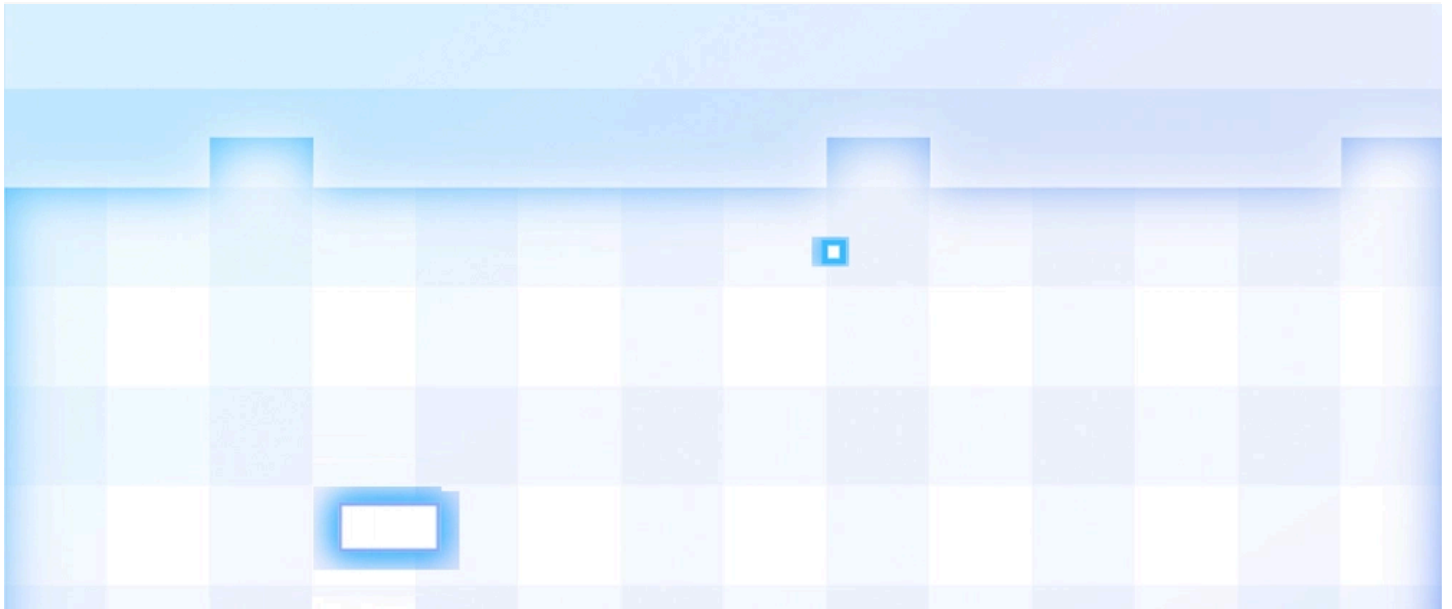




Jane Ori

Posted on 17. Sept. 2023 • Edited on 20. Sept. 2023



78



13



12



6



11

Expert CSS: The CPU Hack

[#css](#) [#howto](#) [#webdev](#) [#tutorial](#)

A "CPU Hack" implies unlocking the ability for continuous crunching of data and re-evaluation of state.

For example, if cyclic vars didn't automatically fail to invalid (`initial`) state in CSS, this would continuously increment the value of `--frame-count` here:

```
body {  
  --input-frame: var(--frame-count, 0);  
  --frame-count: calc(var(--input-frame) + 1);  
}
```

Spoiler alert: You actually *can* do this in CSS, without ever touching JS, I'll show you how!

The 5 Observables

First, let's establish a handful of observations of advanced CSS animation usage so the final demonstration isn't entirely unexpected.

1. Animation State Rules Over All (almost)

The property assignments set by Animation State trump all Selector State property assignments.

body background is always `hotpink` in this example:

```
body {
  animation: example 1s infinite;
  --color: blue;
  background: var(--color);
}
body:hover {
  --color: green;
}
body:has(div:hover) {
  --color: red;
}
@keyframes example {
  0%, 100% { --color: hotpink; }
}
```

This is (partially) why Animation State is not allowed to alter properties that control animations. As in, you cannot^[1] animate the value of `animation-play-state`.

(otherwise, once started, a self-setting animation could only be stopped by JS removal of the element it lives on since the animation could set its own `animation` value and stay alive no matter what other selector states tried to stop it.^[2])

Paused animations are no exception; whatever the value is when it's paused still trumps other states.

[1] Technically there's an unrelated hack that allows this, dealing with inheritance and invalid compute states, but the animation-frame timing of that hack is unreliable for CPU ticking.

[2] This is what went wrong with Ultron.

2. Property Assignments in a Keyframe Can Use var()

```
body {  
  animation: example 1s infinite;  
  --color: blue;  
}  
body:hover {  
  --color: green;  
}  
body:has(div:hover) {  
  --color: red;  
}  
@keyframes example {  
  0%, 100% { background: var(--color); }  
}
```

In this example, the background color is `blue` by default, `green` if hovering, or `red` if hovering specifically within a `div`. The color changes as the user interacts.

3. --var Assignments on Keyframe Results are Cached

We can test this by adding a little indirection to the background color assignment:

```
body {  
  animation: example 1s infinite;  
  --color: blue;  
  
  background: var(--bg);  
}  
body:hover {  
  --color: green;  
}  
body:has(div:hover) {  
  --color: red;  
}  
@keyframes example {  
  0%, 100% { --bg: var(--color); }  
}
```

No matter what, the background is always `blue` because it first evaluated as `blue` and changes to `--color` are not re-computed.

Even if the animation is `paused`, the cached value does not change when states in the background change.

(paused animations use the cached value)

4. Changing An Animation Property Breaks The Cache

By altering the `animation-duration` while the user `:hovers`, the animation cache is recomputed.

```
body {
  animation: example 1s infinite;
  --color: blue;
  background: var(--bg);
}
body:hover {
  --color: green;
  animation-duration: 2s;
}
body:has(div:hover) {
  --color: red;
  animation-duration: 3s;
}
@keyframes example {
  0%, 100% { --bg: var(--color); }
```

The end result here is exactly the same as in #2 above; the background color is `blue` by default, `green` if hovering, or `red` if hovering specifically within a div.

Note: Safari has a bug where it does NOT recompute the cache when an animation property changes, so we've entered Chrome-only territory (firefox can't animate --vars yet)

If we "changed" the `animation-duration` to `1s`, it would not technically change, and the cache does not recompute.

You begin to get interesting behavior if both `:hover` states use the same value that's different from the default state.

```
body {
  animation-duration: 1s;
}
body:hover {
  animation-duration: 2s;
}
```

```
body:has(div:hover) {
  animation-duration: 2s;
}
```

Let's show this one live:

HTML
CSS
Result
EDIT ON

```

body {
  animation: example 1s infinite;
  --color: blue;
  background: var(--bg);
}
body:hover {
  --color: green;
  animation-duration: 2s;
}
body:has(div:hover) {
  --color: red;
  animation-duration: 2s;
}
@keyframes example {
  0%, 100% { --bg: var(--color); }
}

* { margin: 0; font-family: sans-serif; }
body { height: 100vh; }
div {
  padding: 40px;
  background:
}
div:hover {
  background:
}

```

Resources

depending on where your mouse enters the screen (from the top vs bottom), you'll get different colors that "lock" to one or the other until you mouse out.

5. Two Animations

What if instead of pseudo selector state changing the value of `--color`, we made another animation that changed it?

Our `example` animation still sets `--bg` based on `--color`, so we can expect it to still have the same caching behavior.

Changing an animation property of our `example` animation should also, still, cause it to recompute its cache.

So, finally, the `example` animation should accept whatever the current `--color` value is from another animation and cache it with its state.

Here's what that would look like:

```
body {
  animation: color 3s step-end infinite,
    example 1s infinite;

  background: var(--bg);
}
body:hover {
  animation-play-state: running, paused;
}
div::after {
  content: "color preview";
  background: var(--color);
}

@keyframes color {
  0%, 33% { --color: blue; }
  33%, 67% { --color: green; }
  67%, 100% { --color: red; }
}
@keyframes example {
  0%, 100% { --bg: var(--color); }
}
```

Note: Even though we're pausing the `example` animation on `:hover`, that still constitutes a change from the default `running` state so it re-computes and pauses in the same CSS paint frame.

and here's what it feels like:

HTML

CSS

Result

EDIT ON

```
body {
  animation: example 1s infinite,
    color 3s step-end infinite;

  background: var(--bg);
}
body:hover {
  animation-play-state: paused, running;
}
div::after {
  content: "color preview";
  background: var(--color);
}

@keyframes example {
  0%, 100% { --bg: var(--color); }
}
@keyframes color {
  0%, 33% { --color: blue; }
  33%, 67% { --color: red; }
  67%, 100% { --color: green; }
}

* { margin: 0; padding: 0; }
body {
  display: flex;
  align-items: center;
  justify-content: center;
  height: 100vh;
  background-color: #f0f0f0;
}
```

Resources

The bg locks to whatever it was when you entered, then re-computes and re-locks to whatever it is when you leave.

Persistence! NEAT!

The CPU Hack Begins

The previous information implies something extremely interesting; grabbing the cached *value* from an animation does not recompute it, so it shouldn't cause an invalid cyclic state if the source of the cached value is one step removed.

Double capture, compute once, manage the timing... Should be possible.

We have the `example` animation conditionally capturing the value from either normal selector states or from another animation.

Let's imagine it's capturing a number instead of a color, like the `--frame-count` from the opening of this article.

And we'll rename it from `example` to `capture`.

```
body {
  animation: capture 1s infinite;

  --input-frame: 0;
  --frame-count: calc(var(--input-frame) + 1);
}
@keyframes capture {
  0%, 100% { --frame-captured: var(--frame-count); }
}
```

Wouldn't it be great if we could set `--input-frame` to that `--frame-captured` value?

We know that doing it directly would be cyclic because all 3 of these assignments exist in the same frame:

```
--input-frame = --frame-captured
--frame-count = --input-frame + 1
--frame-captured = --frame-count
```

If we capture the captured value though and make sure both captures aren't running at the same time, the capture-capture could hoist that value back to `--input-frame`...

Let's try. We'll call the captured capture `hoist`.

Also, since we don't want them ever running at the same time (because that would definitely be cyclic), let's start them off `paused` to be safe.

```
body {
  animation: hoist 1ms infinite,
    capture 1ms infinite;
  animation-play-state: paused, paused;

  --input-frame: var(--frame-hoist, 0);
  --frame-count: calc(var(--input-frame) + 1);
```

```

}
body::after {
  counter-reset: frame var(--frame-count);
  content: "--frame-count: " counter(frame);
}
@keyframes hoist {
  0%, 100% { --frame-hoist: var(--frame-captured, 0); }
}
@keyframes capture {
  0%, 100% { --frame-captured: var(--frame-count); }
}

```

Now, in order to test this, we also want to set up some dom that we can hover in a specific order to trigger the `animation-play-state` in the right order. No gaps between the elements and we'll give them classes `phase-0`, etc

The first phase is definitely capturing the original output. So we'll keep `hoist` paused and let our old friend `capture` run first:

```

body:has(.phase-0:hover) {
  animation-play-state: paused, running;
}

```

We can stop hovering that element to pause both, which will capture `--frame-count`, or we can go ahead and set up another element to explicitly do that:

```

body:has(.phase-1:hover) {
  animation-play-state: paused, paused;
}

```

And, finally? The moment of truth, test if we can run `hoist` while `capture` is paused, which should give us enough room to avoid the cyclic dependency and plop that output back that the top as input... which should give us our first 2

```

body:has(.phase-2:hover) {
  animation-play-state: running, paused;
}

```

Here it is live, :hover the cursor from top to bottom to complete a loop:

HTMLCSSResultEDIT ON

```
body {
  animation: hoist 1ms infinite,
  capture 1ms infinite;
  animation-play-state: paused, paused;

  --input-frame: var(--frame-hoist, 0);
  --frame-count: calc(var(--input-frame) + 1);
}
body::after {
  counter-reset: frame var(--frame-count);
  content: "--frame-count: " counter(frame);
}
@keyframes hoist {
  0%, 100% { --frame-hoist: var(--frame-captured, 0); }
}
@keyframes capture {
  0%, 100% { --frame-captured: var(--frame-count); }
}
body:has(.phase-
  animation-p'
}
body:has(
  anim
}
br
```

Resources

The CPU Hack

Eureka!

We could make the user pet the dom with their cursor all day orrrr we could move the dom under the cursor the moment it needs to be to automatically trigger :hover

Let's figure that out!

We'll need hovering `.phase-0` to automatically "goto" `.phase-1`, then hovering that will "goto" `.phase-2` ...

And then hovering `.phase-2` needs to return to a `paused, paused` state to avoid any single frame running a compute for both animations at the same time.

Remember: playing or pausing an animation causes it to re-compute on that frame, so changing from `running, paused` straight to `paused, running` is actually, for one frame, running both at the same time.

So we need to "goto" a state that will then "goto" `.phase-0`. Since `.phase-1` is `paused, paused` and "goto" 2, we'll duplicate it and make the new `.phase-3` also pause both but "goto" 0 instead.

Let's add this CSS to what we had:

```
body:has(.phase-3:hover) {  
  animation-play-state: paused, paused;  
}
```

And we'll use this for the HTML:

```
<ol class="cpu">  
  <li class="phase-0"></li>  
  <li class="phase-1"></li>  
  <li class="phase-2"></li>  
  <li class="phase-3"></li>  
</ol>
```

▼ Here's a recap of each phase if interested

- `.phase-0` (`hoist paused, capture running`)
 1. Hoist Value is Frozen
 2. Assign Captured = Output Value
 3. GOTO `.phase-1`
- `.phase-1` (`hoist paused, capture paused`)
 1. Hoist Value is Frozen
 2. Assign Captured Value = Output Value
 3. Freeze Captured (at the end of this css paint frame)
 4. GOTO `.phase-2`
- `.phase-2` (`hoist running, capture paused`)
 1. Captured Value is Frozen
 2. Assign Hoist Value = Captured Value
 3. GOTO `.phase-3`
- `.phase-3` (`hoist paused, capture paused`)
 1. Captured Value is Frozen

2. Assign Hoist Value = Captured Value
3. Freeze Hoist (at the end of this css paint frame)
4. GOTO `.phase-0`

Next, we'll style this `.cpu` element so each of its children take up its whole area when their width becomes `100%`, z-stacked on top of each other in dom order.

```
.cpu { position: relative; list-style: none; }  
.cpu > * {  
  position: absolute;  
  inset: 0px;  
  width: 0px;  
}  
.cpu > .phase-0 { width: 100%; }  
.cpu > .phase-0:hover + .phase-1 { width: 100%; }  
.cpu > .phase-1:hover + .phase-2 { width: 100%; }  
.cpu > .phase-2:hover + .phase-3 { width: 100%; }
```

This should be the final piece; each phase triggers the next and they'll only happen for one CSS Paint Frame each. Let's see it live!

HTML
CSS
Result
EDIT ON

```

body {
  animation: hoist 1ms infinite,
  capture 1ms infinite;
  animation-play-state: paused, paused;

  --input-frame: var(--frame-hoist, 0);
  --frame-count: calc(var(--input-frame) + 1);
}
body::after {
  counter-reset: frame var(--frame-count);
  content: "--frame-count: " counter(frame);
}
@keyframes hoist {
  0%, 100% { --frame-hoist: var(--frame-count, 0); }
}
@keyframes capture {
  0%, 100% { --frame-count: var(--frame-count); }
}
@property --frame-count {
  syntax: "<integer";
  initial-value: 0;
  inherits: false;
}

```

Resources

Note: we also needed to register the output var (`--frame-count`) or else it will suddenly stop working at 100 because of `calc()` technically becoming nested each iteration. Casting it to integer prevents this and is much more efficient. The demo above has the `@property` code included.

Also, technically, one small cleanup you could make:

Drop the `--input-frame` var, just `--frame-hoist` directly, it's cleaner.

The Rest of the Owl

So you have a CPU in CSS. What can you do with it?

100% CSS Compute Integer `--width` and `--height` of the Screen

100% CSS Image-Mouse-Coordinate Zoom on Hover

100% CSS Conway's Game of Life Simulator - Infinite Generations, 42x42

100% CSS Breakout, play it here:

HTML

CSS

Result

EDIT ON

```
@import url("https://propjockey.github.io/css-breakout/breakout.css");

/*
// https://github.com/propjockey/css-breakout/
*/
```

Resources

The End!

If you think this is useful, fun, or interesting, it's the kind of thing I do in my free time! So please do consider following me here, [on CodePen](#), and [on X](#) as well!



// Jane Ori

PS: I've been laid off recently and am looking for a job!

<https://linkedin.com/in/JaneOri>

Over 13 years of full stack (mostly JS) engineering work and consulting, ready for the right opportunity!

DEV

The DEV Team

PROMOTED

Dev Signal Multi-Agent Architecture (ADK & MCP)

ADK & FastAPI App Container

Root Orchestrator Agent (Strategist)

Reddit Scanner Agent (Discovery)

GCP Expert Agent (Researcher)

Blog Drafter Agent (Creative Synthesizer)

Delegates Task

Agent Response / Data

Delegates Task

Agent Response / Data

Delegates Task

Agent Response / Data

Gemini 3 Flash Model (Vertex AI)

Reddit MCP

Reddit MCP

Developer Knowledge MCP

Google Search Tool

Developer Knowledge MCP

Nano Banana MCP (Local Image Gen)

Reddit API

Google Cloud Docs

Web Search

Generated Infographic Header

Shared State (technical_research_findings)

User

Chat Request

Final Response (Blog Post + Image)

Building Capabilities for a Multi-Agent System with Google ADK, MCP, and Cloud Run

My team's mission is to accelerate the developer journey from writing code to running secure AI workloads on Google Cloud. To help developers succeed, we focus on identifying their most pressing questions and building demos that provide straightforward, easy-to-implement solutions.

Read more →

Top comments (12)



Richie Permana

• 18. Sept. 23

×

This is so next level🤔 I almost understand nothing.



Massimo Artizzu • 21. Sept. 23



I... think I'll bookmark this article to read it later... again and again...
Advanced stuff ahoy! 🦄



Jane Ori 🌟 • 21. Sept. 23



Awesome, that's what I do with pages of the CSS spec! :D



Philip Warkentien II • 27. März 24



I am in shock! First, just the in-between tricks are awesome enough
(Persistence, Incremental timer). And, of course, I'm still trying to wrap my
head around the 4-tile fine-position detection with a CSS-only-binary-
search-loop-compute-hack (or something like that).

Amazing project!



Jane Ori 🌟 • 28. März 24



your gecko pen is absolutely adorable and I enjoyed your article on it!

truly delighted by the geckos ^^

glad you enjoyed my work too; thank you for checking it out!



Sebastian G. Marinescu • 18. Sept. 23



WOW, you are a wizard ❤️👍



Den.Dionigi • 25. Sept. 23



Woha, this is deep!



@leptoquark1 • 25. Sept. 23



Bleeding edge stuff here. I like it



Temani Afif • 21. Sept. 23



and I was going to miss the only interesting CSS article since too long because you forgot the tag .. 🤔



Jane Ori 🌟 • 21. Sept. 23



Remembering is so much more a psychotic activity than forgetting.

DEV should blink warnings at you before publishing! >.<;



Šime Vidas • 17. Sept. 23



depending on where your mouse enters the screen (from the top vs bottom), you'll get different colors that "lock" to one or the other until you mouse out.

Not in Safari and Firefox.



Jane Ori 🌟 • 17. Sept. 23



Note: Safari has a bug where it does NOT recompute the cache when an animation property changes, so we've entered Chrome-only territory (firefox can't animate --vars yet)

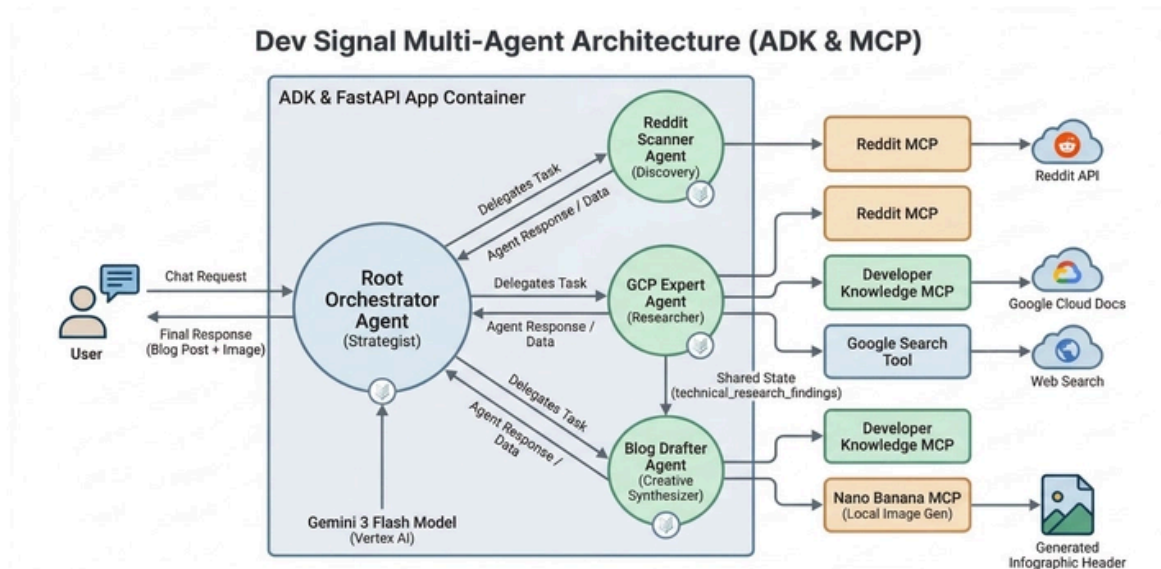
:)

[View full discussion \(12 comments\)](#)

[Code of Conduct](#) • [Report abuse](#)

DEV The DEV Team PROMOTED

...



Architect A Personalized Multi-Agent System with Long-Term Memory

In support of our mission to accelerate the developer journey on Google Cloud, we built Dev Signal — a multi-agent system designed to transform raw community signals into reliable technical guidance by automating the path from discovery to expert creation.



[Load more →](#)

🔧: css-media-vars, augmented-ui, CSS-Sweeper, Space Toggles, & more.

LOCATION

janeori.eth 🦋 Hello World

PRONOUNS

she/her

JOINED

11. Jan. 2022

More from [Jane Ori](#)

CSS - The single coolest API technique you can imagine.

#css #api #webdev

100% CSS: repeat(--n, anything)

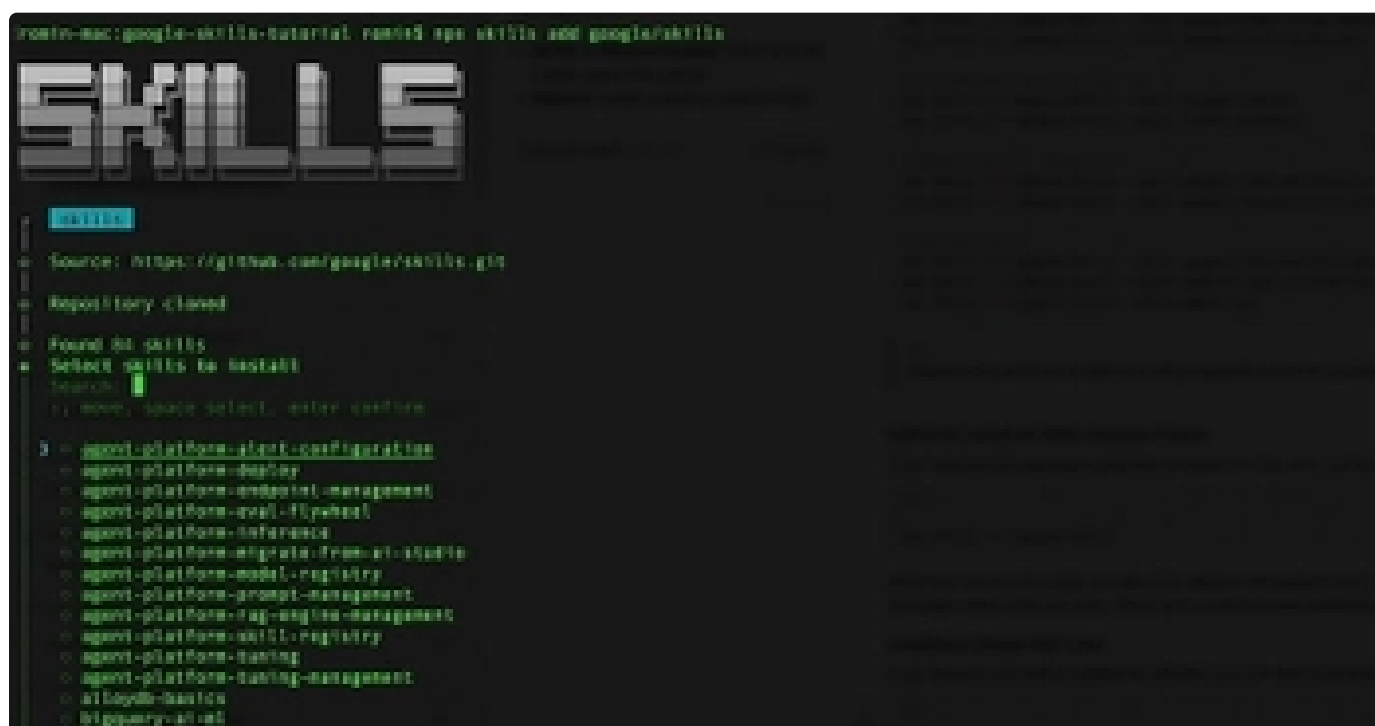
#css #customcssfunctions #webdev #productivity

CSS Breakpoint Units - design with pixels and get fluid UX for free while automatically solving two of the oldest accessibility problems.

#css #design #designsystem #a11y

DEV The DEV Team PROMOTED

...



70+ official Google Cloud Skills, and the 9 you actually need to start

The open-source google/skills repo spans eight categories, from BigQuery and Cloud Run to WAF security audits. Rather than installing everything, this tutorial narrows it to nine skills, shows the npx commands to add them, and explains the progressive disclosure model that keeps your agent's context window from drowning.

Read more →