What's new in CSS and UI: I/O 2023 Edition

Una Kravets

(https://twitter.com/una)

(https://front-end.social/@una)

(https://una.im)

Bramus

(https://twitter.com/bramus)

(https://github.com/bramus)

(https://front-end.social/@bramus)

(https://www.bram.us/)

Adam Argyle

(https://twitter.com/argyleink)

(https://github.com/argyleink)

The past few months have ushered in a golden era for web UI. New platform capabilities have landed with tight cross-browser adoption that support more web capabilities and customization features than ever.

Here are 20 of the most exciting and impactful features that landed recently or are coming soon:

- Container queries (#container_queries)
- Style queries (#style_queries)
- `:has()` selector (#has)
- nth-of microsyntax (#nth-of_syntax)
- `text-wrap: balance` (#text-wrap_balance)
- `initial-letter` (#initial-letter)
- Dynamic viewport units (#dynamic_viewport_units)
- Wide-gamut color spaces (#wide-gamut_color_spaces)

- `color-mix()` (#color-mix)
- Nesting (#nesting)
- Cascade layers (#cascade_layers)
- Scoped styles (#scoped_css)
- Trigonometric functions (#trigonometric_functions)
- Individual transform properties (#individual_transform_properties)
- popover (#popover)
- anchor positioning (#anchor_positioning)
- selectmenu (#selectmenu)
- Discrete property transitions (#discrete_property_transitions)
- Scroll-driven animations (#scroll-driven_animations)
- View transitions (#view_transitions)

The New Responsive

Let's get started with some new responsive design capabilities. New platform features let you build logical interfaces with components that own their responsive styling information, build interfaces that leverage system capabilities to deliver more native-feeling Uls, and let the user become a part of the design process with user preference queries for complete customizability.

Container Queries

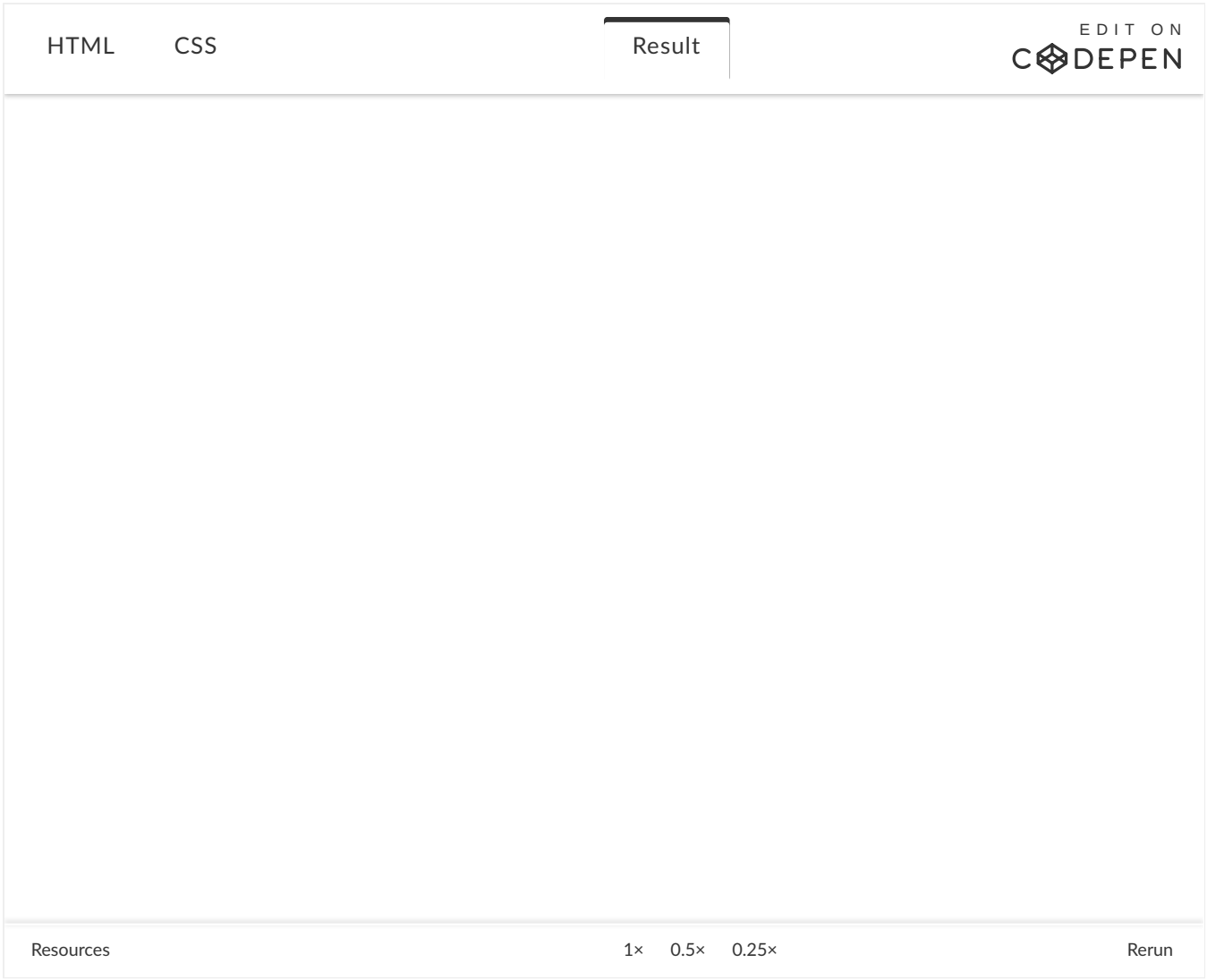
Browser Support	105	105	110	16

Source (https://developer.mozilla.org/docs/Web/CSS/@container)

Container queries (https://developer.mozilla.org/docs/Web/CSS/CSS_Container_Queries) recently became stable across all modern browsers. They allow you to query a parent element's size and style to determine the styles which should be applied to any of its children. Media queries can only access and leverage information from the viewport, which means they can only work on a macro view of a page layout. Container queries, on the other hand, are a more precise tool that can support any number of layouts or layouts within layouts.

In the following inbox example, the **Primary Inbox** and **Favorites** sidebar are both containers. The emails within them adjust their grid layout and show or hide the email timestamp based on available space. This is the exact same component within the page, just appearing in different views

Because we have a container query, the styles of these components are dynamic. If you adjust the page size and layout, the components respond to their individually allocated space. The sidebar becomes a top bar with more space, and we see the layout look more like the primary inbox. When there is less space, they both display in a condensed format.



Learn more about container queries and building logical components in this post (/blog/has-with-cq-m105).

Style Queries

Browser Support	111	111	x	x

Source (https://developer.mozilla.org/docs/Web/CSS/@container)

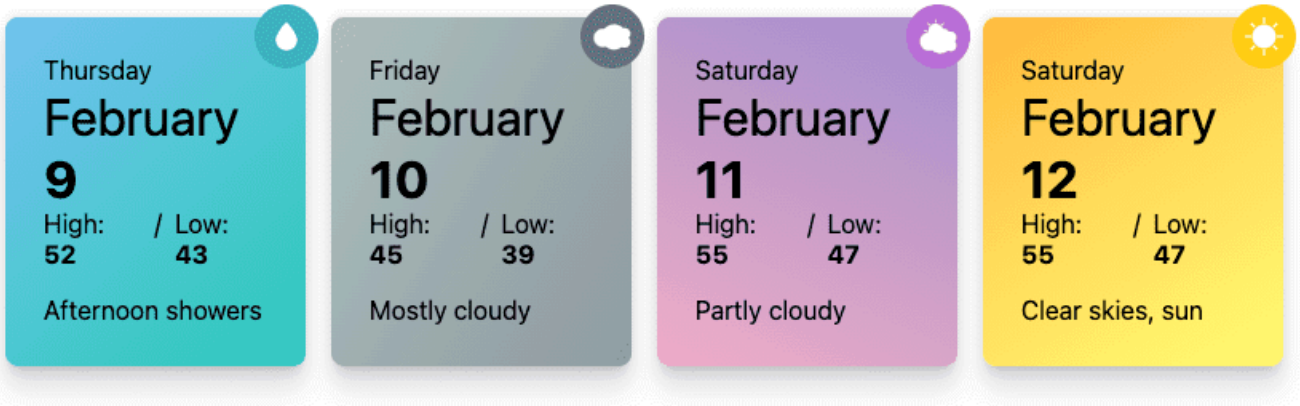
The container query specification also allows you to query the style values of a parent container. This is currently partially implemented in Chrome 111, where you can use CSS custom properties to apply container styles.

The following example uses weather characteristics stored in custom property values, such as rain, sunny, and cloudy, to style the card's background and indicator icon.

```
@container style(--sunny: true) {
 .weather-card {
   background: linear-gradient(-30deg, yellow, orange);
 }

 .weather-card:after {
   content: url(<data-uri-for-demo-brevity>);
   background: gold;
 }
}
```

HTML CSS Result EDIT ON CODEPEN

Resources 1× 0.5× 0.25× Rerun



This is just the beginning for style queries. In the future, we'll have boolean queries to determine if a custom property value exists and reduce code repetition, and currently in discussion are range queries (https://github.com/w3c/csswg-drafts/issues/8376) to apply styles

based on a range of values. This would make it possible to apply the styles shown here using a percent value for the chance of rain or cloud cover.

You can learn more and see more demos in our blog post on style queries (/blog/style-queries).

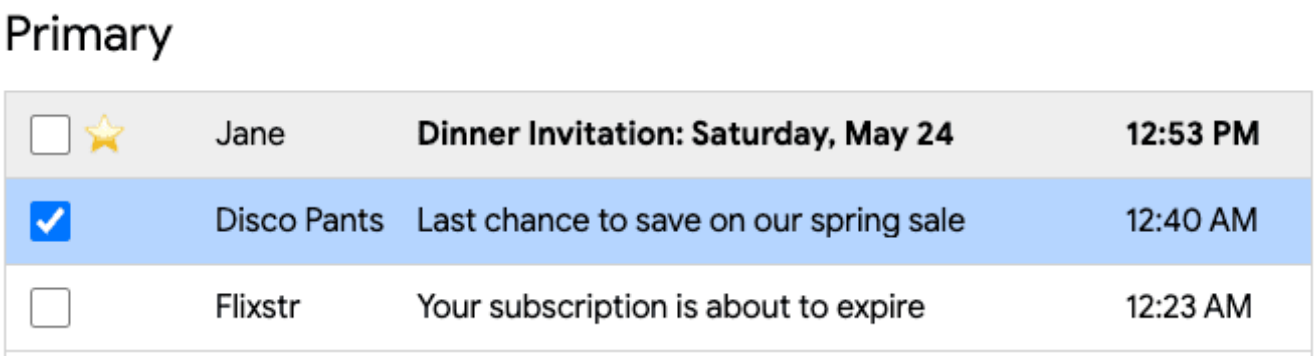
`:has()`

Browser Support	105	105	121	15.4

Source (https://developer.mozilla.org/docs/Web/CSS/:has)

Speaking of powerful, dynamic features, the :has() selector (/blog/has-m105) is one of the most powerful new CSS capabilities landing in modern browsers. With `:has()`, you can apply styles by checking to see if a parent element contains the presence of specific children, or if those children are in a specific state. This means, we essentially now have a parent selector.

Building on the container query example, you can use `:has()` to make the components even more dynamic. In it, an item with a "star" element gets a gray background applied to it, and an item with a checked checkbox a blue background.



But this API isn't limited to parent selection. You can also style any children within the parent. For example, the title is bold when the item has the star element present. This is accomplished with `.item:has(.star) .title`. Using the `:has()` selector gives you access to

parent elements, child elements, and even sibling elements, making this a really flexible API, with new use cases popping up every day.

Note: To prevent rendering performance slowdowns in large DOM trees, we recommend that you scope this selector as closely as possible. For example, using `:has()` to check for matches on the root html element would be slower than checking for matches in a nav bar or in a card element with a smaller tree.

Learn more and explore some more demos, check out this blog post (/blog/has-m105) all about `:has()`.

nth-of syntax

Browser Support	111	111	113	9

The web platform now has more advanced nth-child selection. The advanced nth-child syntax gives a new keyword ("of"), which lets you use the existing micro syntax of An+B, with a more specific subset within which to search.

If you use regular nth-child, such as `:nth-child(2)` on the special class, the browser will select the element that has the class special applied to it, and also is the second child. This is in contrast to `:nth-child(2 of .special)` which will first pre-filter all `.special` elements, and then pick the second one from that list.

Explore this feature further in our article on nth-of syntax (/articles/css-nth-child-of-s).

`text-wrap: balance`

Selectors and style queries aren't the only places that we can embed logic within our styles; typography is another one. From Chrome 114, you can use text-wrap balancing for headings, using the `text-wrap` property with the value `balance`.

0:00

Try a demo (https://codepen.io/web-dot-dev/pen/eYLwpRx)

To balance the text, the browser effectively performs a binary search for the smallest width which doesn't cause any additional lines, stopping at one CSS pixel (not display pixel). To further minimize steps in the binary search the browser starts with 80% of the average line width.

0:00

Try a demo (https://codepen.io/web-dot-dev/pen/KKxjpQm)

Note: While this is a great progressive enhancement you can try out today, it's important to note that this API works only up to 4 lines of text, so it's great for titles and headlines, but likely not what you're looking for with longer pieces of content.

Learn more about it in this article (/blog/css-text-wrap-balance).

`initial-letter`

Browser Support	110	110	x	9

Source (https://developer.mozilla.org/docs/Web/CSS/initial-letter)

Another nice improvement to web typography is `initial-letter`. This CSS property gives you better control for inset drop cap styling.

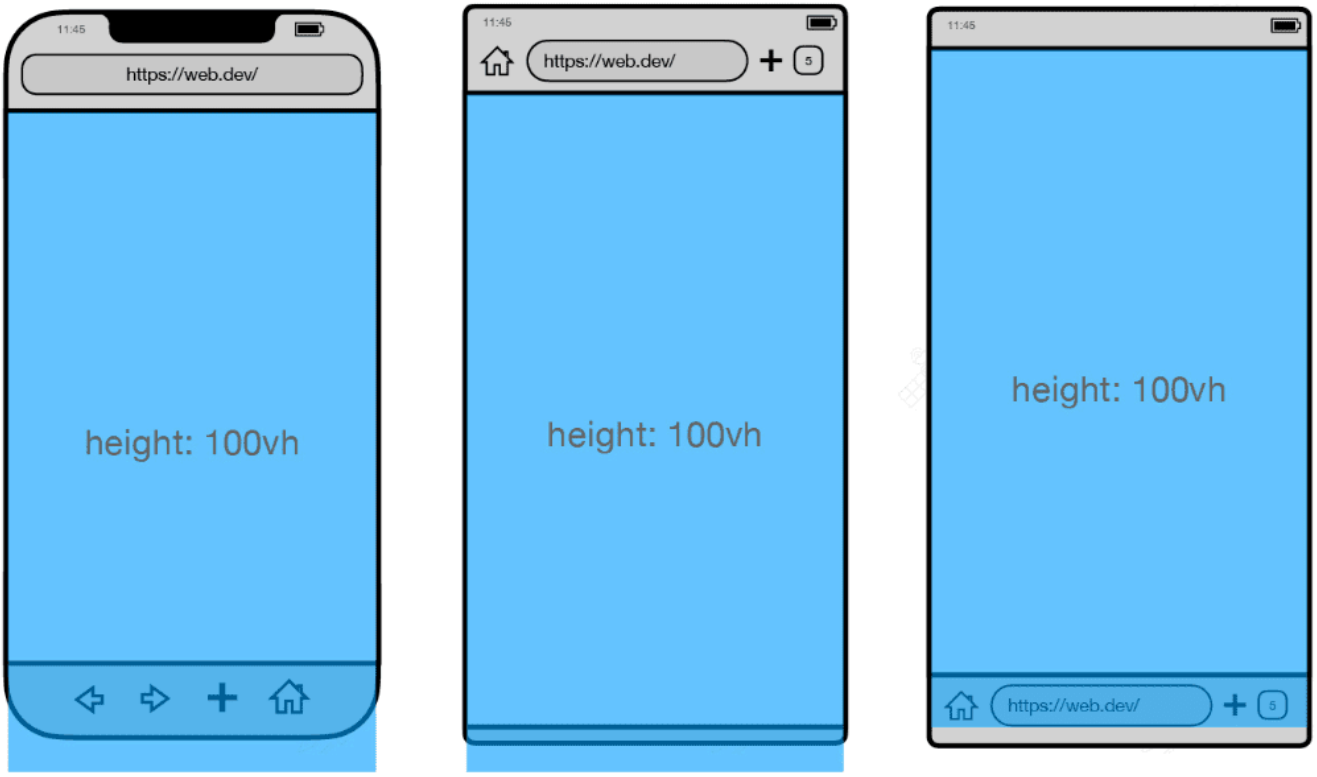
You use `initial-letter` on the `:first-letter` pseudo element to specify: The size of the letter based on how many lines it occupies. The letter's block-offset, or "sink", for where the letter will sit.

Learn more about using `intial-letter` here (/blog/control-your-drop-caps-with-css-initial-letter).

Dynamic viewport units

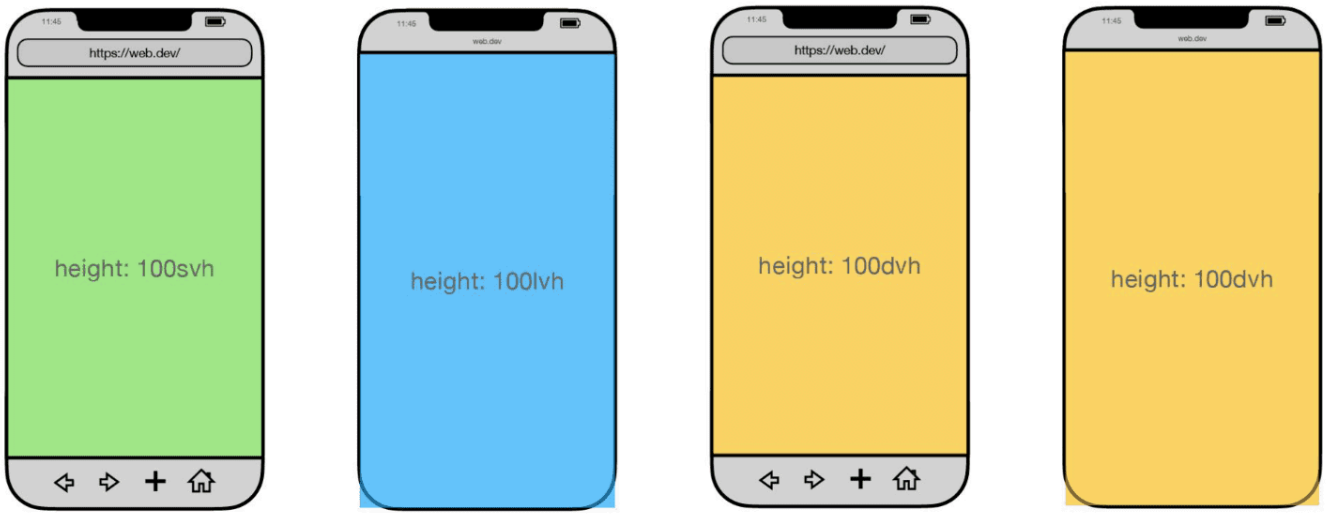
Browser Support				
	108	108	101	15.4

One common problem web developers face today is accurate and consistent full-viewport sizing, especially on mobile devices. As a developer you want **`100vh`** (100% of the viewport height) to mean “be as tall as the viewport”, but the **`vh`** unit doesn’t account for things like retracting navigation bars on mobile, so sometimes it ends up too long and causes scroll.



To resolve this issue, we now have new unit values on the web platform, including: - Small viewport height and width (or **`svh`** and **`svw`**), which represent the smallest active viewport size. - Large viewport height and width (**`lvh`** and **`lvw`**), which represent the largest size. - Dynamic viewport height and width (**`dvh`** and **`dvw`**).

Dynamic viewport units change in value when the additional dynamic browser toolbars, such as the address at the top or tab bar at the bottom, are visible and when they are not.



Note: Note that the dynamic viewport units do not take the presence of the Virtual Keyboard into account. From Chrome 108 you can set a meta-tag to change this behavior (/blog/viewport-resize-behavior#opting_in_to_a_different_behavior).

For more information about these new units, read The large, small, and dynamic viewport units (https://web.dev/blog/viewport-units).

Wide-gamut color spaces

Browser Support	111	111	113	15.4

Source (https://developer.mozilla.org/docs/Web/CSS/color_value/oklch)

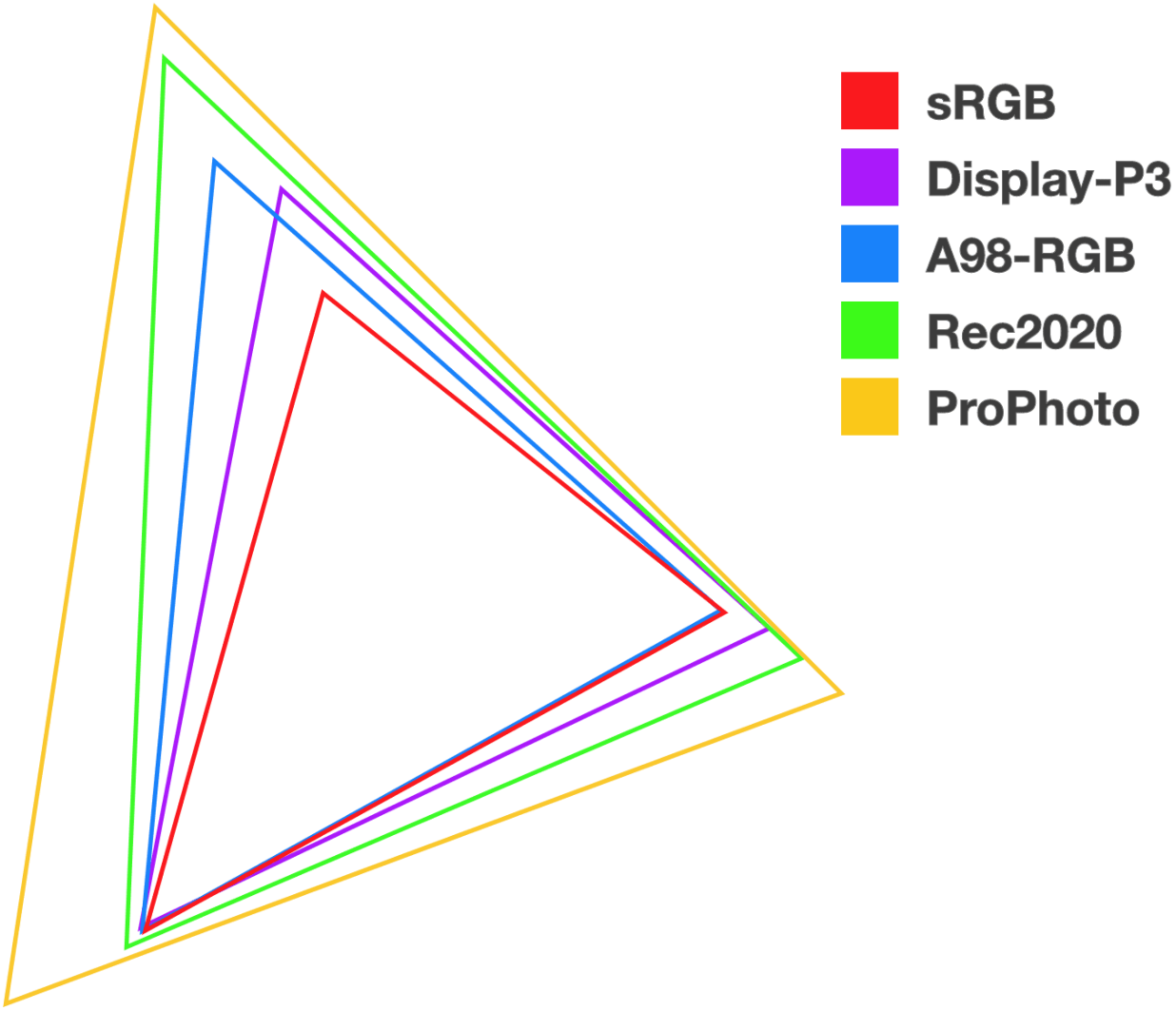
Another new key addition to the web platform are wide-gamut color spaces. Before wide-gamut color became available on the web platform, you could take a photograph with vivid colors, viewable on modern devices, but you couldn't get a button, text color, or background to match those vivid values.

0:00

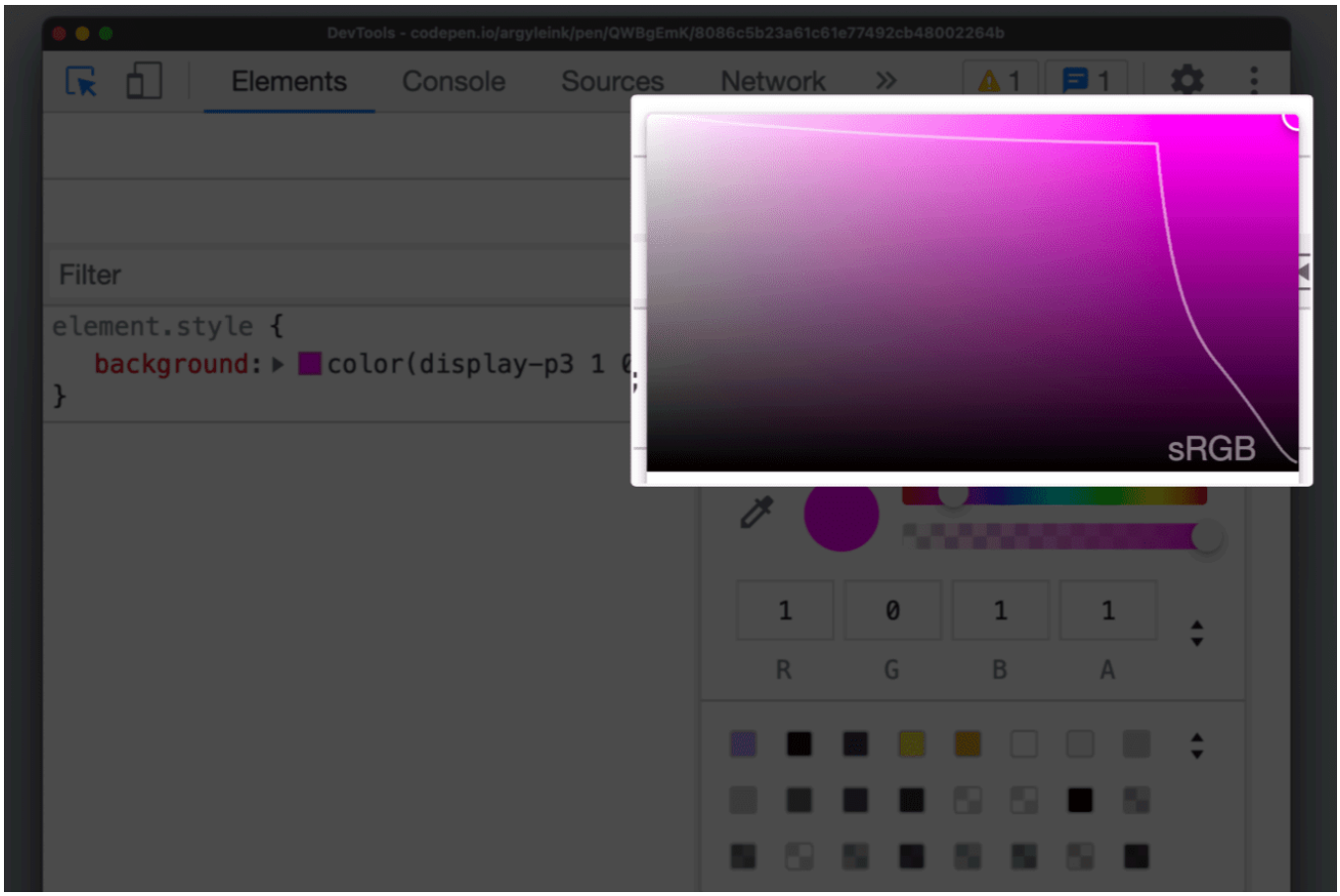
A series of images are shown transitioning between wide and narrow color gamuts, illustrating color vividness and its effects.

Try it for yourself
(https://ciechanow.ski/color-spaces/#:~:text=you%20can%20drag%20the%20slider%20to%20see%20how%20the%20extent%20of%20the%20chromaticity%20triangle%20corresponds%20to%20the%20representable%20colors.)

But now we have a range of new color spaces on the web platform including REC2020, P3, XYZ, LAB, OKLAB, LCH, and OKLCH. Meet the new web color spaces, and more, in the HD Color Guide (/docs/css-ui/high-definition-css-color-guide#meet-the-new-web-color-spaces).



And you can immediately see in DevTools how the color range has expanded, with that white line delineating where the srgb range ends, and where the wider-gamut color range begins.



Lots more tooling available for color! Don't miss all the great gradient improvements either. There's even a brand new tool Adam Argyle built to help you try out a new web color picker and gradient builder, try it out at gradient.style (https://gradient.style).

color-mix()

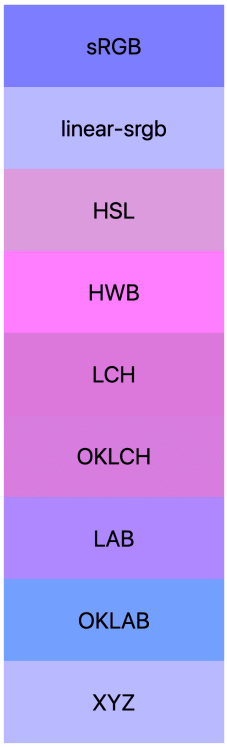
Browser Support	111	111	113	16.2

Source (https://developer.mozilla.org/docs/Web/CSS/color_value/color-mix)

Expanding on expanded color spaces is the `color-mix()` (/blog/css-color-mix) function. This function supports the mixing of two color values to create new values based on the channels of the colors getting mixed. The color space in which you mix affects the results.

Working in a more perceptual color space like oklch will run through a different color range than something like srgb.

```
color-mix(in srgb, blue, white);
color-mix(in srgb-linear, blue, white);
color-mix(in lch, blue, white);
color-mix(in oklch, blue, white);
color-mix(in lab, blue, white);
color-mix(in oklab, blue, white);
color-mix(in xyz, blue, white);
```



Try the demo (https://codepen.io/web-dot-dev/pen/eYjKMVV)

The **`color-mix()`** function provides a long-requested capability: the ability to preserve opaque color values while adding some transparency to them. Now, you can use your brand color variables while creating variations of those colors at different opacities. The way to do this is to mix a color with transparent. When you mix your brand color blue with 10% transparent, you get a 90% opaque brand color. You can see how this enables you to quickly build color systems.

You can see this in action in Chrome DevTools today with a really nice preview venn diagram icon in the styles pane (/docs/devtools/css/color#change-colors).

```
:root {
  --colorPrimary: hotpink;
  --colorPrimary-a10: color-mix(in srgb, var(--colorPrimary), transparent 90%);
  --colorPrimary-a20: color-mix(in srgb, var(--colorPrimary), transparent 80%);
  --colorPrimary-a30: color-mix(in srgb, var(--colorPrimary), transparent 70%);
  --colorPrimary-a40: color-mix(in srgb, var(--colorPrimary), transparent 60%);
  --colorPrimary-a50: color-mix(in srgb, var(--colorPrimary), transparent 50%);
  --colorPrimary-a60: color-mix(in srgb, var(--colorPrimary), transparent 40%);
  --colorPrimary-a70: color-mix(in srgb, var(--colorPrimary), transparent 30%);
  --colorPrimary-a80: color-mix(in srgb, var(--colorPrimary), transparent 20%);
  --colorPrimary-a90: color-mix(in srgb, var(--colorPrimary), transparent 10%);
```

See more examples and details in our blog post on color-mix (/blog/css-color-mix) or try out this color-mix() playground (https://color-mix.style).

CSS Foundations

Building new capabilities that have clear user wins is one part of the equation, but many of the features landing in Chrome have a goal of improving developer experience, and creating more reliable and organized CSS architecture. These features include CSS nesting, cascade layers, scoped styles, trigonometric functions, and individual transform properties.

Nesting

Browser Support	120	120	117	17.2

Source (https://developer.mozilla.org/docs/Web/CSS/Nesting_selector)

CSS nesting, something folks love from Sass, and one of the top CSS developer requests for years, is finally landing on the web platform. Nesting allows developers to write in a more succinct, grouped format that reduces redundancy.

```
.card {}
.card:hover {}

/* can be done with nesting like */
.card {
  &:hover {

  }
}
```

You can also nest Media Queries (/learn/design/media-queries), which also means you can nest Container Queries (https://web.dev/articles/cq-stable). In the following example, a card is changed from a portrait layout to a landscape layout if there's enough width in it's container:

```
.card {
  display: grid;
  gap: 1rem;

  @container (width >= 480px) {
    display: flex;
  }
}
```

The layout adjustment to `flex` occurs when the container has more (or equal to) `480px` of inline space available. The browser will simply apply that new display style when the

conditions are met.

For more information and examples, check out our post on CSS nesting (/articles/css-nesting).

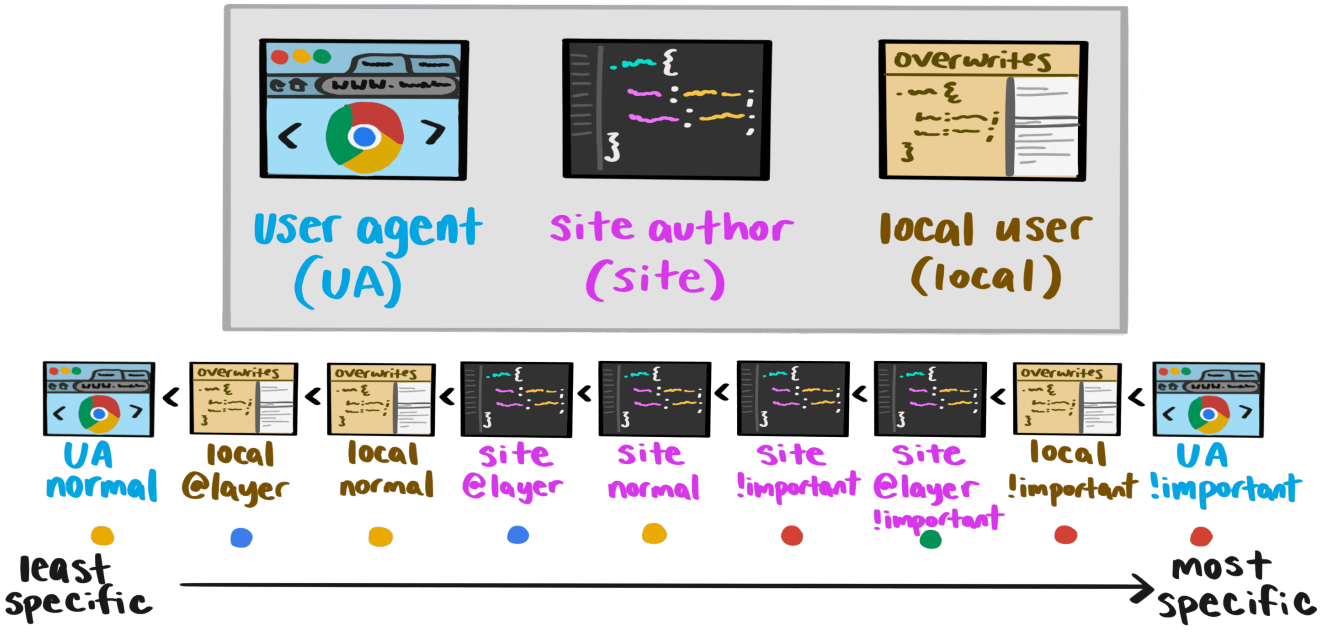
Cascade layers

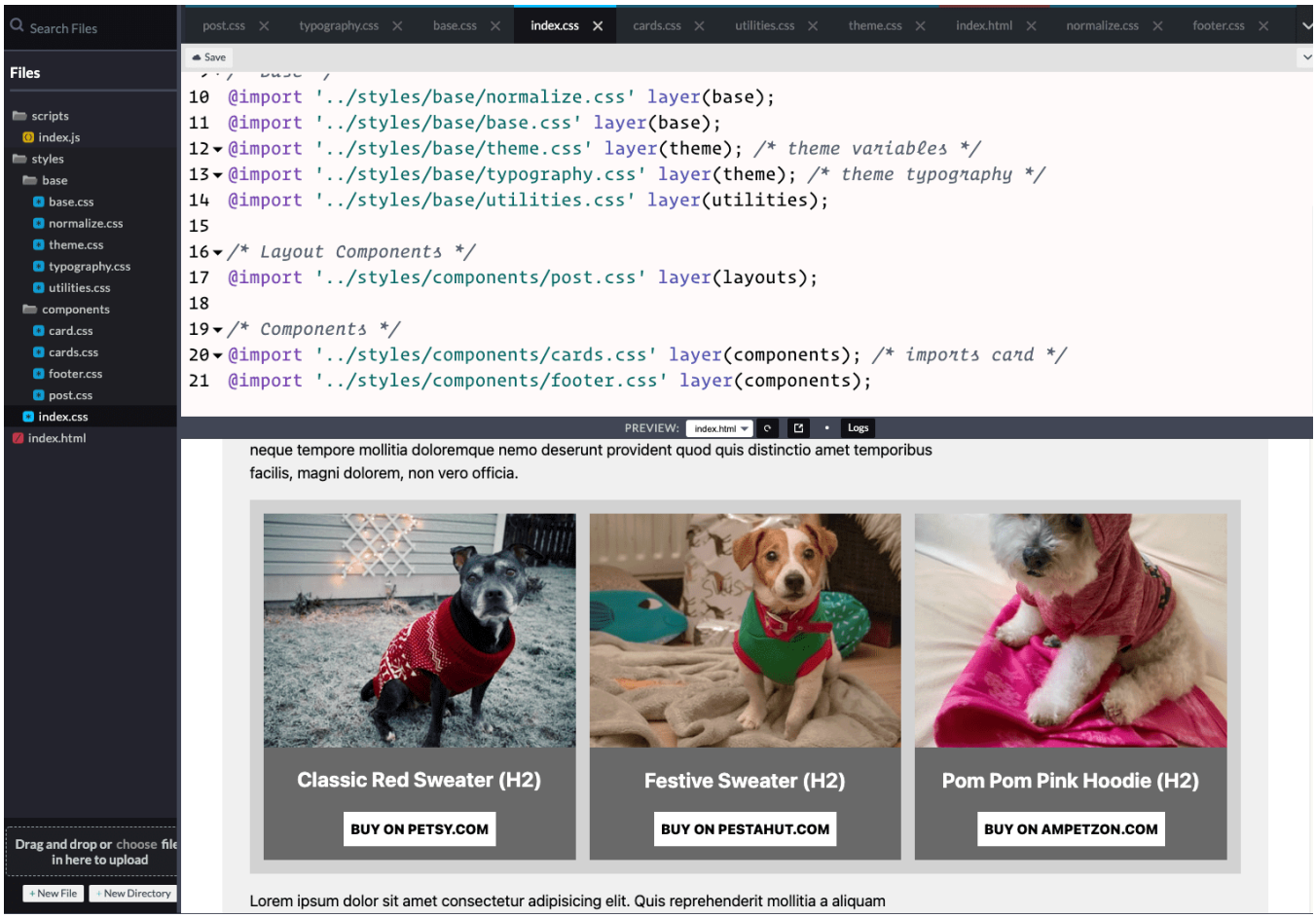
Browser Support	99	99	97	15.4

Source (https://developer.mozilla.org/docs/Web/CSS/@layer)

Another developer pain point we've identified is ensuring consistency in which styles win over others, and one part of resolving this is having better control over the CSS cascade.

Cascade layers (/blog/cascade-layers) solve this by giving users control over which layers have a higher precedence than others, meaning more fine-tuned control of when your styles are applied.





Explore the project on Codepen. (https://codepen.io/web-dot-dev/project/editor/ZGQLkq)

Learn more about how to use cascade layers in this article (/blog/cascade-layers).

Scoped CSS

Browser Support	118	118	x	17.4

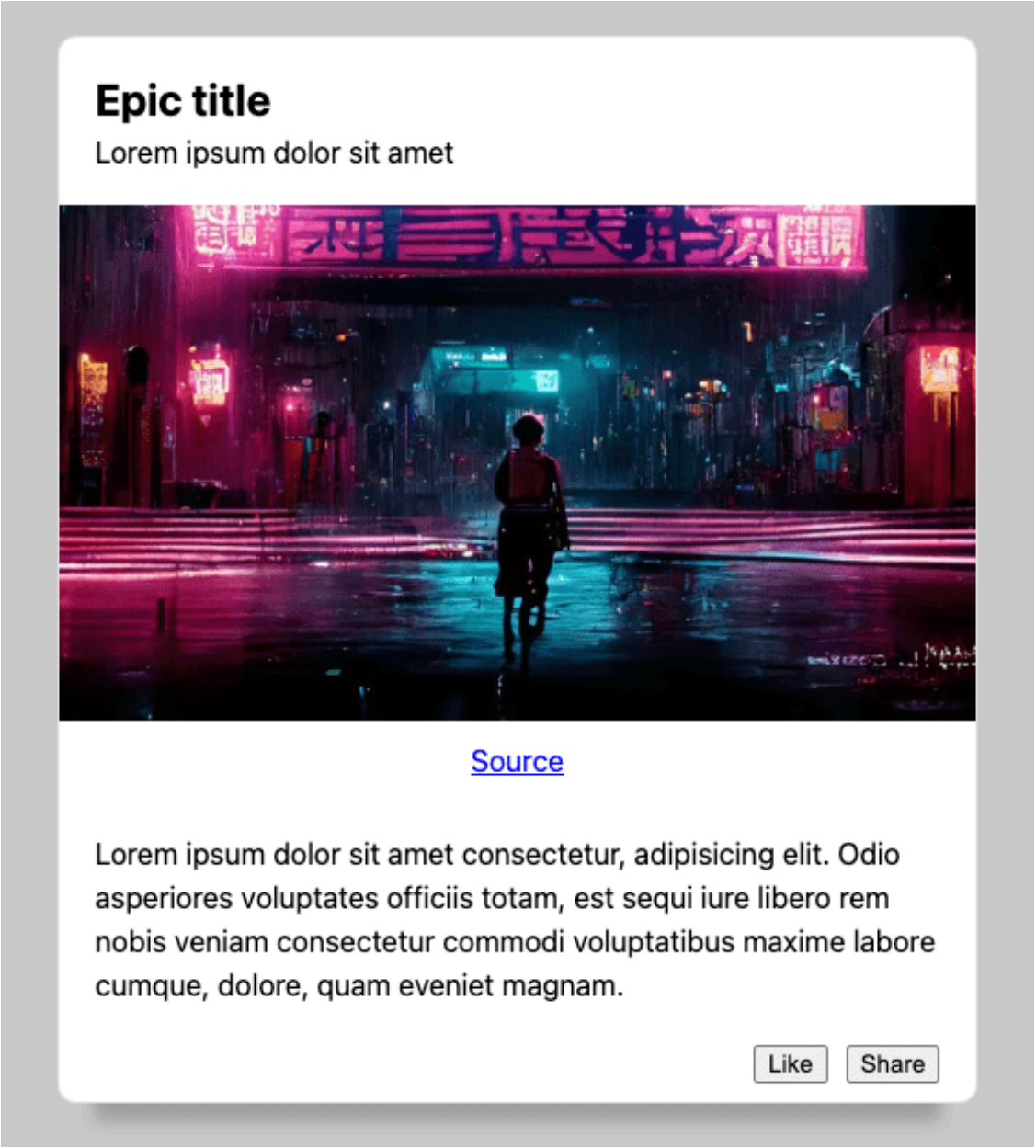
Source (https://developer.mozilla.org/docs/Web/CSS/@scope)

CSS scoped styles allow developers to specify the boundaries for which specific styles apply, essentially creating native namespacing in CSS. Before, developers relied on 3rd party scripting to rename classes, or specific naming conventions to prevent style collision, but soon, you can use **@scope**.

Here, we're scoping a `.title` element to a `.card`. This would prevent that title element from conflicting with any other `.title` elements on the page, like a blog post title or other heading.

```
@scope (.card) {
  .title {
    font-weight: bold;
  }
}
```

You can see `@scope` with scoping limits together with `@layer` in this live demo:



Learn more about **@scope** in the css-cascade-6 specification (https://www.w3.org/TR/css-cascade-6/#scoped-styles).

Trigonometric functions

Browser Support	111	111	108	15.4

Source (https://developer.mozilla.org/docs/Web/CSS/sin)

Another piece of new CSS plumbing are the trigonometric functions being added to the existing CSS math functions. These functions are now stable in all modern browsers, and enable you to create more organic layouts on the web platform. One great example is this radial menu layout, which is now possible to design and animate using `sin()` and `cos()` functions.

In the demo below, the dots revolve around a central point. Instead of rotating each dot around its own center and then moving it outwards, each dot is translated on the X and Y axes. The distances on the X and Y axes are determined by taking the `cos()` and, respectively, the `sin()` of the `--angle` into account.

See our article on trigonometric functions (https://web.dev/articles/css-trig-functions) for more detailed information on this topic.

Individual transform properties

Browser Support	104	104	72	14.1

Source (https://developer.mozilla.org/docs/Web/CSS/translate)

Developer ergonomics continue to improve with individual transform functions. Since the last time we held I/O, individual transforms went stable across all modern browsers.

In the past, you would rely on the transform function to apply sub-functions to scale, rotate, and translate a UI element. This involved a lot of repetition, and was especially frustrating when applying multiple transforms at different times in the animation.

```
.target {
  transform: translateX(50%) rotate(30deg) scale(1.2);
}

.target:hover {
  transform: translateX(50%) rotate(30deg) scale(2); /* Only scale changed here, yet
}
```

Now, you can have all of this detail in your CSS animations by separating the types of transforms and applying them individually.

```
.target {
  translate: 50% 0;
  rotate: 30deg;
  scale: 1.2;
```

```
}

.target:hover {
  scale: 2;
}
```

With this, changes in translation, rotation, or scale can happen simultaneously at different rates of change in different times during the animation.

See this post on individual transform functions (https://web.dev/articles/css-individual-transform-properties) for more information.

Customizable Components

To make sure we're resolving some of the key developer needs through the web platform, we're working with the OpenUI community group (http://open-ui.com/) and have identified three solutions to start with:

1. Built-in popup functionality with event handlers, a declarative DOM structure, and accessible defaults.
2. A CSS API to tether two elements to each other to enable anchor positioning.
3. A customizable dropdown menu component, for when you want to style content inside of a select.

Popover

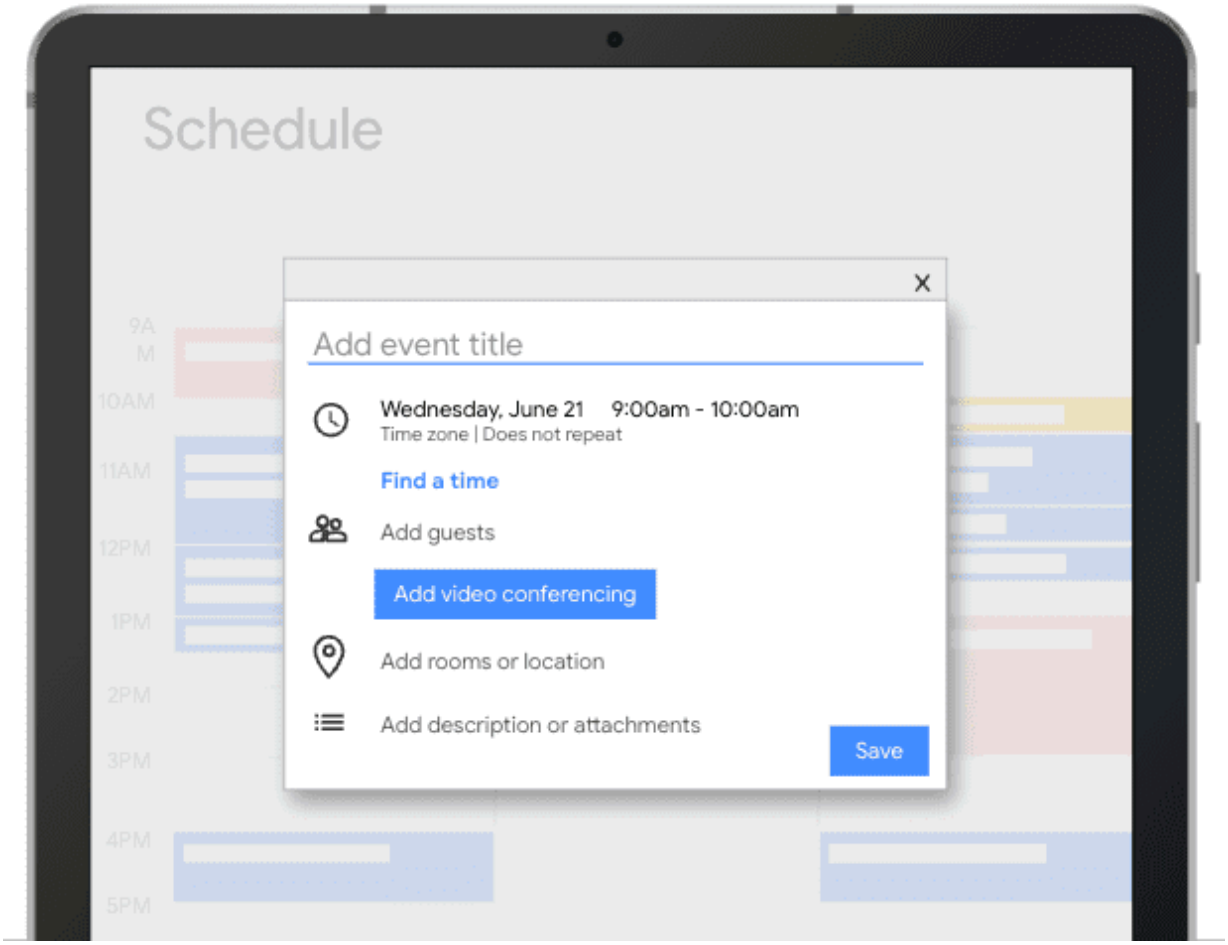
The popover API (https://developer.mozilla.org/docs/Web/HTML/Global_attributes/popover) gives elements some built-in browser-support magic such as:

- Support for top-layer, so you don't have to manage **`z-index`**. When you open a popover or a dialog, you're promoting that element to a special layer on top of the page.
- Light-dismiss behavior for free in **`auto`** popovers, so when you click outside of an element, the popover is dismissed, removed from the accessibility tree, and focus

properly managed.

- Default accessibility for the connective tissue of the popover's target and the popover itself.

All of this means less JavaScript has to be written to create all of this functionality and track all of these states.



The DOM structure for popover is declarative and can be written as clearly as giving your popover element an `id` and the `popover` attribute. Then, you sync that id to the element which would open the popover, such as a button with the `popovertarget` attribute:

```
<div id="event-popup" popover>
  <!-- Popover content goes in here -->
</div>
```

```
<button popovertarget="event-popup">Create New Event</button>
```

`popover` is a shorthand for `popover=auto`. An element with `popover=auto` will force-close other popovers when opened, receive focus when opened, and can light-dismiss. Conversely, `popover=manual` elements do not force-close any other element type, do not receive focus immediately, and do not light-dismiss. They close via a toggle or other close action.

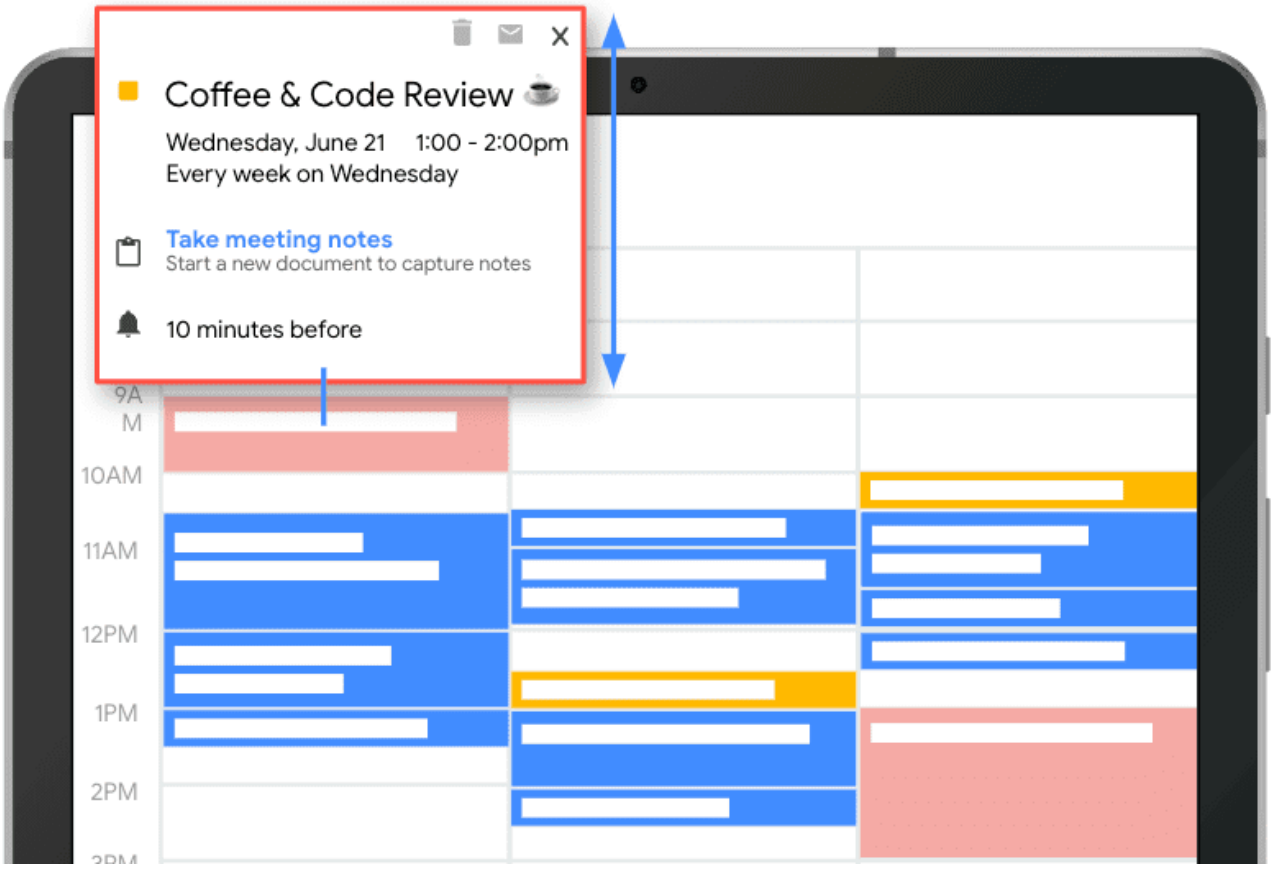
The most up-to-date documentation on popovers can currently be found on MDN (https://developer.mozilla.org/docs/Web/HTML/Global_attributes/popover).

Anchor positioning

Popovers are also frequently used in elements such as dialogs and tooltips, which typically need to be anchored to specific elements. Take this event example. When you click on a calendar event, a dialog appears near the event you've clicked on. The calendar item is the anchor, and the popover is the dialog which shows the event details.

You can create a centered tooltip with the **anchor()** function, using the width from the anchor to position the tooltip at 50% of the anchor's x position. Then, use existing positioning values to apply the rest of the placement styles.

But what happens if the popover doesn't fit in the viewport based on the way you've positioned it?



To solve for this, the anchor positioning API includes fallback positions that you can customize. The following example creates a fallback position called "top-then-bottom". The browser will first try to position the tooltip at the top, and if that doesn't fit in the viewport, the browser would then position it under the anchoring element, on the bottom.

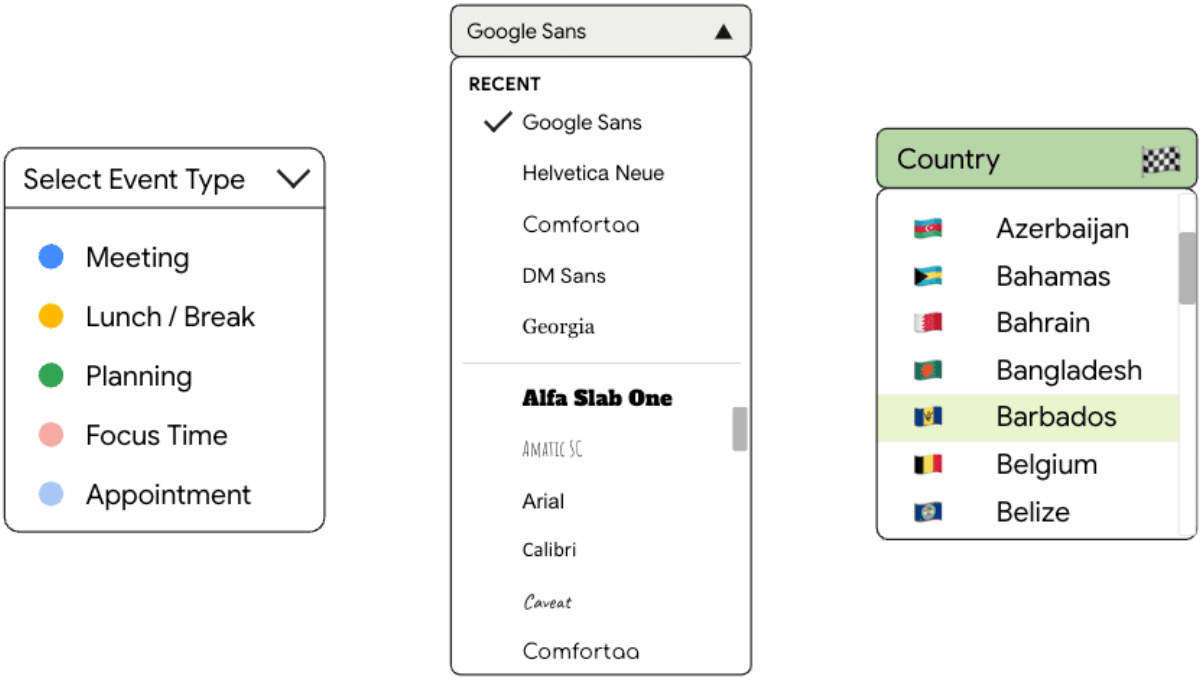
```
.center-tooltip {
  position-fallback: --top-then-bottom;
  translate: -50% 0;
}

@position-fallback --top-then-bottom {
  @try {
    bottom: calc(anchor(top) + 0.5rem);
    left: anchor(center);
  }

  @try {
    top: calc(anchor(bottom) + 0.5rem);
    left: anchor(center);
  }
}
```

Note: You can get really granular here, which may get verbose, but it's also an opportunity for libraries and design systems to write the positioning logic and for you to reuse it everywhere.

Learn more about anchor positioning in this blog post (/blog/tether-elements-to-each-other-with-css-anchor-positioning).

`<selectmenu>`

With both popover and anchor positioning, you can build fully customizable selectmenus. The OpenUI community group has been investigating the fundamental structure of these menus and looking for ways to allow for the customization of any content within them. Take these visual examples:



To build that left-most **selectmenu** example, with colored dots corresponding to the color that would show within a calendar event, you can write it as follows:

```
<selectmenu>
  <button slot="button" behavior="button">
    <span>Select event type</span>
    <span behavior="selected-value" slot="selected-value"></span>
    <span><img src="icon.svg"/></span>
  </button>
  <option value="meeting">
    <figure class="royalblue"></figure>
    <p>Meeting</p>
  </option>
  <option value="break">
    <figure class="gold"></figure>
     <p>Lunch/Break</p>
  </option>
  ...
</selectmenu>
```

Discrete property transitions

In order for all of this to transition popovers in and out smoothly, the web needs some way to animate discrete properties. These are properties that typically weren't animatable in the past, such animating to and from the top-layer and animating to and from `display: none`.

As a part of the work to enable nice transitions for popovers, selectmenus, and even existing elements like dialogs or custom components, browsers are enabling new plumbing to support these animations.

The following popover demo, animates popovers in and out using **`:popover-open`** for the open state, **`@starting-style`** for the before-open state, and applies a transform value to the element directly for the after-open-is-closed state. To make this all work with display, it needs adding to the **`transition`** property, like so:

```
.settings-popover {
  &:popover-open {
    /*   0. before-change   */
    @starting-style {
      transform: translateY(20px);
      opacity: 0;
    }

    /*   1. open (changed) state   */
    transform: translateY(0);
    opacity: 1;
  }

  /*   2. After-change state */
  transform: translateY(-50px);
  opacity: 0;

  /*  enumarate transitioning properties, including display */
  transition: transform 0.5s, opacity 0.5s, display 0.5s allow-discrete;
}
```

Interactions

Which brings us to interactions, the last stop on this tour of web UI features.

We already talked about animating discrete properties, but there are also some really exciting APIs landing in Chrome around scroll-driven animations and view transitions

Scroll-driven animations

Browser Support	115	115	x

Source (https://developer.mozilla.org/docs/Web/CSS/animation-timeline)

Scroll-driven animations allow you to control the playback of an animation based on the scroll position of a scroll container. This means that as you scroll up or down, the animation scrubs forward or backward. Additionally, with scroll-driven animations you can also control an animation based on an element's position within its scroll container. This allows you to create interesting effects such as a parallax background image, scroll progress bars, and images that reveal themselves as they come into view.

This API supports a set of JavaScript classes and CSS properties that allow you to easily create declarative scroll-driven animations.

To drive a CSS Animation by scroll use the new `scroll-timeline`, `view-timeline`, and `animation-timeline` properties. To drive a JavaScript Web Animations API, pass a `ScrollTimeline` or `ViewTimeline` instance as the `timeline` option to `Element.animate()`

These new APIs work in conjunction with existing Web Animations and CSS Animations APIs, meaning that they benefit from the advantages of these APIs. That includes the ability to have these animations run off the main thread. Yes, read that correctly: you can now have silky smooth animations, driven by scroll, running off the main thread, with just a few lines of extra code. What's not to like?!

For an extensive in-depth guide to how to create these scroll-driven animations, please refer to this article on scroll-driven animations (/docs/css-ui/scroll-driven-animations).

View transitions

Browser Support	111	111	x	x

Source (https://developer.mozilla.org/docs/Web/CSS/view-transition-name)

The View Transition API makes it easy to change the DOM in a single step, while creating an animated transition between the two states. These can be simple fades between views, but you can also control how individual parts of the page should transition.

View Transitions can be used as a Progressive Enhancement: take your code that updates the DOM by whatever method and wrap it in the view transition API with a fallback for browsers that don't support the feature.

```
function spaNavigate(data) {
  // Fallback for browsers that don't support this API:
  if (!document.startViewTransition) {
    updateTheDOMSomehow(data);
    return;
  }

  // With a transition:
  document.startViewTransition(() => updateTheDOMSomehow(data));
}
```

What the transition should look like is controlled via CSS

```
@keyframes slide-from-right {
  from { opacity: 0; transform: translateX(75px); }
}

@keyframes slide-to-left {
  to { opacity: 0; transform: translateX(-75px); }
}

::view-transition-old(root) {
  animation: 350ms both slide-to-left ease;
}

::view-transition-new(root) {
  animation: 350ms both slide-from-right ease;
}
```

As demonstrated in this wonderful demo (https://live-transitions.netlify.app/) by Maxi Ferreira (https://twitter.com/charca), other page interactions, such as a playing video, keep working while a View Transition is happening.

View Transitions currently work with Single-Page Apps (SPAs) from Chrome 111. Multiple-page app support is being worked on. For more, check out our full view transitions guide (/docs/web-platform/view-transitions) to walk you through it all.

Conclusion

Keep up with all the latest landings in CSS and HTML right here on developer.chrome.com (/) and check out the I/O videos (https://www.youtube.com/playlist?list=PLOU2XLYxmsIJGxIV8Lt8gF_79Z334LQ6h) for more web landings.

What's new in web UI

What's new in web animations

How to create personalized web experiences

Last updated 2023-05-11 UTC.