@FUNCTION ATTR() CALC-SIZE() CSS FIELD-SIZING IF() INTERPOLATE-SIZE INVOKERS LINEAR() RANDOM() READING-FLOW TEXT-WRAP

What You Need to Know about Modern CSS (2025 Edition)

CHRIS COYIER ON SEPTEMBER 19, 2025

We published an edition of What You Need To Know about Modern CSS last year (2024), and for a while I really wasn't sure if only a year later we'd have enough stuff to warrant and new yearly version. But time, and CSS, have rolled forward, and guess what? There is

more this year than there was last. At least in this somewhat arbitrary list of *"things Chris thinks are valuable to know that are either pretty fresh or have enjoyed a boost in browser support."*

ANIMATE TO AUTO

What is this?

We don't often set the `height` of elements that contain arbitrary content. We usually let elements like that be as tall as they need to be for the content. The trouble with that is we haven't been able to animate from a fixed number (like zero) to whatever that intrinsic height is (or vice versa). In other words, animate to `auto` (or other sizing keywords like `min-content` and the like).

Now, we can opt-in to being able to animate to these keywords, like:

CSS

```css
html {
  interpolate-size: allow-keywords;
  /* Now if we transition
     "height: 0;" to "height: auto;"
     anywhere, it will work */
}
```

If we don't want to use an opt-in like that, alternatively, we can use the `calc-size()` function to make the transition work without needing `interpolate-size`.

CSS

```css
.content {
  height: 3lh;
  overflow: hidden;
  transition: height 0.2s;
```

```
  &.expanded {
    height: calc-size(auto, size);
  }
}
```

Why should I care?

This is the first time we've ever been able to do this in CSS. It's a relatively common need and it's wonderful to be able to do it so naturally, without breaking behavior.

And it's not just `height` (it could be any property that takes a size) and it's not just `auto` (it could be any sizing keyword).

Support

Browser Support	Just Chrome.
Progressive Enhancement	Yes! Typically, this kind of animation isn't a hard requirement, just a nice-to-have.
Polyfill	Not really. The old fallbacks including things like animating `max-height` to a beyond-what-is-needed value, or using JavaScript to attempt to measure the size off-screen and then doing the real animation to that number. Both suck.

Usage Example

style.css

```css
.content {
  padding: 1rem;
  height: 3lh;
  overflow: hidden;
  transition: height 0.2s;
}

.demo-1 {
  interpolate-size: allow-keywords;

  .content {
    &.expanded {
      height: auto;
    }

    /* just for looks */
    & {
      position: relative;
      &::after {
        content: "";
        position: absolute;
        inset: 0;
        background-image: linear-gradient(to top, #03a9f4, transparent 100px);
      }
      &.expanded::after {
        display: none;
      }
    }
  }
}

.demo-2 {
  .content {
    &.expanded {
      height: auto; /* gotta have fallback */
      height: calc-size(auto, size);
    }

    /* just for looks */
    & {
      position: relative;
      &::after {
        content: "";
        position: absolute;
        inset: 0;
        background-image: linear-
```

Preview

EDIT ON CODEPEN

Animating to Sizing Keywords

Animate height to auto

interpolate-size: allow-keywords;

Toggle

Animate height to auto

calc-size(auto, size)

Toggle

Animate inline-size to min-content

❤️

🐘

🤪

Animate ::details-content

```
    gradient(to top, #66bb6a,
    transparent 100px);
47        }
48        &.expanded::after {
49          display: none;
50        }
51      }
52    }
53  }
54
55  .demo-3 {
56    interpolate-size: allow-
    keywords;
57
58    button {
59      padding: 0.25rem;
60      margin-block-end: 0.25rem;
```

```
transition: height 0.2s, content-vis
transition-behavior: allow-discrete;
```

▶ Open Me

POPOVERS & INVOKERS

These are separate and independently useful things, and really rather HTML-focused, but it's nice to show them off together as they complement each other nicely.

What is this?

A `popover` is an attribute you can put on any HTML element that essentially gives it open/close functionality. It will then have JavaScript APIs for opening and closing it. It's similar-but-different to modals. Think of them more in the tooltip category, or something that you might want more than one of open sometimes.

Invokes are also HTML attributes that give us access to those JavaScript APIs in a declarative markup way.

Why should I care?

Implementing functionality at the HTML level is very powerful. It will work without JavaScript, be done in an accessible way, and likely get important UX features right that you might miss when implementing yourself.

Support

Browser Support	Popovers are everywhere, but invokers are Chrome only at time of publication. There are sub-features here though, like `popover="hint"` which has slightly less support so far.
Progressive Enhancement	Not so much. These type of functions typically need to *work*, so ensuring they do with a polyfill instead of handling multiple behaviors is best.
Polyfill	Yep! For both: Popovers Polyfill Invokers Polyfill

Usage Example

Remember there are JavaScript APIs for popovers also, like `myPopover.showPopover()` and `secondPopover.hidePopover()` but what I'm showing off here is specifically the HTML invoker controls for them. There are *also* some alternative HTML controls (e.g. `popovertarget="mypopover" popovertargetaction="show"`) which I suppose are fine to use as well? But something feels better to me about the more generic command invokers approach.

index.pen.html

Preview

EDIT ON CODEPEN

```
<button commandfor="popover-1"
command="toggle-popover">
  Toggle Popover
</button>

<button commandfor="popover-2"
command="toggle-popover">
  Toggle 2nd ("manual") Popover
</button>

<aside popover id="popover-1">
  Content of popover. <a
href="https://en.wikipedia.org/wik
i/Lizard">Lizards (focusable
element)</a>. <button
commandfor="popover-1"
command="hide-
popover">Close</button></button>
</aside>

<div id="popover-2"
popover="manual">
```

Also — remember popovers pair particularly well with anchor positioning which is another CSS modern miracle.

@FUNCTION

What is this?

CSS has lots of functions already. Think of `calc()`, `attr()`, `clamp()`, perhaps hundreds more. They are actually technically called CSS *value* functions as they always return a single value.

The magic with with `@function` is that now *you can write your own.*

CSS

```
@function --titleBuilder(--name) {
  result: var(--name) " is cool.";
}
```

Why should I care?

Abstracting logic into functions is a computer programming concept as old as computers itself. It can just *feel right*, not to mention be DRY, to put code and logic into a single shared place rather than repeat yourself or complicate the more declarative areas of your CSS with complex statements.

Support

Browser Support	Chrome only

Progressive Enhancement	It depends on what you're trying to use the value for. If it's reasonable, it may be as simple as:
	`property: fallback;`
	`property: --function();`

Polyfill	Not really. Sass has functions but are not based on the same spec and will not work the same.

Usage Example

index.html Preview EDIT ON CODEPEN

Using @function in CSS

Title of Card

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Iste vero autem ratione, tempore aut corrupti, dolorum quidem, sapiente voluptate animi reprehenderit corporis? Eum saepe ad ducimus maxime

Title of Card

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Iste vero autem ratione, tempore aut corrupti, dolorum quidem, sapiente voluptate animi reprehenderit corporis? Eum saepe ad ducimus maxime

Title of Card

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Iste vero autem ratione, tempore aut corrupti, dolorum quidem, sapiente voluptate animi reprehenderit corporis? Eum saepe ad ducimus maxime

Other Resources

- Una Kravets: 5 Useful CSS functions using the new @function rule
- Bramus Van Damme: CSS `@function` + CSS `if()` = 🤯
- Juan Diego Rodríguez: Functions in CSS?!
- Draft: CSS Functions and Mixins Module

IF()

What is this?

Conceptually, CSS is already full of conditional logic. Selectors themselves will match and apply styles *if* they match an HTML element. Or media queries will apply *if* their conditions are met.

But the `if()` function, surprisingly, is the first specific logical construct that exists soley for the function of applying logical branches.

Why should I care?

Like all functions, including custom @functions like above, `if()` returns a single value. It just has a syntax that might help make for more readable code and potentially prevent certain types of code repetition.

Support

Browser Support	Chrome only It depends on the property/value you are using it with. If you're OK with a fallback value, it might be fine to use.

Progressive Enhancement	`property: fallback;`
	`property: if(`
	`style(--x: true): value;`
	`else: fallback;`
	`);`

Polyfill	Not really. CSS processes tend to have logical constructs like this, but they will not re-evaluate based on dynamic values and DOM placement and such.

Usage Example

Baking logic into a single value like this is pretty neat!

CSS

```
.grid {
  display: grid;
  grid-template-columns:
    if(
       media(width > 900px): repeat(auto-fit, minmax(200px, 1fr));
       media(width > 600px): repeat(3, 1fr);
       media(width > 300px): repeat(2, 1fr);
       else: 1fr;
    );
}
```

The syntax is a lot like a switch statement with as many conditions as you need. The first match wins.

CSS

```
if(
  condition: value;
  condition: value;
  else: value;
)
```

Conditions can be:

- `media()`
- `supports()`
- `style()`

FIELD-SIZING

What is this?

The new `field-sizing` property in CSS is for creating form fields (or any editable element) that automatically grows to to the size of their contents.

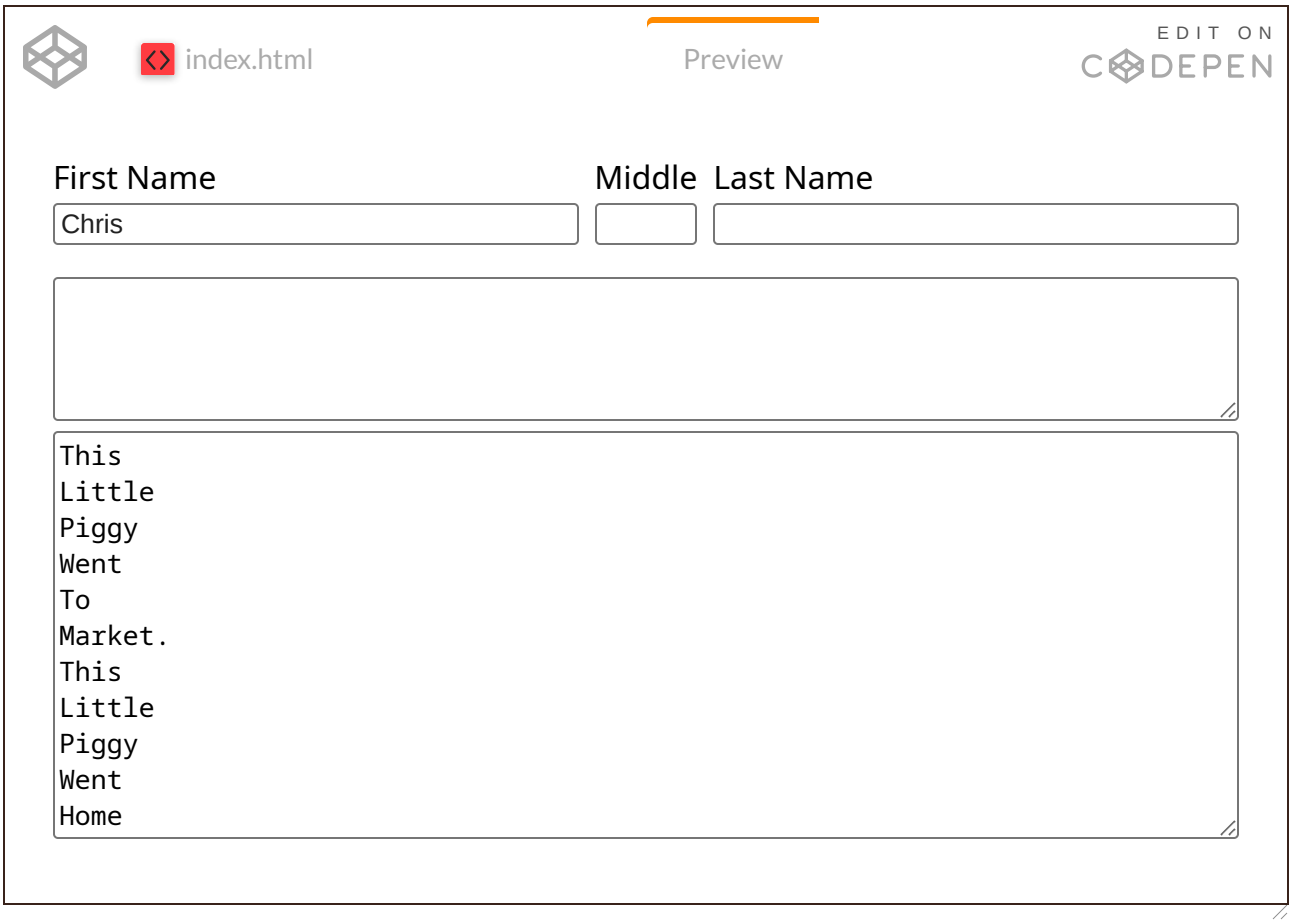
Why should I care?

This is a need that developers have been creating in JavaScript since forever. The most classic example is the `<textarea>`, which makes a lot of sense to be sized to as large as the user entering information into it needs to be, without having to explicitly resize it (which is difficult at best on a small mobile screen). But inline resizing can be nice too.

Support

Browser Support	Chrome and looks to be coming soon to Safari.
Progressive Enhancement	Yes! This isn't a hard requirement usually but more of a UX nicety.
Polyfill	There is some very lightweight JavaScript to replicate this if you want to.

Usage Example



CUSTOM SELECTS

What is this?

Styling the *outside* of a `<select>` has been decently possible for a while, but when you open it up, what the browser renders is an operating-system specific default. Now you can opt-in to entirely styleable select menus.

Why should I care?

Support

Browser Support	Chrome only
Progressive Enhancement	100%. It just falls back to a not-styled `<select>` which is fine.
Polyfill	Back when this endeavor was using `<selectlist>` there was, but in my opinion the progressive enhancement story is so good you don't need it.

Usage Example

First you opt-in then you go nuts.

CSS

```
select,
::picker(select) {
  appearance: base-select;
}
```

TEXT-WRAP

What is this?

The `text-wrap` property in CSS allows you to instruct the browser that it can and should wrap text a bit differently. For example, `text-wrap: balance;` will attempt to have each line of text as close to the same length as possible.

Why should I care?

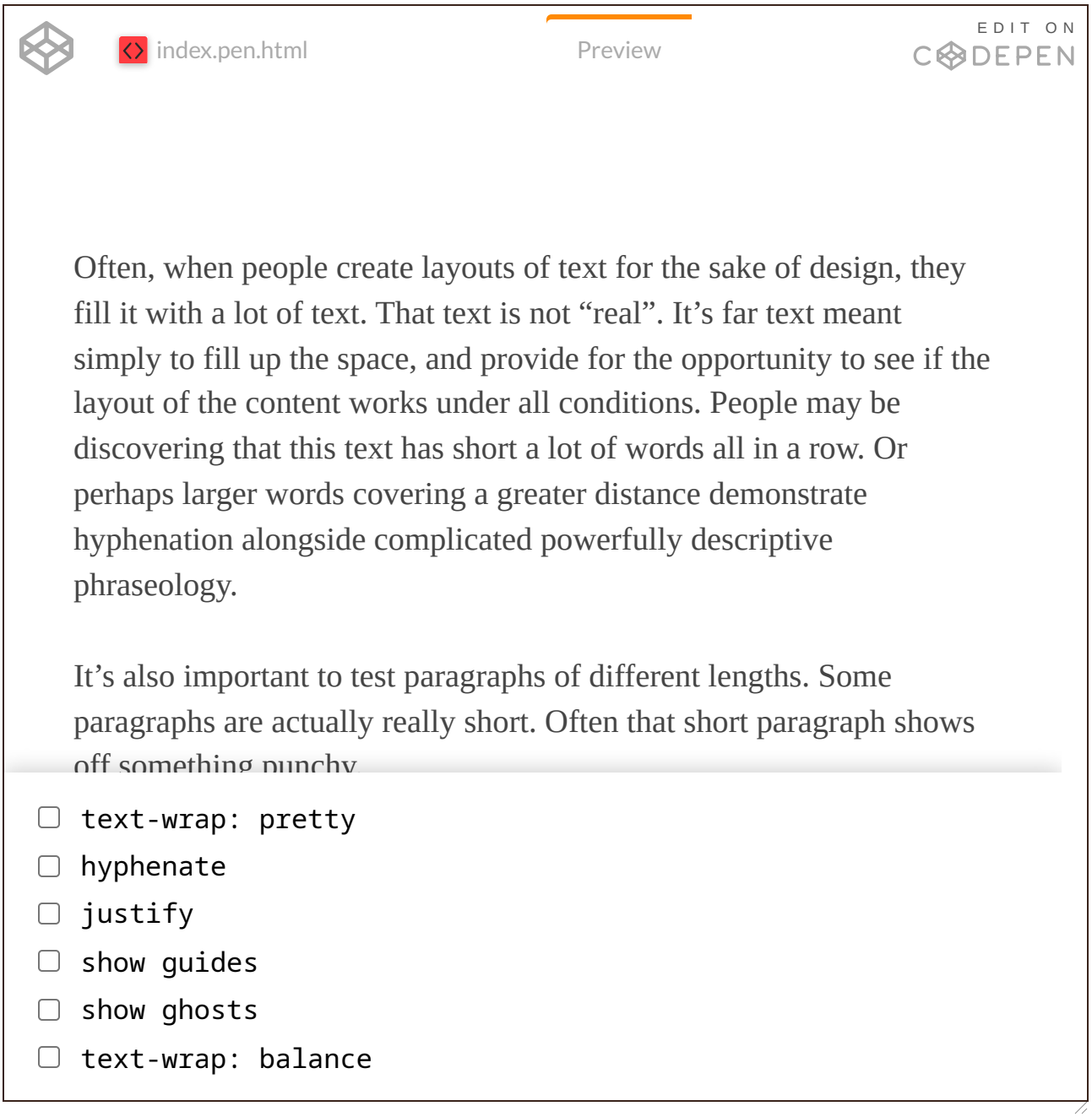
This can be a much nicer default for large `font-size` elements like headers. It also can help with single-word-on-the-next-line orphans, but there is also `text-wrap:`

`pretty;` which can do that, and is designed for smaller-longer text as well, creating better-reading text. Essentially: better typography for free.

Support

Browser Support	`balance` is supported across the board but `pretty` is only Chrome and Safari so far.
Progressive Enhancement	Absolutely. As important as we might agree typography is, without these enhancements the text is still readable and accessible.
Polyfill	There is one for balance.

Usage Example



Resources

- Jen Simmons: Better typography with text-wrap pretty
- Stephanie Stimac: When to use CSS text-wrap: balance; vs text-wrap: pretty;

LINEAR() EASING

What is this?

I think this one a *little* confusing because `linear` as a keyword for `transition-timing-function` or `animation-timing-function` kinda means "flat and boring" (which is sometimes what you want, like when changing opacity for istance). But this `linear()` function *actually* means you're about to do an easing approach that is probably extra fancy, like having a "bouncing" effect.

Why should I care?

Even the fancy `cubic-bezier()` function can only do a really limited bouncing affect with an animation timing, but the sky is the limit with `linear()` because it takes an unlimited number of points.

Support

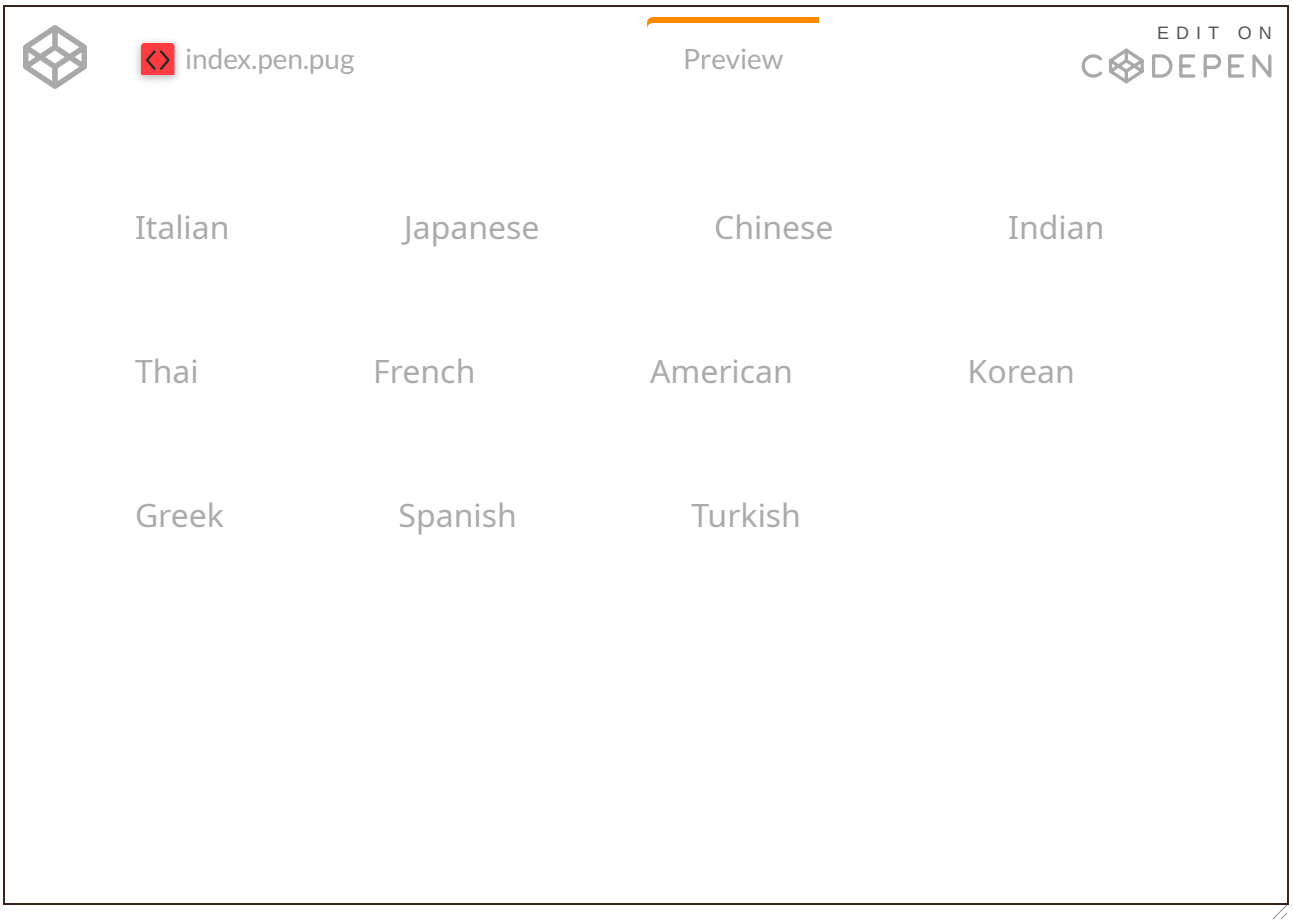
Browser Support	Across the board
Progressive Enhancement	Sure! You could fall back to a named easing value or a `cubic-bezier()`
Polyfill	Not that I know of, but if fancy easing is very important to you, JavaScript libraries like GSAP have this covered in a way that will work in all browsers.

Usage Example

CSS

```
.bounce {
  animation-timing-function: linear(
```

```
    0, 0.004, 0.016, 0.035, 0.063, 0.098, 0.141 13.6%, 0.25, 0.391, 0.563,
0.765,
    1, 0.891 40.9%, 0.848, 0.813, 0.785, 0.766, 0.754, 0.75, 0.754, 0.766,
0.785,
    0.813, 0.848, 0.891 68.2%, 1 72.7%, 0.973, 0.953, 0.941, 0.938, 0.941,
0.953,
    0.973, 1, 0.988, 0.984, 0.988, 1
  );
}
```



Resources

- Jake Archibald: linear() easing generator
- Matthias Martin: Easing Wizard

SHAPE()

What is this?

While CSS has had a `path()` function for a while, it only took a 1-for-1 copy of the `d` attribute from SVG's `<path>` element, which was forced to work only in pixels and has a somewhat obtuse syntax. The `shape()` function is basically that, but fixed up properly for CSS.

Why should I care?

The `shape()` function can essentially draw anything. You can apply it as a value to `clip-path`, cutting elements into any shape, and do so responsively and with all the power of CSS (meaning all the units, custom properties, media queries, etc). You can also apply it to `offset-path()` meaning placement and animation along any drawable path. And presumably soon `shape-outside` as well.

Support

Browser Support	It's in Chrome and Safari and flagged in Firefox, so everywhere fairly soon.
Progressive Enhancement	Probably! Cutting stuff out and moving stuff along paths is usually the stuff of aesthetics and fun and falling back to less fancy options is acceptable.
Polyfill	Not really. You're better off working on a good fallback.

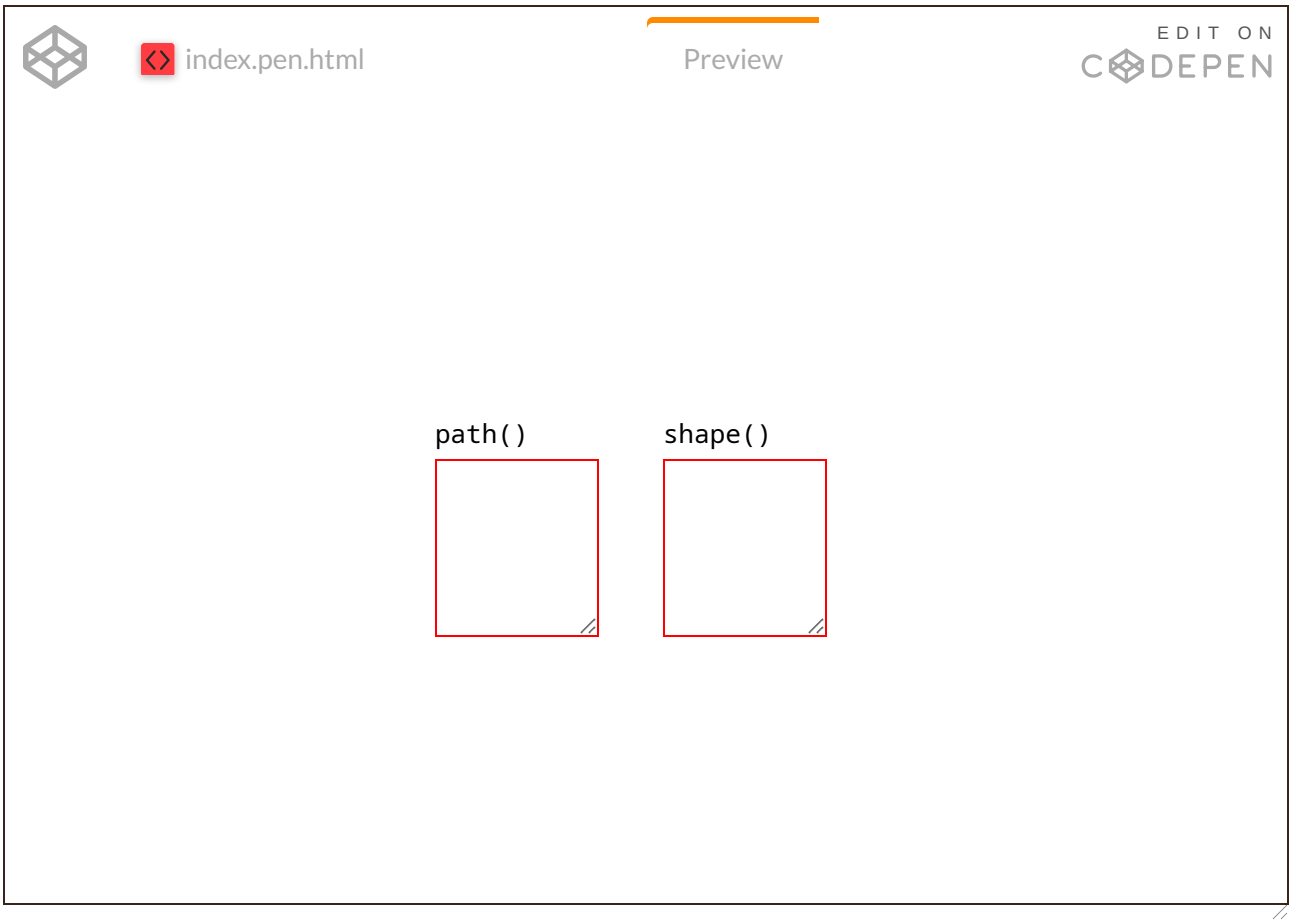
Usage Example

Literally any SVG path can be converted to `shape()`.

CSS

```
.arrow {
  clip-path: shape(
    evenodd from 97.788201% 41.50201%,
    line by -30.839077% -41.50201%,
    curve by -10.419412% 0% with -2.841275% -3.823154% / -7.578137%
-3.823154%,
    smooth by 0% 14.020119% with -2.841275% 10.196965%,
    line by 18.207445% 24.648236%, hline by -67.368705%,
    curve by -7.368452% 9.914818% with -4.103596% 0% / -7.368452% 4.393114%,
    smooth by 7.368452% 9.914818% with 3.264856% 9.914818%,
    hline by 67.368705%, line by -18.211656% 24.50518%,
    curve by 0% 14.020119% with -2.841275% 3.823154% / -2.841275% 10.196965%,
    curve by 5.26318% 2.976712% with 1.472006% 1.980697% / 3.367593%
2.976712%,
    smooth by 5.26318% -2.976712% with 3.791174% -0.990377%, line by
30.735919% -41.357537%,
    curve by 2.21222% -7.082013% with 1.369269% -1.842456% / 2.21222%
-4.393114%,
    smooth by -2.21222% -7.082013% with -0.736024% -5.239556%,
    close
  );
}
```

The natural re-sizeability and more readable syntax is big advantage over `path()`:



MORE POWERFUL ATTR()

What is this?

The `attr()` function in CSS can pull the string value of the matching HTML element. So with `<div data-name="Chris">` I can do `div::before { content: attr(data-name); }` to pull off an use "Chris" as a string. But now, you can apply *types* to the values you pull, making it a lot more useful.

Why should I care?

Things like numbers and colors are a lot more useful to pluck off and use from HTML attributes than strings are.

HTML

```
attr(data-count type(<number>))
```

Support

Browser Support	Chrome only
Progressive Enhancement	It depends on what you're doing with the values. If you're passing through a color for a little aesthetic flourish, sure, it can be a enhancement that fallback to something else or nothing. If it's crucial layout information, probably not.
Polyfill	Not that I know of.

Usage Example

index.html Preview EDIT ON CODEPEN

String Usage

Duplicating text for an aesthetic effect.

Chris Coyier

Layout Control

Passing integer of columns for layout, applying gap automatically.

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Nihil, est blanditiis vero praesentium architecto aliquam sed, deleniti corrupti dolore eos necessitatibus nesciunt eaque nulla omnis. Non fuga ut ad dolore quibusdam error vitae dolorem? Incidunt obcaecati, suscipit dolorum hic quae vitae officiis fugiat qui delectus vero quos facilis veniam itaque laborum nam. Optio mollitia soluta nulla est sed neque hic perspiciatis distinctio repellat obcaecati, iste aliquid tempore exercitationem ex veritatis illum voluptates fuga quis.

Color Usage

Passing a color, which is manipulated into two colors.

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Eligendi voluptatem dignissimos suscipit omnis, dolore, eos itaque vel possimus nesciunt fuga consectetur reprehenderit ipsa laudantium. Necessitatibus voluptatum eligendi quasi a mollitia.

Rounding Font Sizes

Passing an integer font-size, which is rounded to the nearest 5.

Lorem ipsum dolor sit amet consectetur adipisicing elit.

Architecto ipsam enim quam, non tempora sunt veritatis?

Dolor tenetur facilis, tempora consequatur vitae illum delectus!

Porro aliquam dolorem eveniet similique aliquid asperiores repellat.

READING FLOW

What is this?

There are various ways to change the layout such that the visual order no longer matches the source order. The new `reading-order` property allow us to continue to do that while updating the behavior such that tabbing through the elements happens in a predictable manner.

Why should I care?

For a long time we've been told: don't re-order layout! The source order should match the visual order as closely as possible, so that tabbing focus through a page happens in a sensible order. When you mess with the visual order and not source order, tabbing can become zig-zaggy and unpredictable, even causing scrolling, which is a bad experience and a hit to accessibility. Now we can inform the browser that we've made changes and to follow a tabbing order that makes sense for the layout style we're using.

Support

Browser Support	Chrome only
Progressive Enhancement	Not particularly. We should probably not be re-ordering layout wildly until this feature is more safely across all browsers.

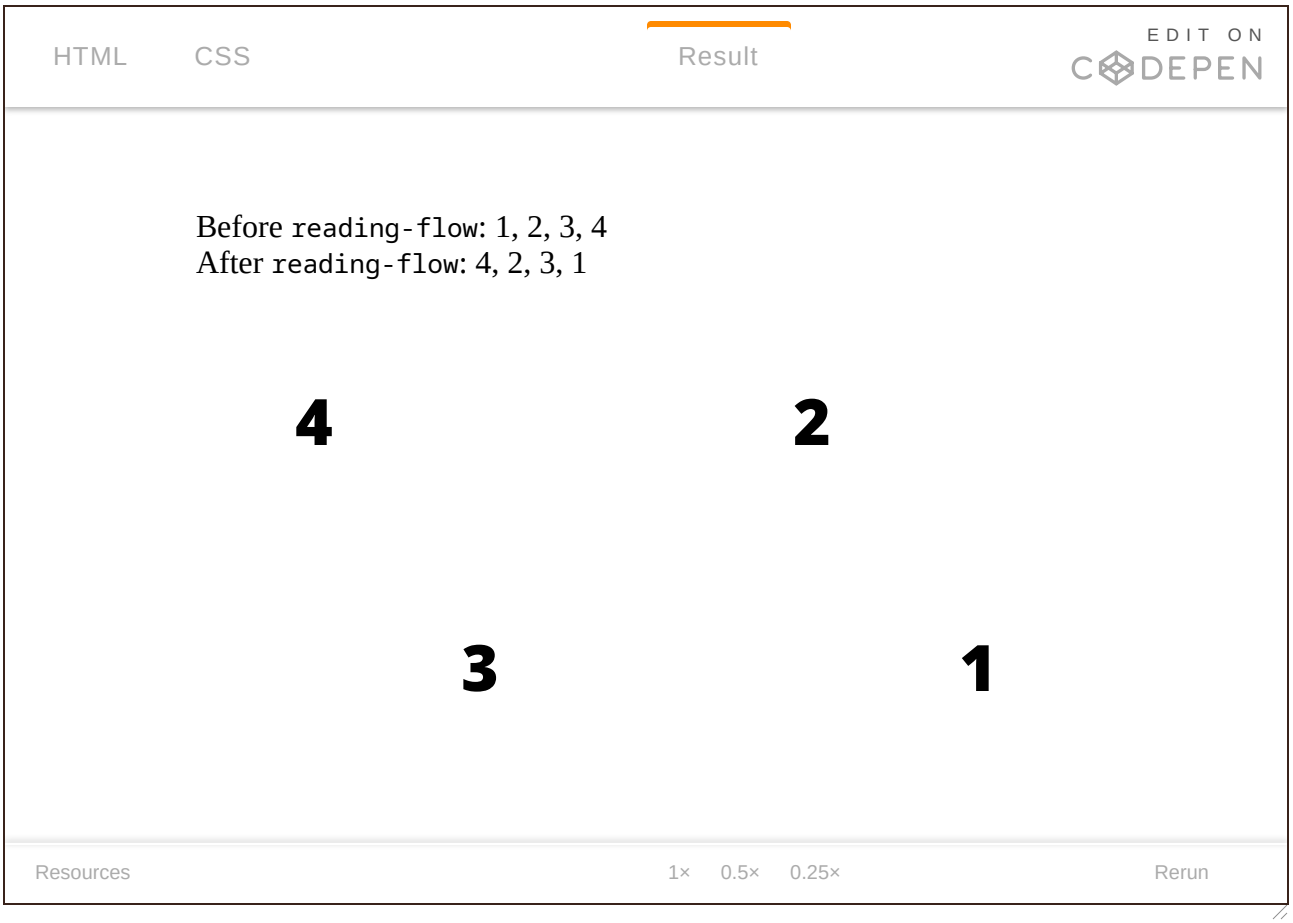
Polyfill	No, but if you were so-inclined you could (hopefully very intelligently) update the `tabindex` property of the elements to a sensible order.

Usage Example

CSS

```
.grid {
  reading-flow: grid-rows;
}
```

Re-ordering a grid layout is perhaps of the most common things to re-order, and having the tabbing order follow the rows after re-arranging is sensible, so that's what the above line of code is doing. But you'll need to set the value to match what you are doing. For instance if you are using flexbox layout, you'd likely set the value to `flex-flow`. See MDN for the list of values.



Resources

- Rachel Andrew: Reading flow ships in Chrome 137
- Daniel Schwarz: What We Know (So Far) About CSS Reading Order
- Di Zhang: Use CSS reading-flow for logical sequential focus navigation

Stuff to Keep an Eye On

- “Masonry” layout, despite having different preliminary implementations, is not yet finalized, but there is enough movement on it it feels like we’ll see that get sorted out next year. The most interesting development at the moment is the proposal of `item-flow` and how that could not only help with Masonry but bring other layout possibilities to other layout mechanisms beyond grid.
- The CSS function `random()` is in Safari and it’s amazing.

- The CSS property `margin-trim` is super useful and we're waiting patiently to be able to use it more than just Safari.
- The `sibling-index()` and `sibling-count()` functions are in Chrome and, for one thing, are really useful for staggered animations.
- For View Transitions, `view-transition-name: match-element;` is awfully handy as it prevents us from needing to generate unique names on absolutely everything. Also — Firefox has View Transitions in development, so that's huge.
- We should be able to use `calc()` to multiply and divide with units (instead of requiring the 2nd to be unitless) soon, instead of needing a hack.
- We never did get "CSS4" (Zoran explains nicely) but I for one still think some kind of named versioning system would be of benefit to *everyone*.
- If you're interested in a more straightforward list of "new CSS things" for say the last ~5 years, Adam Argyle has a great list.

Great Stuff to Remember

- Container queries (and units) are still relatively new and the best thing since media queries in CSS.
- The `:has()` pseudo-class is wildly useful for selecting elements where the children exist or are in a particular state.
- Ultra cool modern CSS features like View Transitions, Anchor Positioning, and Scroll-Driven Animations have all made it to Safari.
- All the useful extra viewport units (shoutout `dvh`) are now in baseline.

9 responses to “What You Need to Know about Modern CSS (2025 Edition)”

NIZAMUDDIN SHAIKH SAYS:

SEPTEMBER 21, 2025 AT 1:06 AM

New pop-ups are amazing. Functions and if conditions open many avenues in CSS writing. Attr, texts, select and all new features enable quality programming.

Reply

FRANK CONIJN SAYS:

SEPTEMBER 27, 2025 AT 3:49 AM

Great article, but have you tried the popover example on an iPad? It doesn’t work on mine, while the iPad is current on updates (iOS 18.6).

Reply

FRANK CONIJN SAYS:

SEPTEMBER 27, 2025 AT 12:31 PM

Please ignore my post because you already mentioned that only Chrome supported it. (On https://caniuse.com/?search=popover it says that Safari supports it as well,

but that's not correct when it comes to iOS.)

Reply

MICHAEL CLAWSON SAYS:

OCTOBER 1, 2025 AT 12:19 AM

My company's traffic is around 90% iOS Safari. Chrome only features are absolutely irresponsible for us to use. I don't believe progressive enhancement or fallbacks or polyfills makes it worth it either. We shouldn't be spending dev time making something look pretty or work better for 5% or less of our customers.

Now if your traffic is more evenly mixed then maybe it makes sense to use them but have fallbacks or polyfills, or use as a progressive enhancement. But I think it still doesn't makes sense to feature them from a broader ecosystem perspective. Chrome is the new IE. It can do what it wants because it has so much market share. A lot of Chrome only features will never become a standard and standards are what we should be building for. Chrome only features belong in the "stuff to keep an eye on section".

That said. There's some cool stuff in here. Thank you for writing them up.

Reply

BENOÎT ROULEAU SAYS:

OCTOBER 4, 2025 AT 9:14 AM

To me, the most interesting thing about `if()` isn't just that it "makes for more readable code and potentially prevents certain types of code repetition", but that it allows setting the value of a property conditionally based on the value of a custom property ON THE SAME ELEMENT (unlike style queries, which can only check property values on a container element).

Reply

CHRIS COYIER SAYS:

OCTOBER 5, 2025 AT 10:26 AM

Interesting. Good observation.

Reply

MAX SAYS:

OCTOBER 20, 2025 AT 5:17 PM

I think it's time for us to consider a bold change in CSS naming. Rather than sticking to 3, 4, 5, and so forth, we could opt for CSSQ. The Q symbolizes queen. I've always regarded CSS as the queen of the web.

Reply

ROBERT BOTTOMLEY SAYS:

NOVEMBER 7, 2025 AT 7:56 PM

If the example for if():

if(
media(max-width > 300px): repeat(2, 1fr);
media(max-width > 600px): repeat(3, 1fr);
media(max-width > 900px): repeat(auto-fit, minmax(250px, 1fr));
else: 1fr;
);

You may want to reverse the matches. You say "first match wins." If so, the 600px and 900px matches will never execute since they are both greater than 300 and that will always win.

Reply

CHRIS COYIER SAYS:

NOVEMBER 8, 2025 AT 1:44 PM

You're 100% right. I also screwed up that media query syntax. It should just be `width` or `inline-size`. So fixed is like:

```
if(
  media(width > 900px): repeat(auto-fit, minmax(200px, 1fr));
  media(width > 600px): repeat(3, 1fr);
  media(width > 300px): repeat(2, 1fr);
  else: 1fr;
);
```

Reply

Leave a Reply

Comment *

Name *

Email *

Website

Save my name, email, and website in this browser for the next time I comment.

Post Comment

$966,000

Frontend Masters donates to open source projects through thanks.dev and Open Collective, as well as donates to non-profits like The Last Mile, Annie Canons, and Vets Who Code.

Contact: support@frontendmasters.com
