

We can create something like  
this

 Running

**GEL-KAYANO 32**

Asics

Size Info	US	JP	EU	UK
	8	26cm	41.5	7

¥ 20,000 22,000



**Does this replace @media  
(or why not just use @media)?**

Again, use both! They serve different purposes.

## Speaker notes

The @container is not replacing @media query. They serve different purposes. When you want to adjust certain properties based on the viewport, use @media query.

@container query is useful when you want a certain part of the site to be displayed differently depending on the size of that component (or in an advanced usecase, other container). This allows you to design a component that works in multiple sizes, regardless of the viewport size.

@container has @media query to置き換えるものではありません。両者は異なる目的を持っています。ビューポートに基づいて特定のプロパティを調整したい場合は、@media queryを使用します。

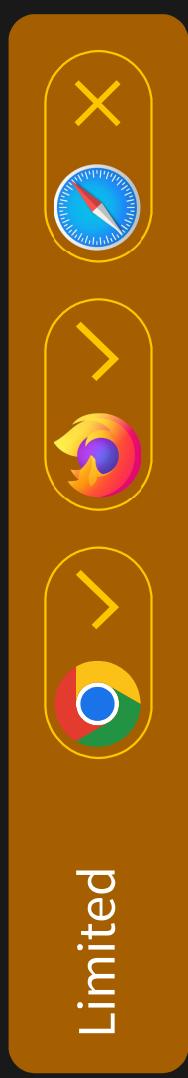
@container queryは、サイトの特定の部分を、そのコンポーネント自体のサイズに応じて表示を変えたい場合に役立ちます。これにより、ビューポートのサイズに関係なく、複数のサイズで機能するコンポーネントを設計できます。

# Utilities

## Speaker notes

Next, we're doing to look at some simple utility functions that we can use in a lot of places.  
次に、様々な場面で使えるシンプルなユーティリティ関数をいくつか見ていきましょう。

**User-select: none**



## Speaker notes

Note that this feature is not yet a baseline feature, even though support for Chromium and Firefox-based browsers are already there since 2020 (unprefixed)! Chrome and Firefox had it (prefixed) since version 1 back on 2008 and 2004 respectively.

On Safari, we still need to use the `webkit-user-select: none` prefix even on the latest version, and unfortunately it doesn't work on iOS webview.

注意点として、この機能はまだベースライン機能ではありません。Chromium系やFirefox系のブラウザでは2020年から（プレフィックスなしで）サポートされているにもかかわらずです！ちなみに、Chromeでは2008年のバージョン1から、Firefoxでは2004年のバージョン1から（プレフィックス付きで）利用可能でした。

Safariでは、最新バージョンであっても依然として -webkit-user-select: none というプレフィックスを使用する必要があります。そして残念なことに、iOSのwebViewでは機能しません。

Sometimes, these happens



## Speaker notes

Often when we're creating some kind of components or decorative elements, we don't want the user to be able to select the text on that element. For example, users are not supposed to be able to select the text displayed on a button. コンポーネントや装飾用の要素を作成する際、その要素上のテキストをユーザーに選択させたくないケースがよくあります。例えば、ボタンに表示されているテキストは、通常ユーザーが選択できるべきではありません。

# user-select: none to the rescue

```
1 button {  
2   user-select: none;  
3 }  
4   .product {  
5     /* ... */  
6   }  
7   .banner {  
8     user-select: none;  
9   }  
10  }  
11 }  
12 }  
13 .header {  
14   user-select: none;  
15 }
```

However...

**Don't abuse it**

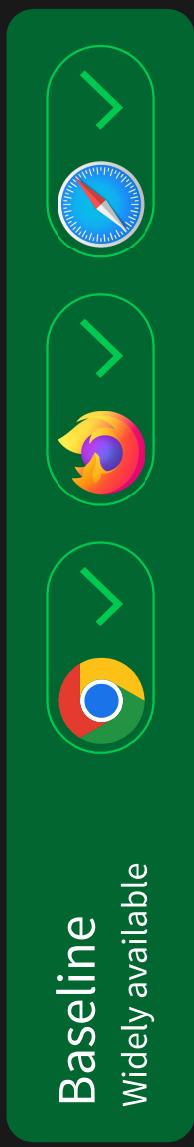
## Speaker notes

`user-select: none` disables user selection. While it's okay to apply this to interactive or decorative elements like buttons or overlays, it's not recommended to apply this to text or other sections where users might want to select, copy, or input.

While some people use this to make web app look more native, this is not a good practice for accessibility and user experience.

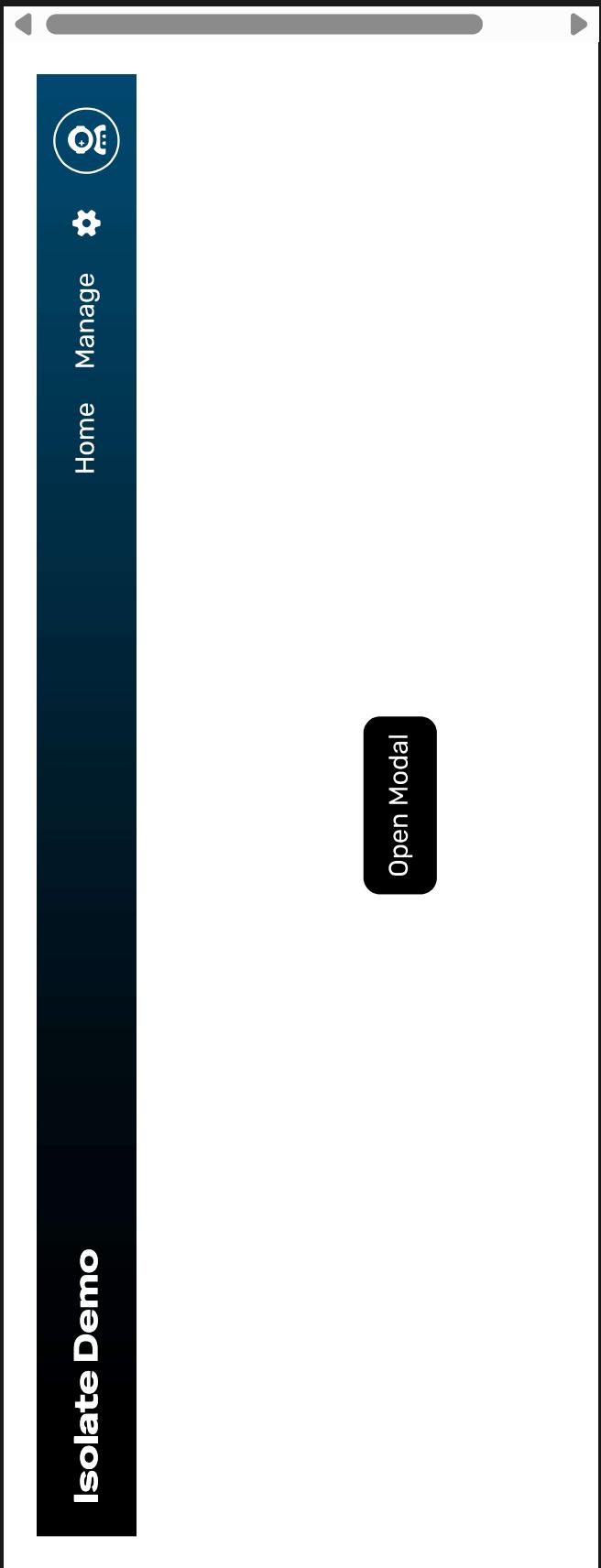
`user-select: none`は、ユーザーによるテキスト選択を無効にします。ボタンやオーバーレイといったインターラクティブな要素や装飾用の要素に適用するのは問題ありませんが、ユーザーが選択やコピーをしたいと思うようなテキスト部分に適用することは推奨されません。  
Webアプリをよりネイティブアプリらしく見せるためにこのプロパティを使う人もいますが、アクセシビリティやユーザーエクスペリエンスの観点からは、良いプラクティスとは言えません。

# isolation: isolate



Baseline  
widely available

**Have you ever experienced  
this?**



## Speaker notes

To build a modal, most of the time we'll rely on both `position: fixed` with `z-index` to help place it on top of the other elements. Not only that, most of the time, the modal itself will have a backdrop/barrier and dialog, which needs a separate z-index.

However, if we use `z-index` on other part of the site, sometimes we can find elements showing on top of the modal, which is not ideal. The solution would be to track which z-index is used and assign a specific value for a specific usage. This means maintaining list of which z-index is allowed per component type.

Or, we can use `isolation: isolate`.

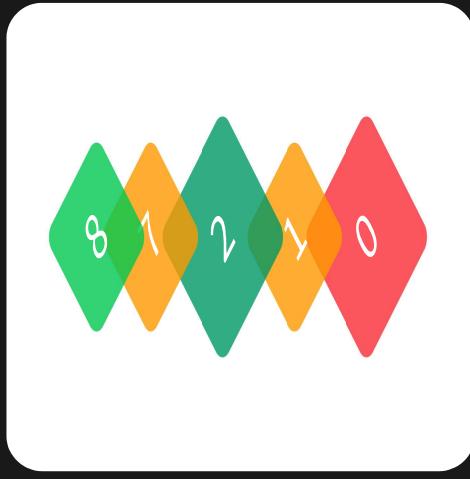
モーダルを実装する際、他の要素の上に配置するために、多くの場合 position: fixed と z-index に依存します。それだけではなく、モーダル自体が背景のオーバーレイ (backdrop) とダイアログ本体で構成されることが多く、それに z-index が必要になります。

しかし、サイトの他の部分で z-index を使用していると、時々 モーダルの上に他の要素が表示されてしまうことがあります。これは望ましくありません。解決策としては、どの z-index が使われているかを管理し、特定の用途に特定の値を割り当てることになります。これは、コンポーネントの種類ごとに、許可される z-index のリストを保守・管理することを意味します。

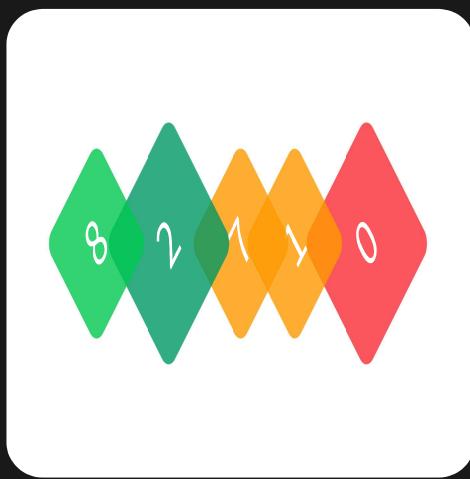
あるいは、isolation: isolate を使う、という手もあります。

# Isolate helps solve this

Without isolate



With isolate



## Speaker notes

By defaults, all elements are placed within the same context. This is what caused the elements to be overlaid on top of other elements depending on the z-index. However, most of the time we want the z-index to work only on a certain group. To solve this issue with z-index, we can use the 'isolate' property.

When we use 'isolate' on a parent element, the children will be put in a separate stacking context. Any z-index value are valid only within that context. This lets each element to have any z-index value but it will not affect other elements in the parent or other stacking contexts.

デフォルトでは、全ての要素は同じスタッキングコンテキスト (stacking context) 内に配置されます。これが、z-indexの値に応じて要素が他の要素の上に重なって表示されてしまう原因です。しかし、多くの場合z-indexは特定のグループ内でのみ機能してほしいものです。この問題を解決するために、isolation: isolateが使えます。  
親要素にisolation: isolateを使用すると、その子要素は独立したスタッキングコンテキストに配置されます。z-indexの値は、そのコンテキスト内でのみ有効になります。これにより、各要素がどのようにz-index値を持つしていくのも、親や他のスタッキングコンテキストにある要素には影響を与えなくなります。

# Applying isolate is easy

```
<main class="body" style="z-index: 0">  
  <div style="z-index: 1">1</div>  
  <div style="z-index: 7">7</div>  
</main>
```

```
<div class="modal" style="z-index: 2">  
  <div style="z-index: 8">8</div>  
</div>
```



```
<main class="body" style="z-index: 0; isolation: isolate;">  
  <div style="z-index: 1">1</div>  
  <div style="z-index: 7">7</div>  
</main>
```

```
<div class="modal" style="z-index: 2; isolation: isolate;">  
  <div style="z-index: 8">8</div>  
</div>
```

# Nested CSS

Baseline  
from 2023



# You might be familiar with this in SCSS

```
.button {  
  translate: 0 var(--translate-y, -5px);  
  transition: translate 100ms;  
  padding: 0.25rem 1rem;  
  border-radius: 0.25rem;  
  
  &:hover {  
    -translate-y: -10px;  
  }  
  
  &.primary {  
    background-color: oklch(55.3% 0.195 38.402);  
  }  
  
  &:hover {  
    background-color: oklch(70.5% 0.213 47.604);  
  }  
}
```

# This is plain CSS!

Press Me!

## STYLES

```
.button {  
  translate: 0 var(--translate-y, -5px);  
  transition: translate 100ms;  
  padding: 0.25rem 1rem;  
  border-radius: 0.25rem;  
  
  &:hover {  
    --translate-y: -10px;  
  }  
  
  &.primary {  
    background-color: oklch(55.3% 0.195 38.402);  
  
    &:hover {  
      background-color: oklch(70.5% 0.213 47.604);  
    }  
  }  
}
```

# One difference

```
.button {  
  &--text {  
    /* In SCSS this produces .button--text */  
    /* This is not possible in CSS */  
  }  
}
```

Solution? Please don't.

## Speaker notes

Unlike with SCSS, the &(parent selector) operator cannot be used to create a new selector by appending additional text. While this is common for BEM (Block, Elements, and Modifiers), most of us now works with components (either React or Vue). This means we naturally build our features as components, each with a very well defined scope. Each CSS can be split per component and scoped to work only with that component. There's no need for a globally unique CSS class name anymore.

Due to that, I would recommend avoiding this naming system. Using this kind of naming system makes maintenance difficult. You have to be aware of all the class names used across the application to avoid conflicts. Not only that, finding where a certain class is used is difficult since we can't just search by class name anymore. A class named `box_view-more--active` might be defined in something like:

```
.main {  
  &_aside {  
    .box {  
      &_view-more {  
        &--active{}  
      }  
    }  
  }  
}
```

which makes life unnecessarily difficult.

SCSSとは異なり、(ネイティブCSSの) & (親セレクタ) は、テキストを連結して新しいセレクタを作成する (例: `&_element` ためにには使えません)。この記法はBEM (Block, Element, Modifier) でよく使われますが、現在の私たちの多くはコンポーネント (ReactやVueなど) を使って開発しています。つまり、機能は自然とコンポーネントとして構築され、それぞれが明確に定義されたスコープを持ちます。CSSはコンポーネントごとに分割し、そのコンポーネント内でのみ機能するようにスコープを限定できるため、もはやグローバルで一意なCSSクラス名は必要ありません。そのため、このような命名規則はメンテナンスを困難にします。コンフリクトを避けるためには、アプリケーション全体で使われている全てのクラス名を把握している必要があります。それだけではなく、特定のクラスがどこで使われているかを見つけるのも困難です。なぜなら、もはやクラス名で単純に検索することができないからです。例えば、`box_view-more--active`というクラス名は、以下のよう年に定義されているかもしません。

```
.main {  
  &_aside {
```

```
.box {  
  &__view-more {  
    &--active{}  
  }  
}  
}
```

これは不必要に作業を複雑にしてしまいます。

# Further Reading & Resources

To practice flexbox and grid alignments

[CSS Grid Garden](#)    [Flexbox Froggy](#)

More detailed look at @container  
(since there are a lot of hidden intricacies)

[A Friendly Introduction to Container Queries](#)  
by Josh W. Comeau

# Further Reading & Resources

If you're more concerned about design

[7 Practical Tips for Cheating at Design](#)

by Adam Wathan & Steve Schober

[Refactoring UI \(paid resource\)](#)

by Adam Wathan & Steve Schober

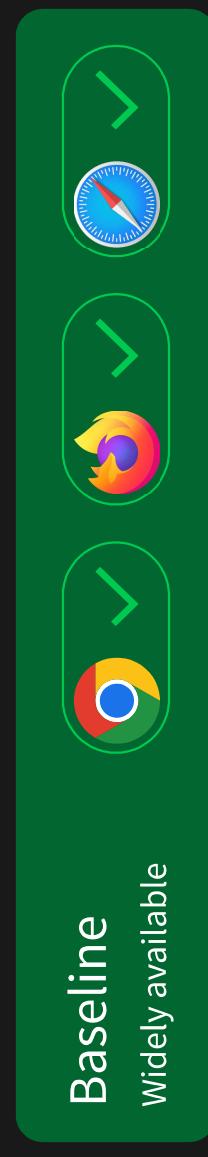
# Further Reading & Resources

How this presentation was made

[reveal.js](#)  
for the base slide  
[Tailwind](#)  
for styling

[Lit](#)

for native custom web component



Baseline  
Widely available

Questions?

# Thank you!

You can find the slide PDF, including speaker notes here:

<https://github.com/albertalrisa/20250804-practical-css>

(And the source code, but I can't promise the slide source code is tidy)

