# CSS FEATURES IN 2023 AND BEYOND A BRIEF SUMMARY

AHMET BUGRA YIGITER

# ABOUT ME

Hipolabs

Pera Wallet

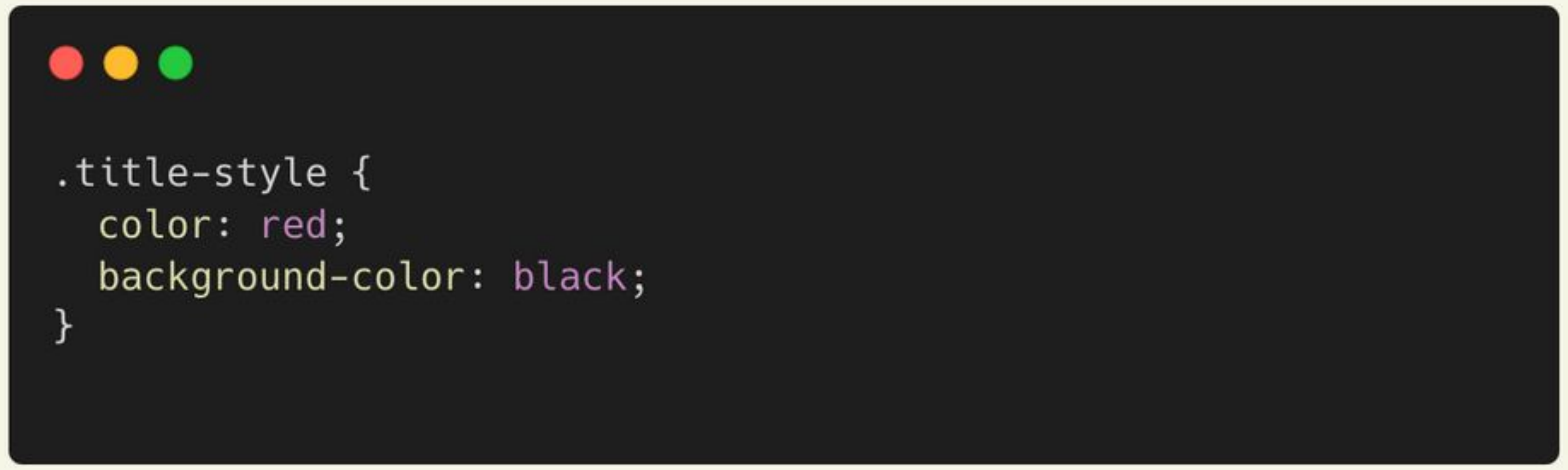
Crea

Horizon Blockchain

## LET'S DIVE IN!

# CSS... WHAT?

Cascading Style Sheets is a powerful language for styling web pages and user interfaces. It is constantly evolving, with new features being added all the time. In 2023, there are a number of new CSS features that are particularly exciting for web UI development.



```css
.title-style {
  color: red;
  background-color: black;
}
```

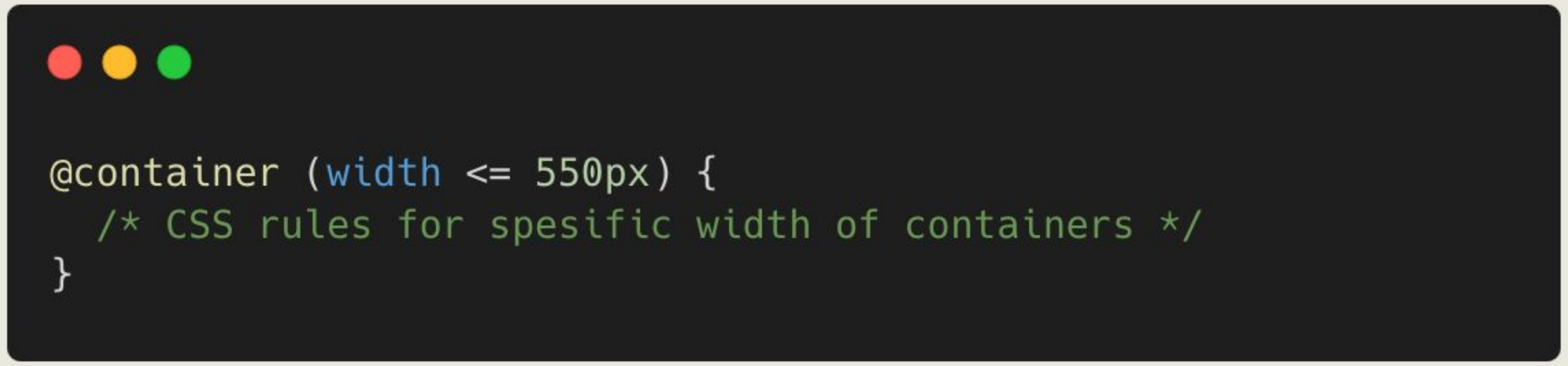
# NEW CSS FEATURES IN 2023

- Container queries
- Style queries
- :has() selector
- nth-of microsyntax
- text-wrap: balance
- initial-letter
- Dynamic viewport units
- Wide-gamut color spaces
- color-mix()
- Nesting
- Cascade layers
- Scoped styles
- Trigonometric functions
- Individual transform properties
- popover
- anchor positioning
- selectmenu
- Discrete property transitions
- Scroll-driven animations
- View transitions

Browser Support 105 105 110 16

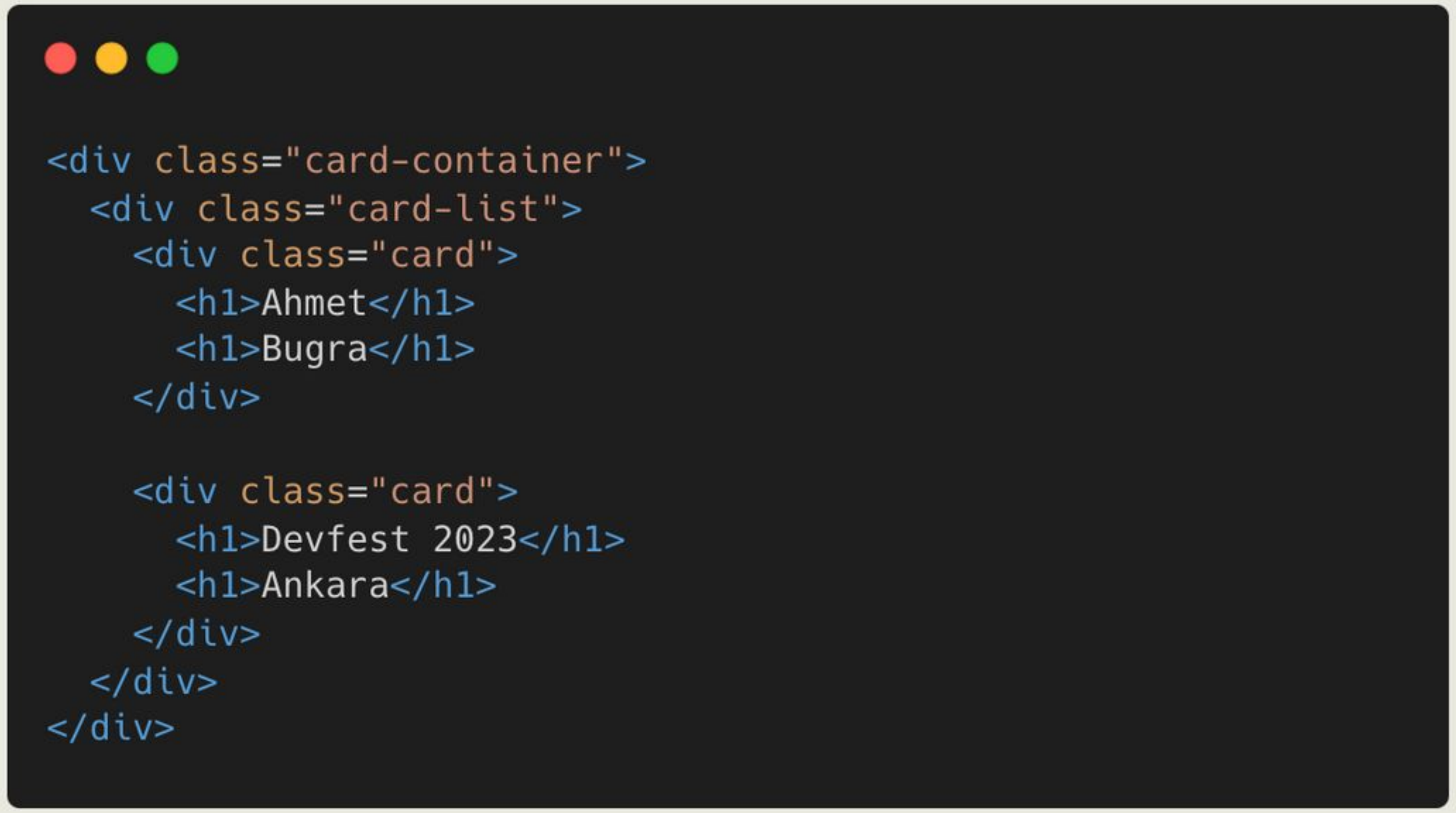
## CONTAINER QUERIES

One of the most significant new features is container queries. Container queries allow you to style elements based on the size of their parent container. This gives you more control over your layout and makes it easier to create responsive designs.

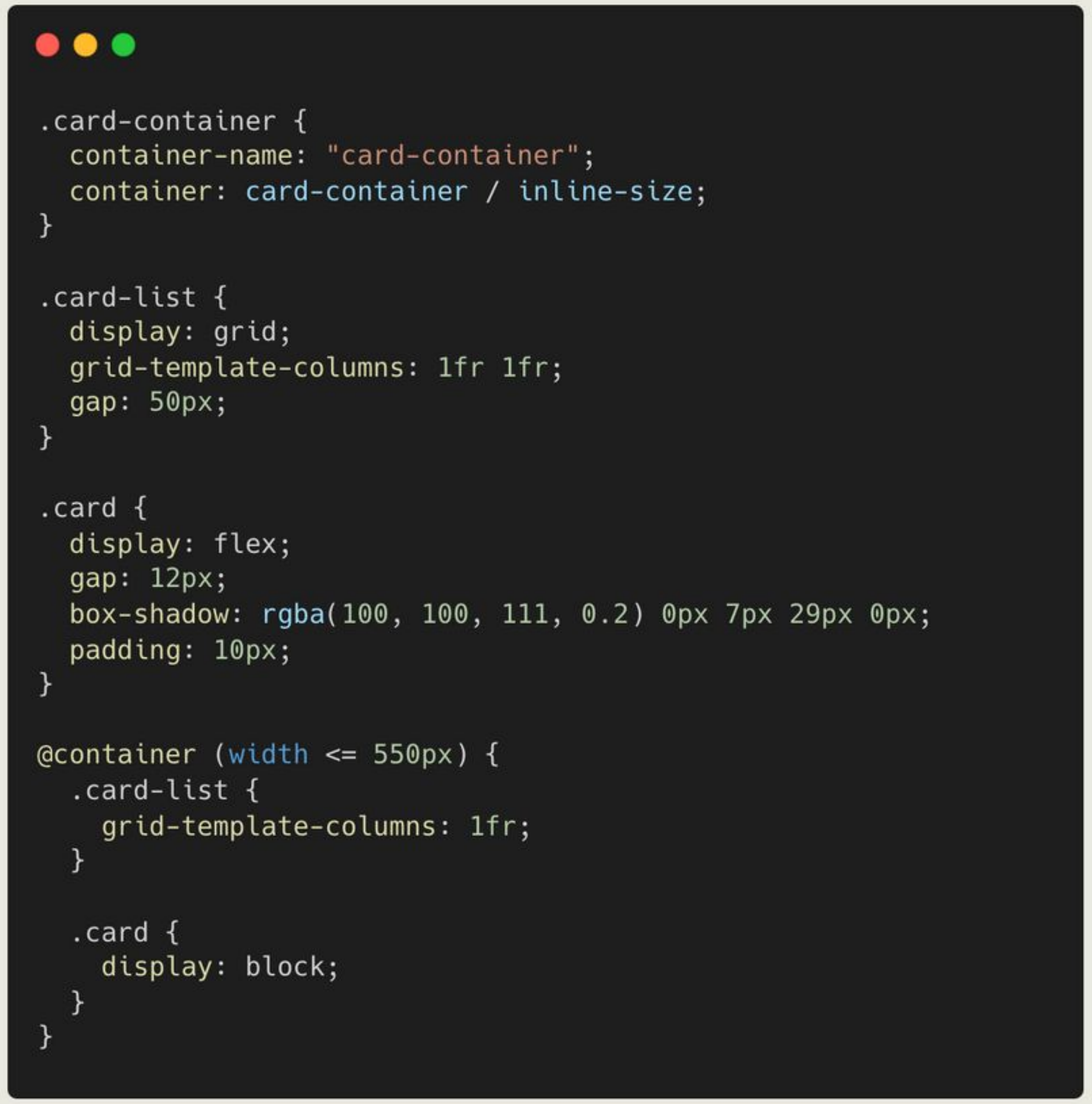


```css
@container (width <= 550px) {
  /* CSS rules for spesific width of containers */
}
```

# LET'S MAKE DEMO!



```html
<div class="card-container">
  <div class="card-list">
    <div class="card">
      <h1>Ahmet</h1>
      <h1>Bugra</h1>
    </div>

    <div class="card">
      <h1>Devfest 2023</h1>
      <h1>Ankara</h1>
    </div>
  </div>
</div>
```

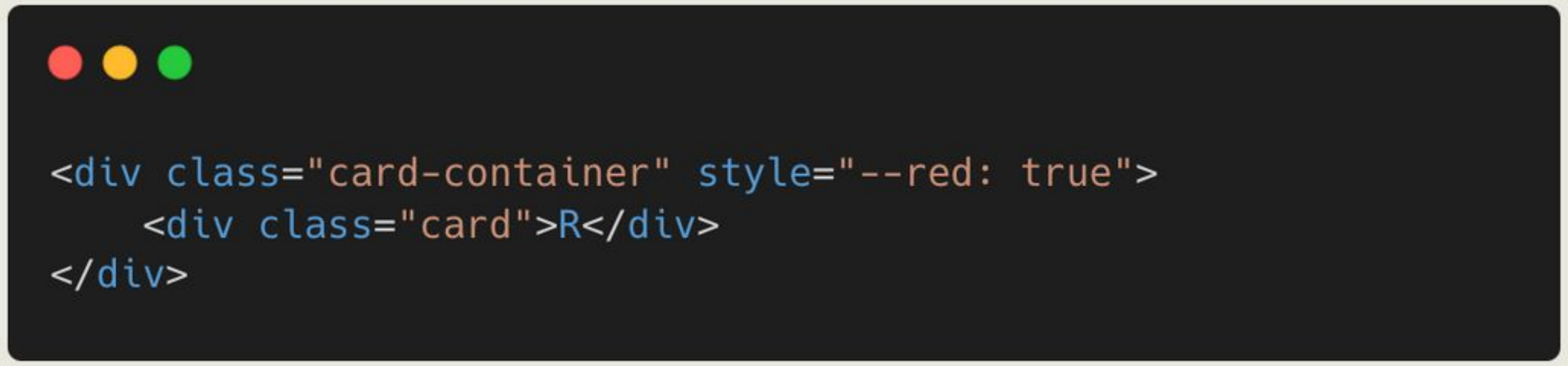


```css
.card-container {
  container-name: "card-container";
  container: card-container / inline-size;
}

.card-list {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 50px;
}

.card {
  display: flex;
  gap: 12px;
  box-shadow: rgba(100, 100, 111, 0.2) 0px 7px 29px 0px;
  padding: 10px;
}

@container (width <= 550px) {
  .card-list {
    grid-template-columns: 1fr;
  }

  .card {
    display: block;
  }
}
```

Browser Support 111 111 X X

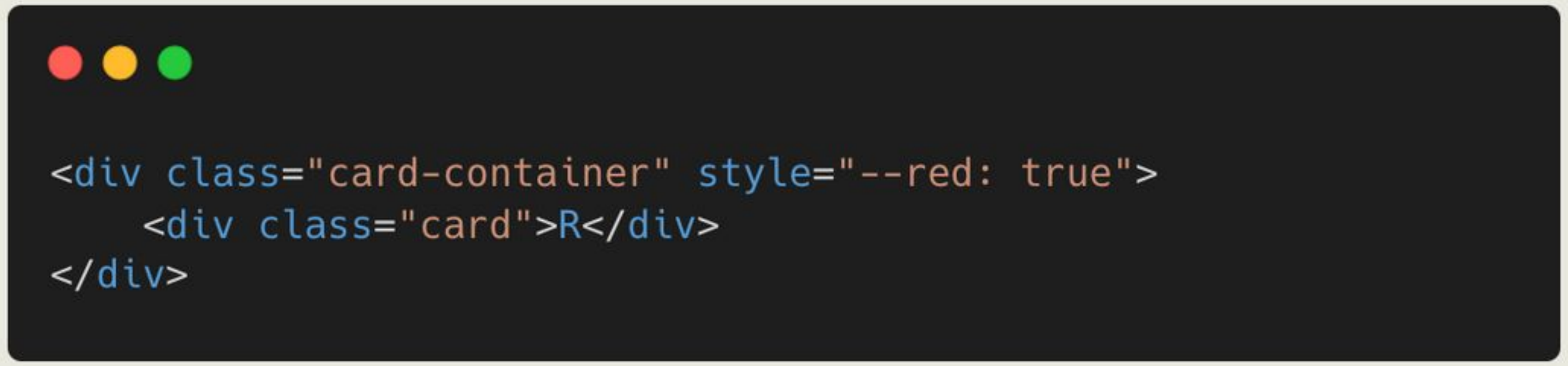
## STYLE QUERIES

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.

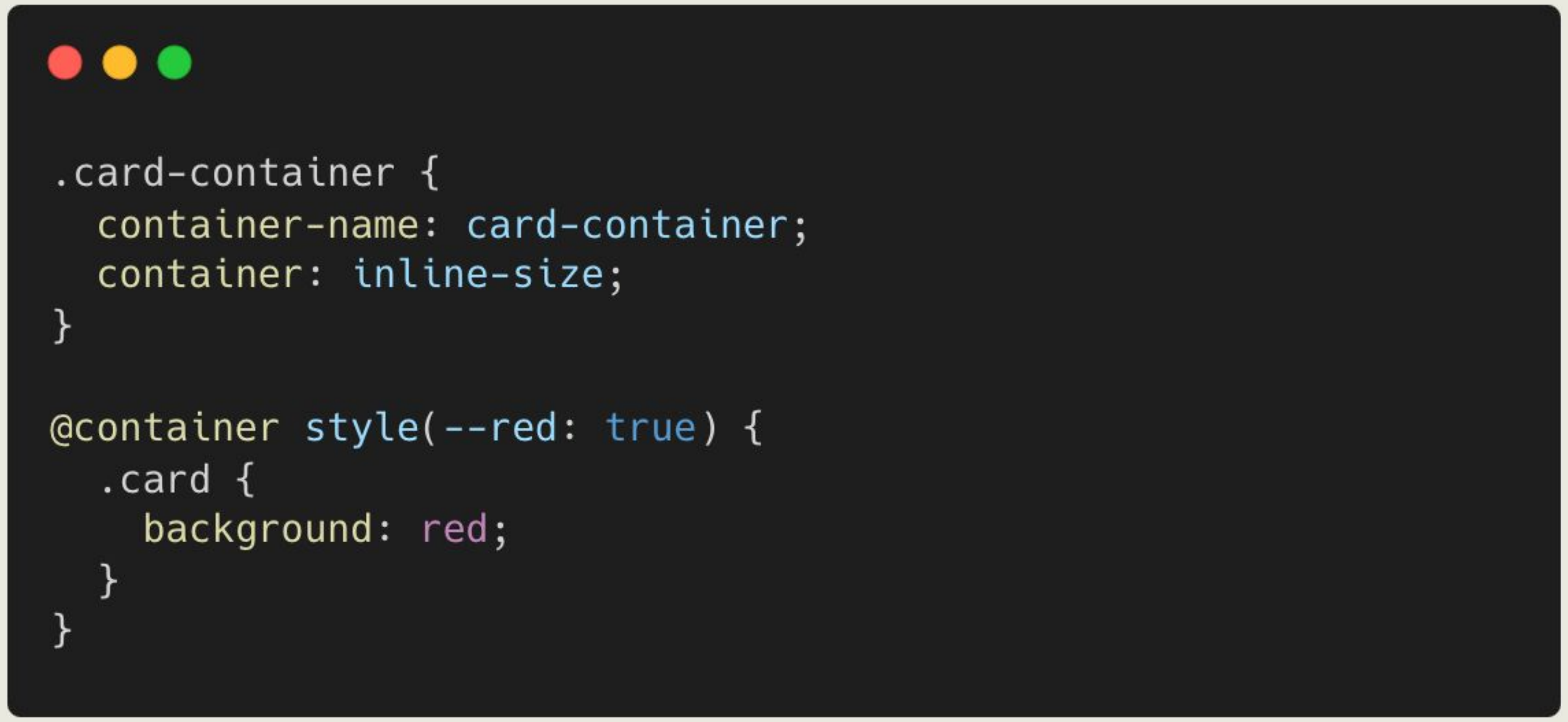


```
<div class="card-container" style="--red: true">
    <div class="card">R</div>
</div>
```

## LET'S MAKE DEMO!



```html
<div class="card-container" style="--red: true">
    <div class="card">R</div>
</div>
```

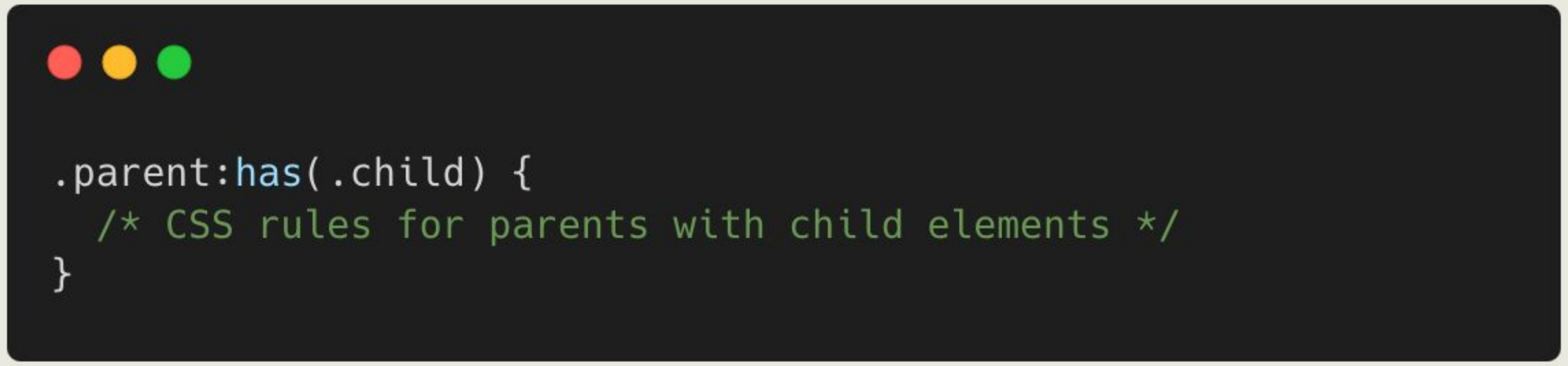


```css
.card-container {
  container-name: card-container;
  container: inline-size;
}

@container style(--red: true) {
  .card {
    background: red;
  }
}
```

Browser Support 105 105 121 15.4

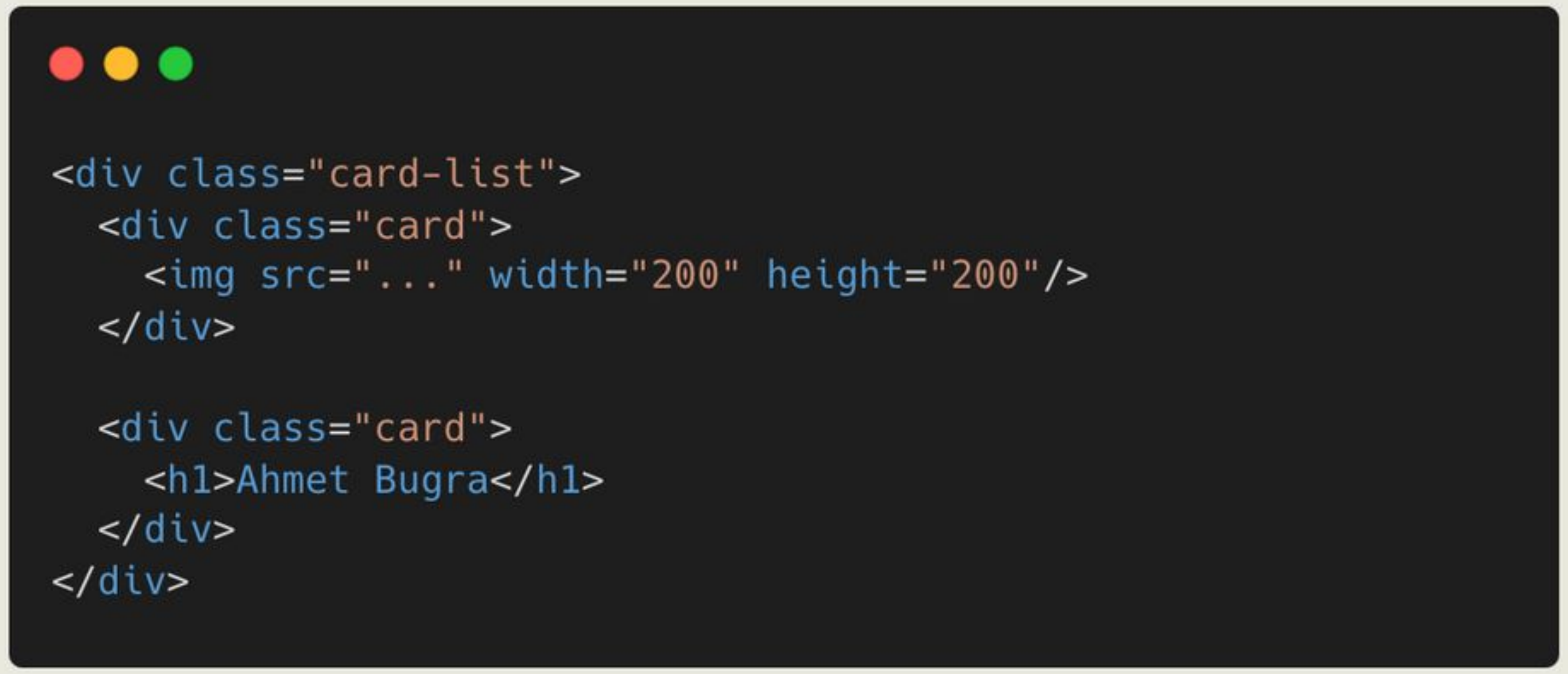
## :HAS()

Another new feature is the :has() selector. This selector allows you to style elements based on the presence of specific child elements. This can be used to create more complex and dynamic UIs.

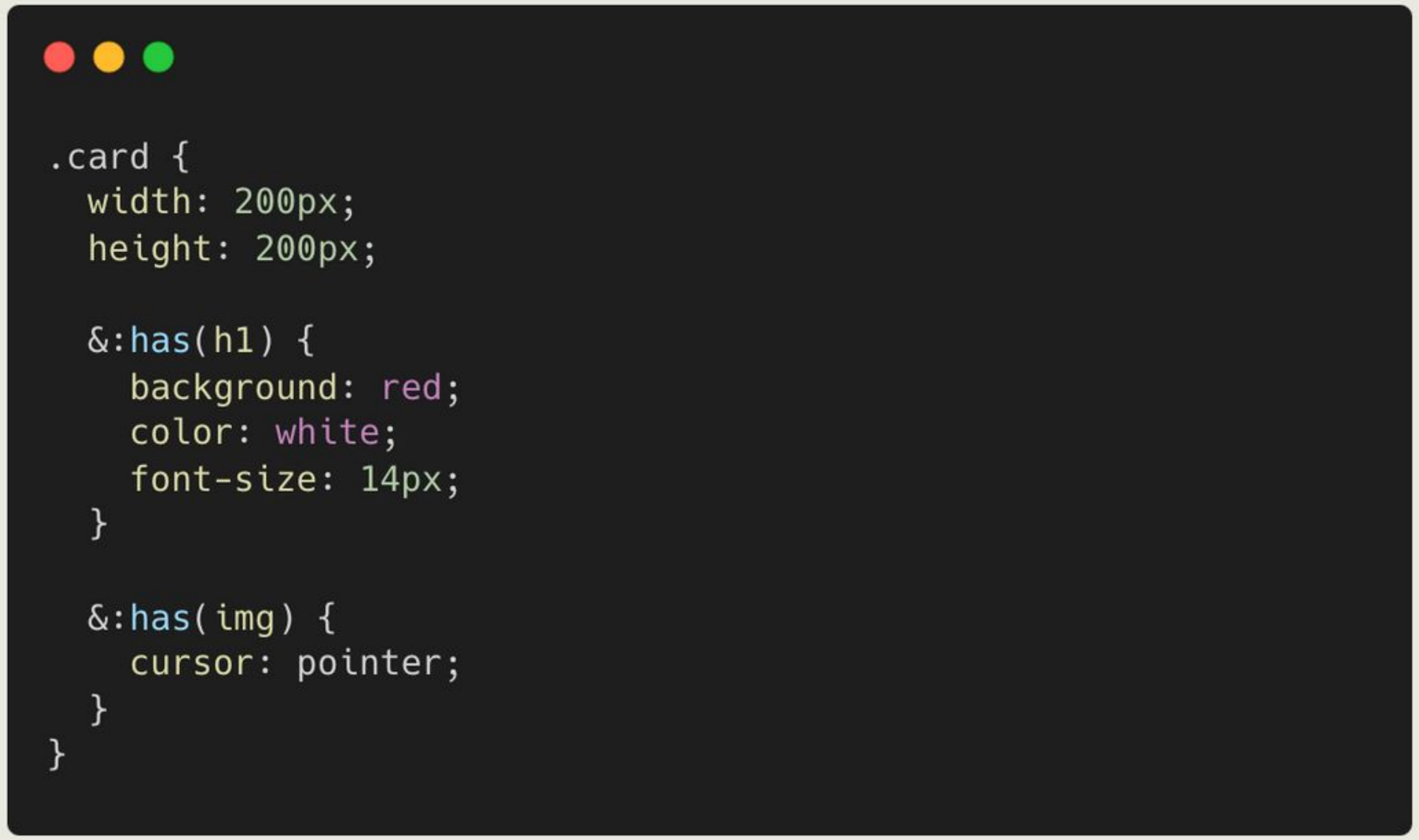


```css
.parent:has(.child) {
  /* CSS rules for parents with child elements */
}
```

## LET’S MAKE DEMO!



```html
<div class="card-list">
  <div class="card">
    <img src="..." width="200" height="200"/>
  </div>

  <div class="card">
    <h1>Ahmet Bugra</h1>
  </div>
</div>
```

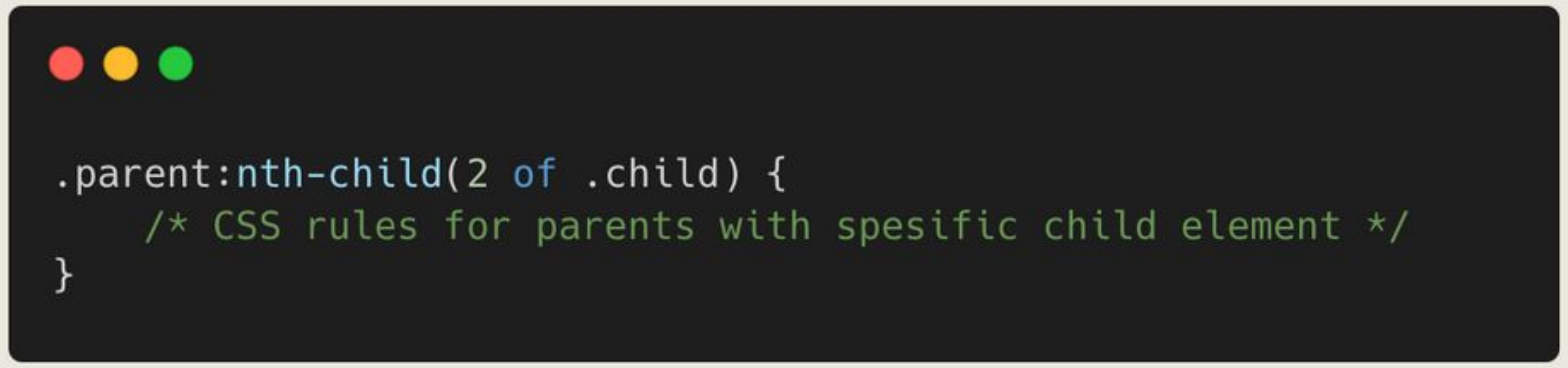


```css
.card {
  width: 200px;
  height: 200px;

  &:has(h1) {
    background: red;
    color: white;
    font-size: 14px;
  }

  &:has(img) {
    cursor: pointer;
  }
}
```

Browser Support 111 111 113 9

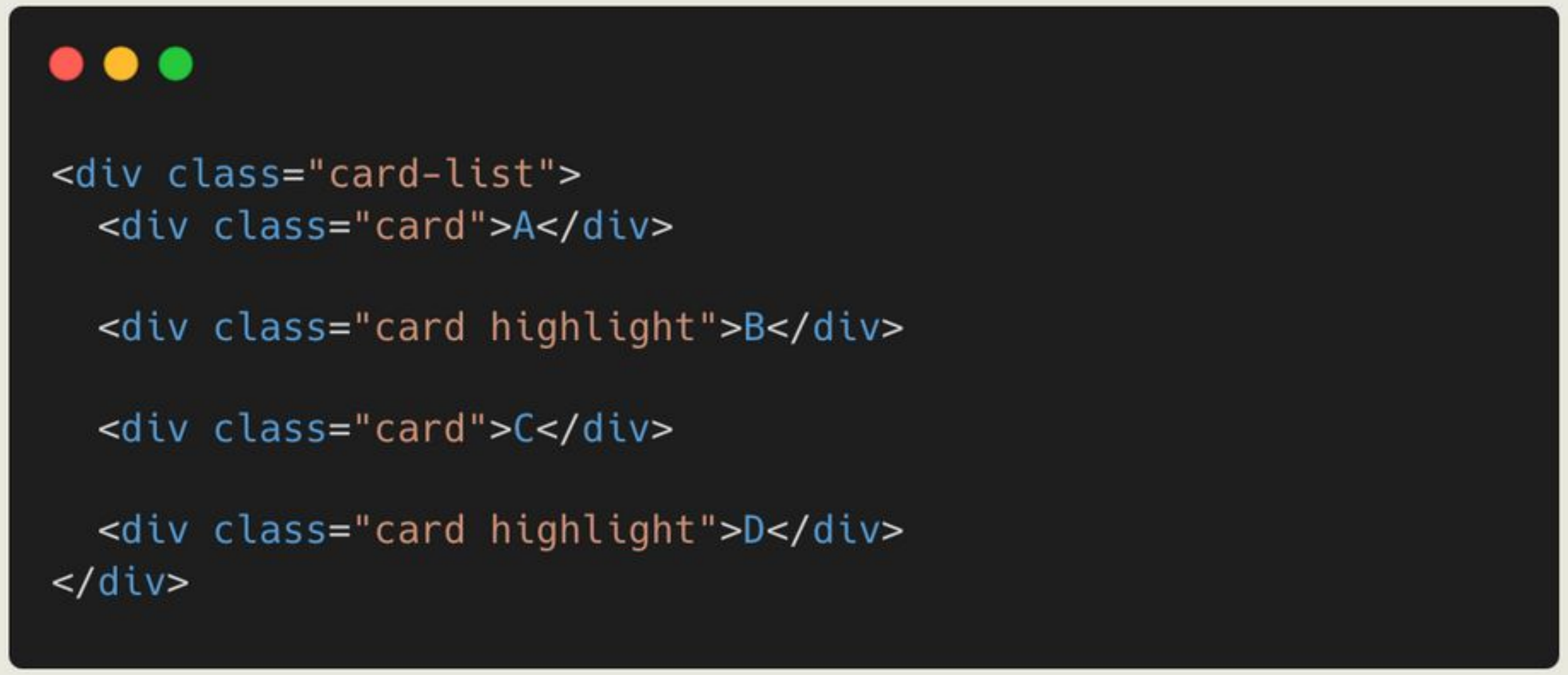
## :NTH-CHILD()

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.

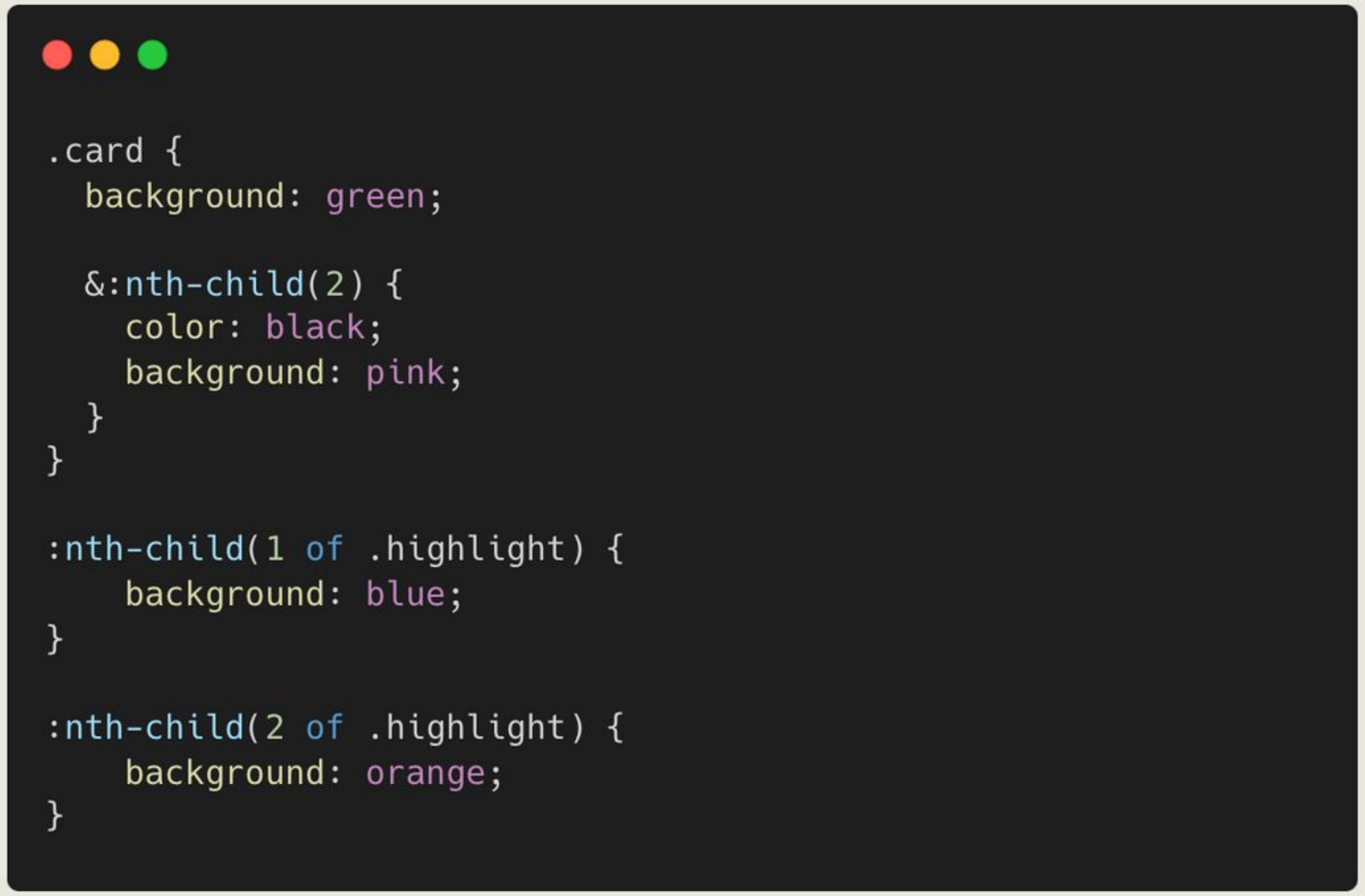


```css
.parent:nth-child(2 of .child) {
    /* CSS rules for parents with spesific child element */
}
```

# LET'S MAKE DEMO!



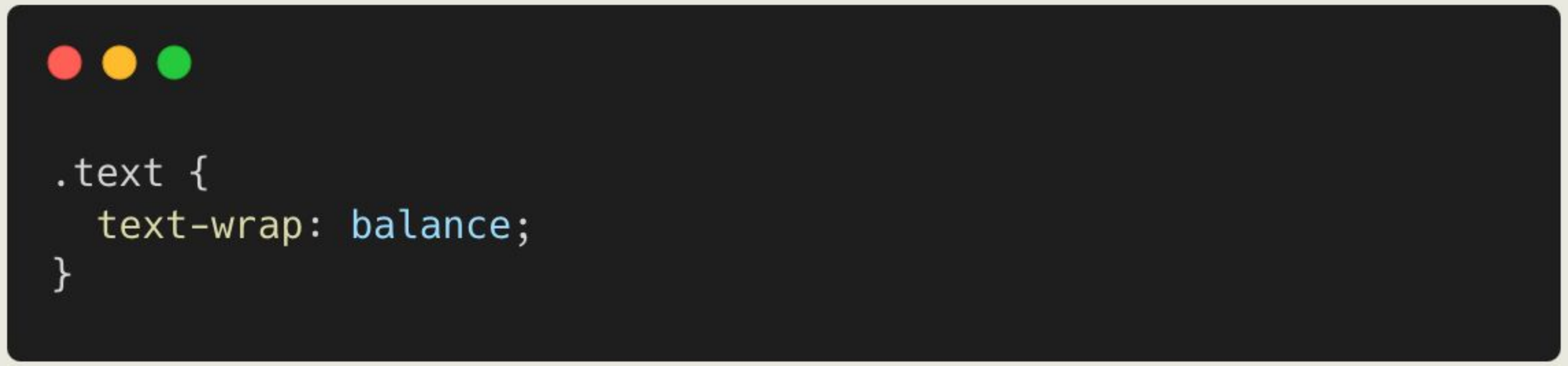
```html
<div class="card-list">
  <div class="card">A</div>

  <div class="card highlight">B</div>

  <div class="card">C</div>

  <div class="card highlight">D</div>
</div>
```



```css
.card {
  background: green;

  &:nth-child(2) {
    color: black;
    background: pink;
  }
}

:nth-child(1 of .highlight) {
    background: blue;
}

:nth-child(2 of .highlight) {
    background: orange;
}
```

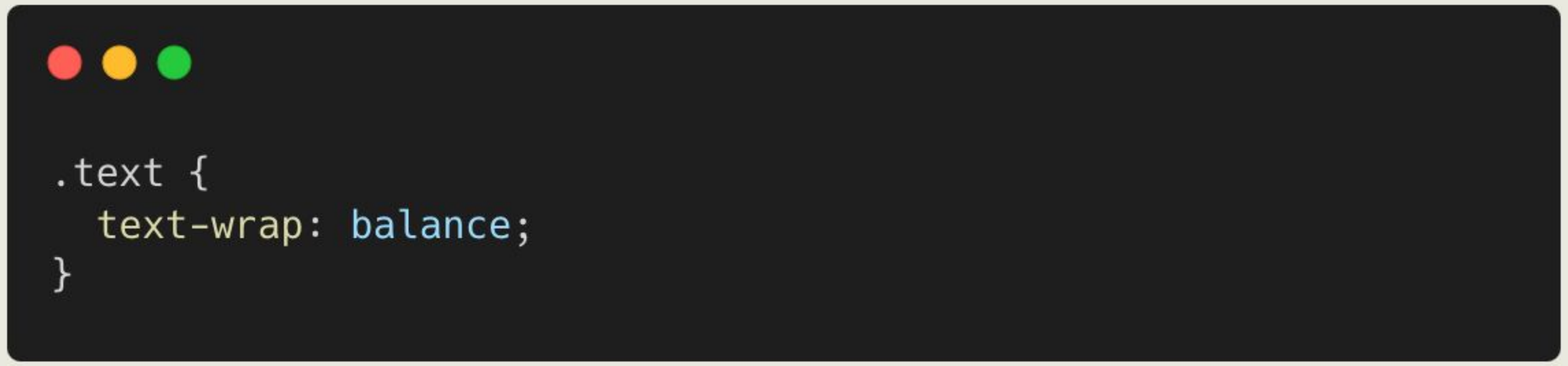
## TEXT-WRAP: BALANCE

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.



```css
.text {
  text-wrap: balance;
}
```

## LET'S MAKE DEMO!

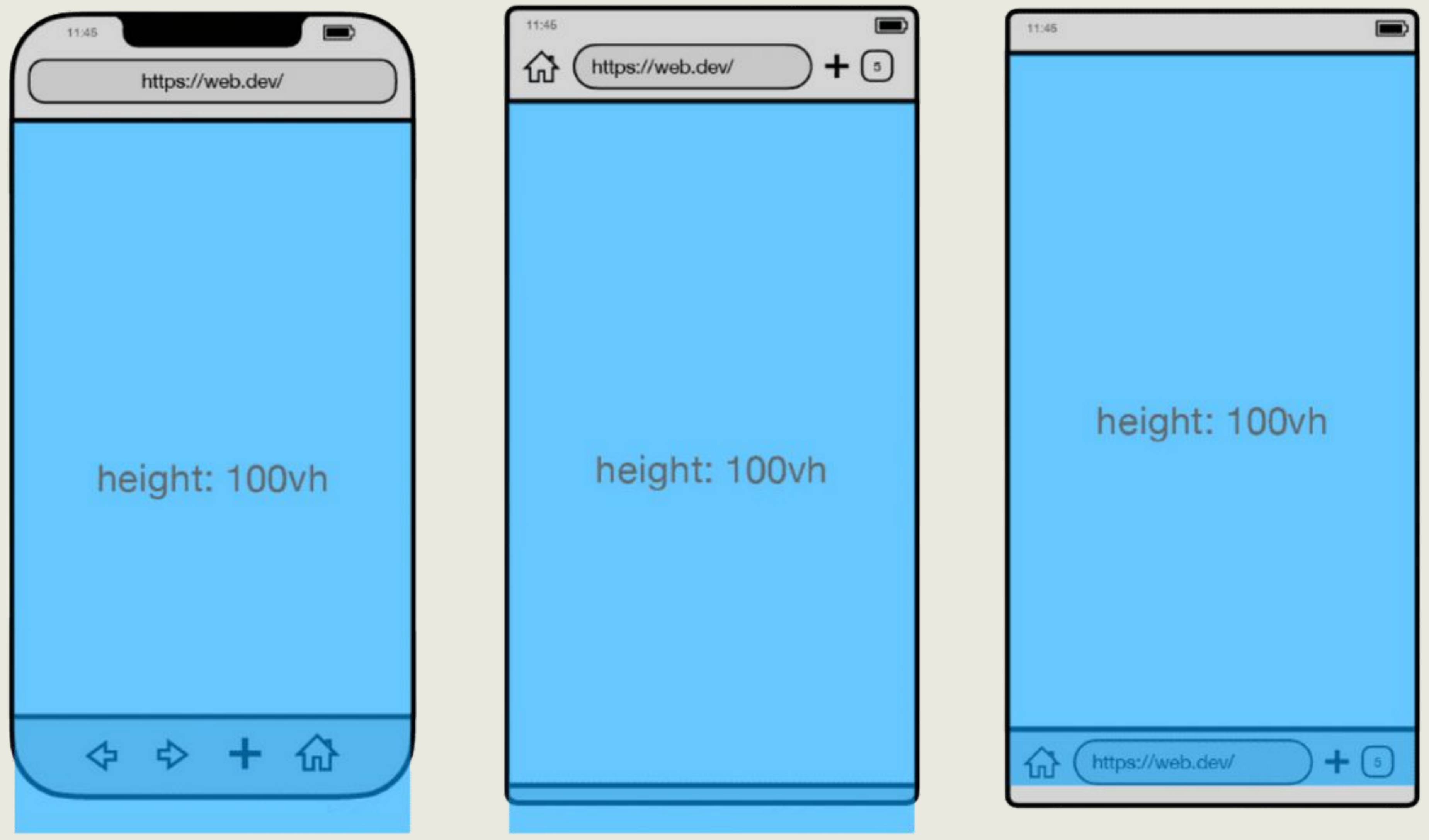


```
.text {
  text-wrap: balance;
}
```

Browser Support 108 108 101 15.4

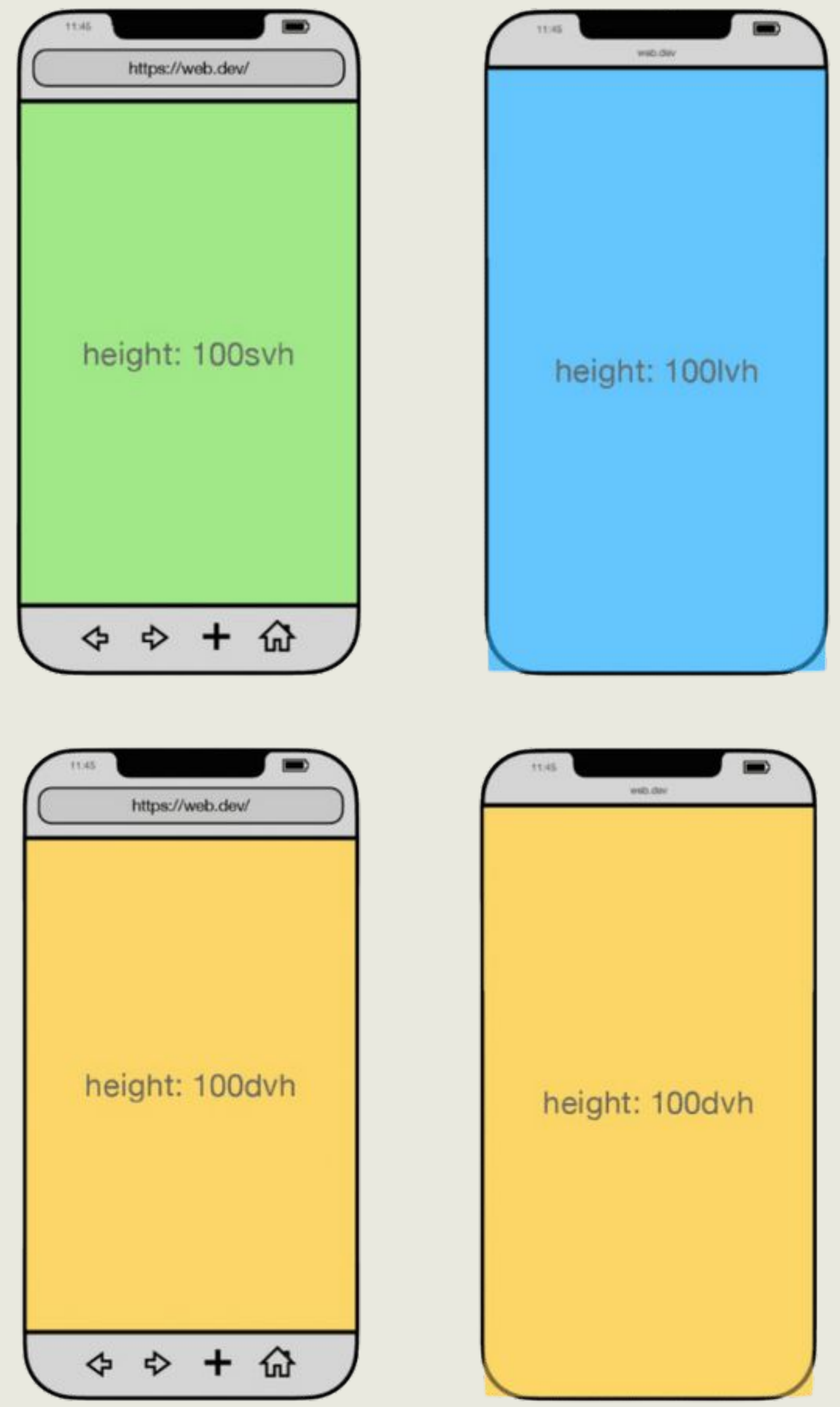
## DYNAMIC VIEWPORT UNITS

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.

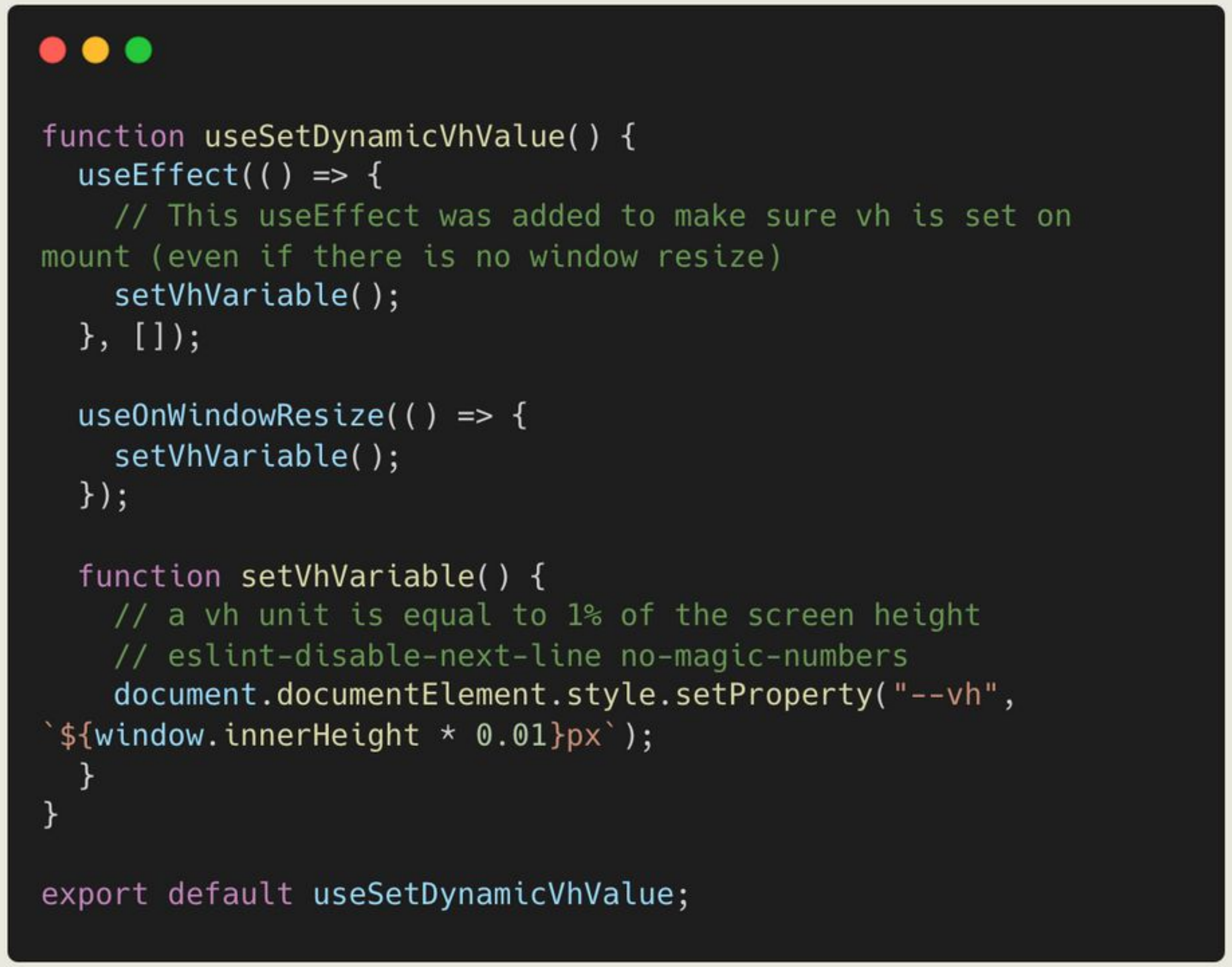


## NEW CSS FEATURE FOR DYNAMIC VIEWPORT UNITS

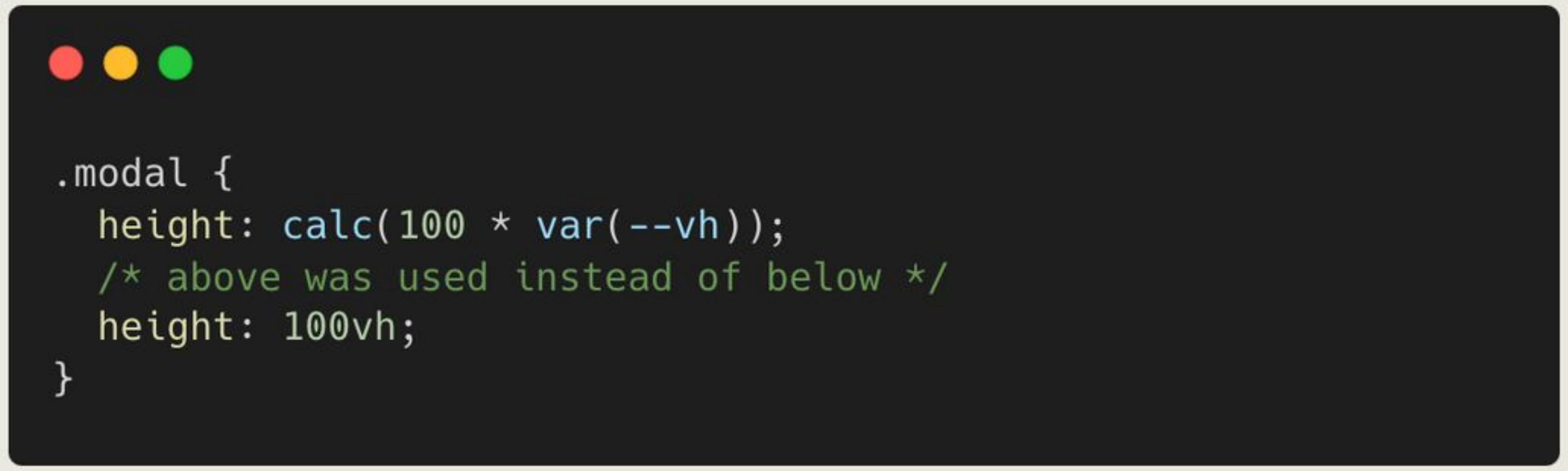
To resolve this issue, we now have new unit values on the web platform, including: - Small viewport height and width (or svh and svw), which represent the smallest active viewport size. - Large viewport height and width (lvh and lvw), which represent the largest size. - Dynamic viewport height and width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



## OLD WAY FOR CALCULATE DYNAMIC VIEWPORT UNITS



```javascript
function useSetDynamicVhValue() {
  useEffect(() => {
    // This useEffect was added to make sure vh is set on
mount (even if there is no window resize)
    setVhVariable();
  }, []);

  useOnWindowResize(() => {
    setVhVariable();
  });

  function setVhVariable() {
    // a vh unit is equal to 1% of the screen height
    // eslint-disable-next-line no-magic-numbers
    document.documentElement.style.setProperty("--vh",
`${window.innerHeight * 0.01}px`);
  }
}

export default useSetDynamicVhValue;
```

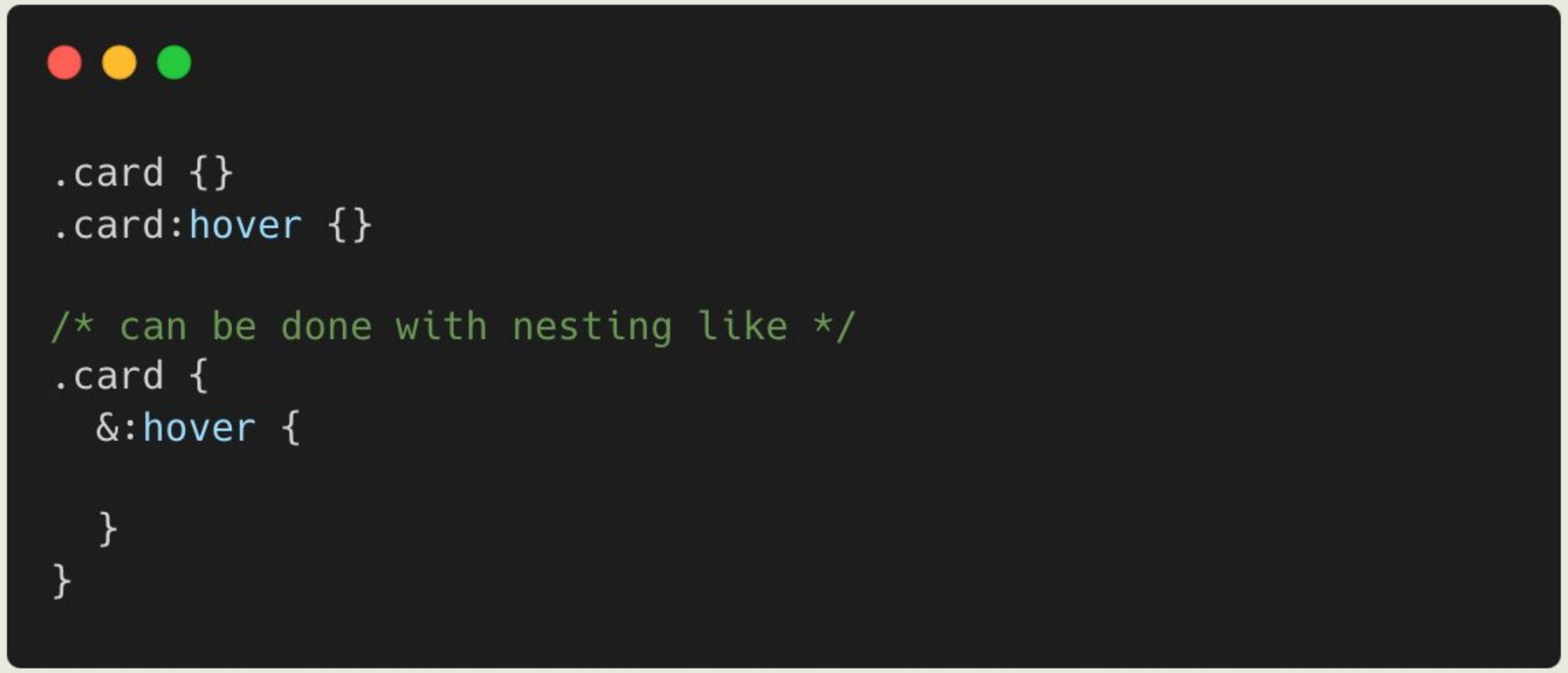


```css
.modal {
  height: calc(100 * var(--vh));
  /* above was used instead of below */
  height: 100vh;
}
```

Browser Support 112 112 117 16.5

## NESTING!

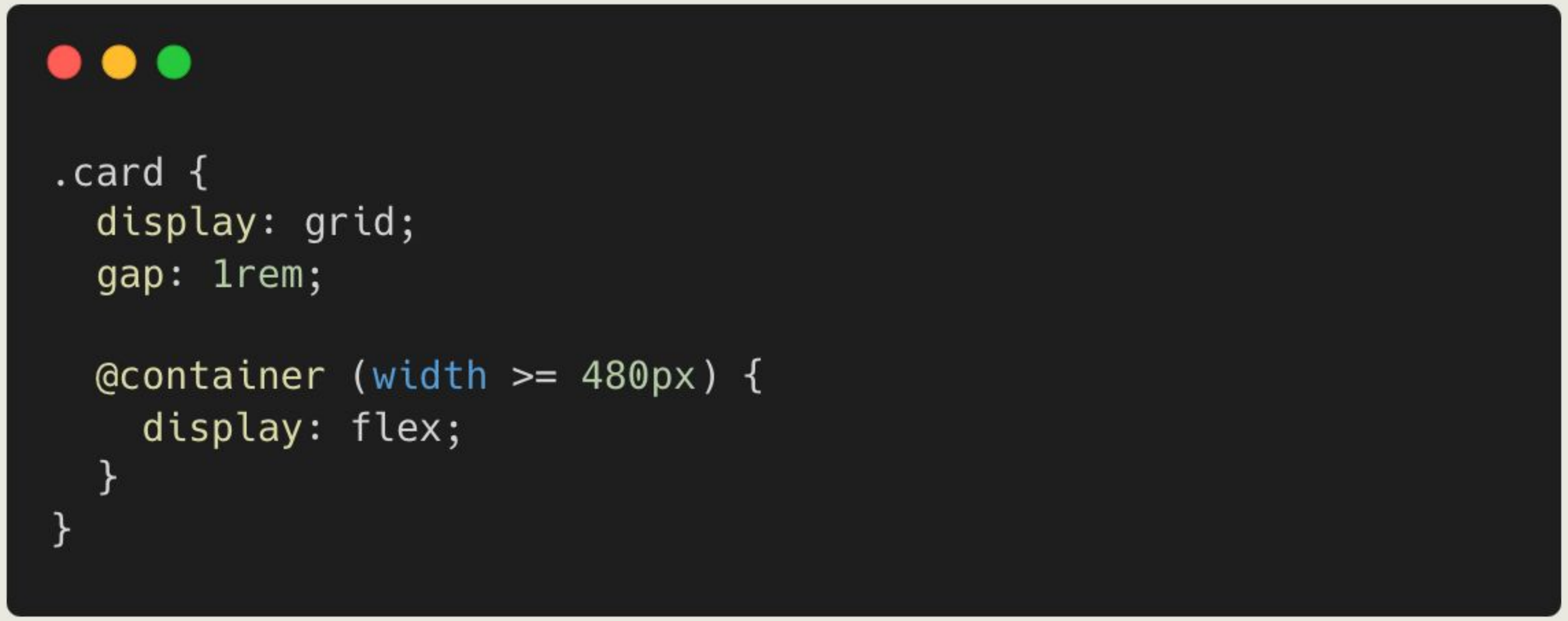
CSS nesting, something folks love from Sass, and one of the top CSS developer requests for years, is finally landing on the web platform. Nesting allows developers to write in a more succinct, grouped format that reduces redundancy.



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

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

  }
}
```

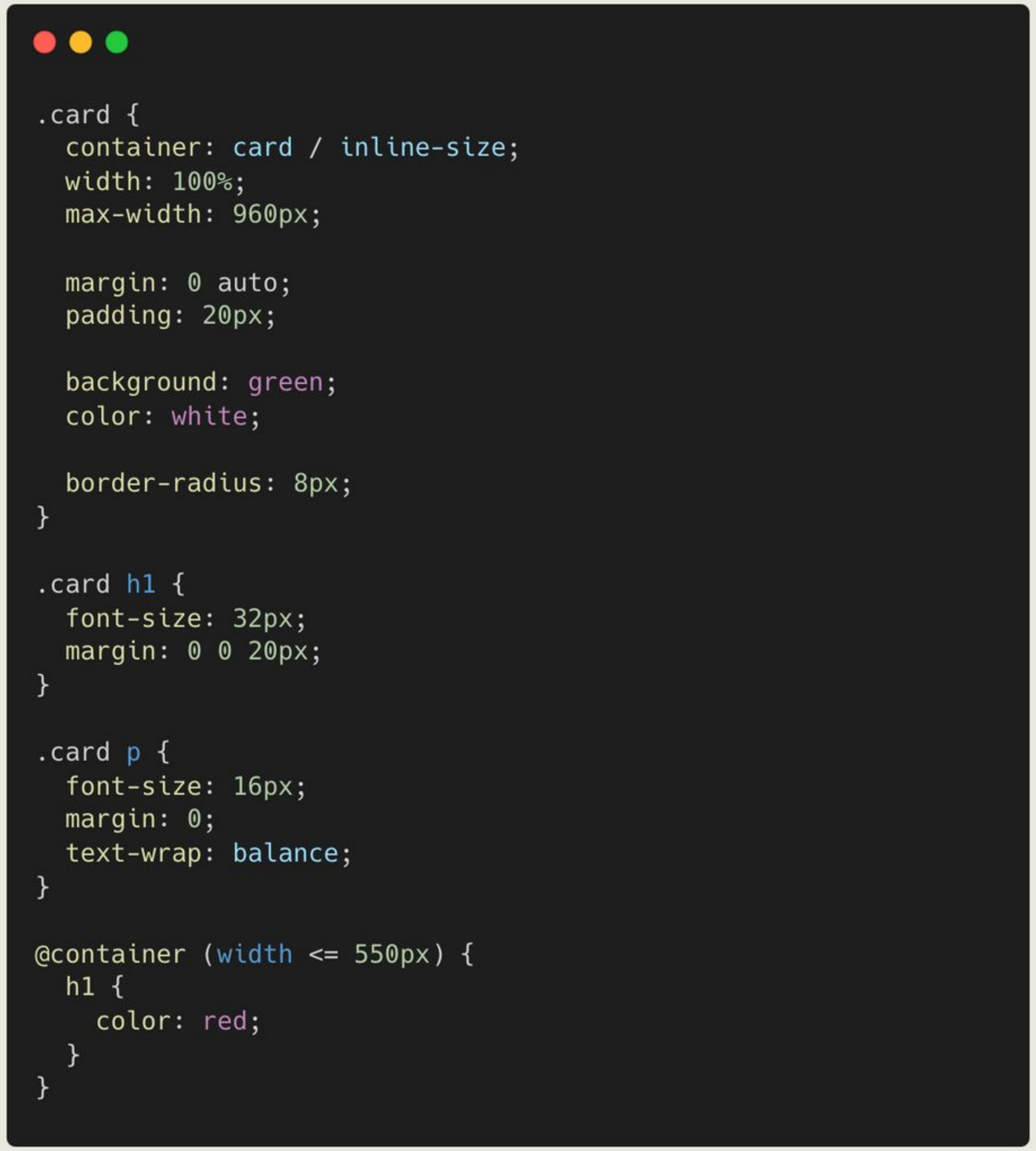
## NESTING!

You can also nest Media Queries, which also means you can nest Container Queries. 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:



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

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

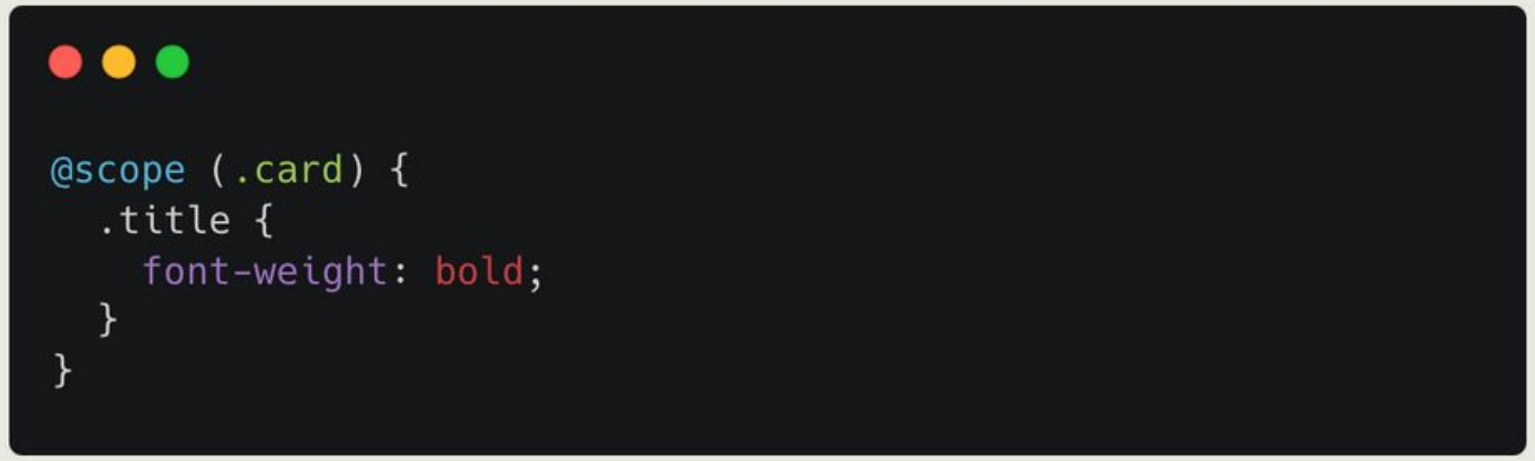
## LET'S MAKE DEMO!



```css
.card {
  container: card / inline-size;
  width: 100%;
  max-width: 960px;

  margin: 0 auto;
  padding: 20px;

  background: green;
  color: white;

  border-radius: 8px;
}

.card h1 {
  font-size: 32px;
  margin: 0 0 20px;
}

.card p {
  font-size: 16px;
  margin: 0;
  text-wrap: balance;
}

@container (width <= 550px) {
  h1 {
    color: red;
  }
}
```

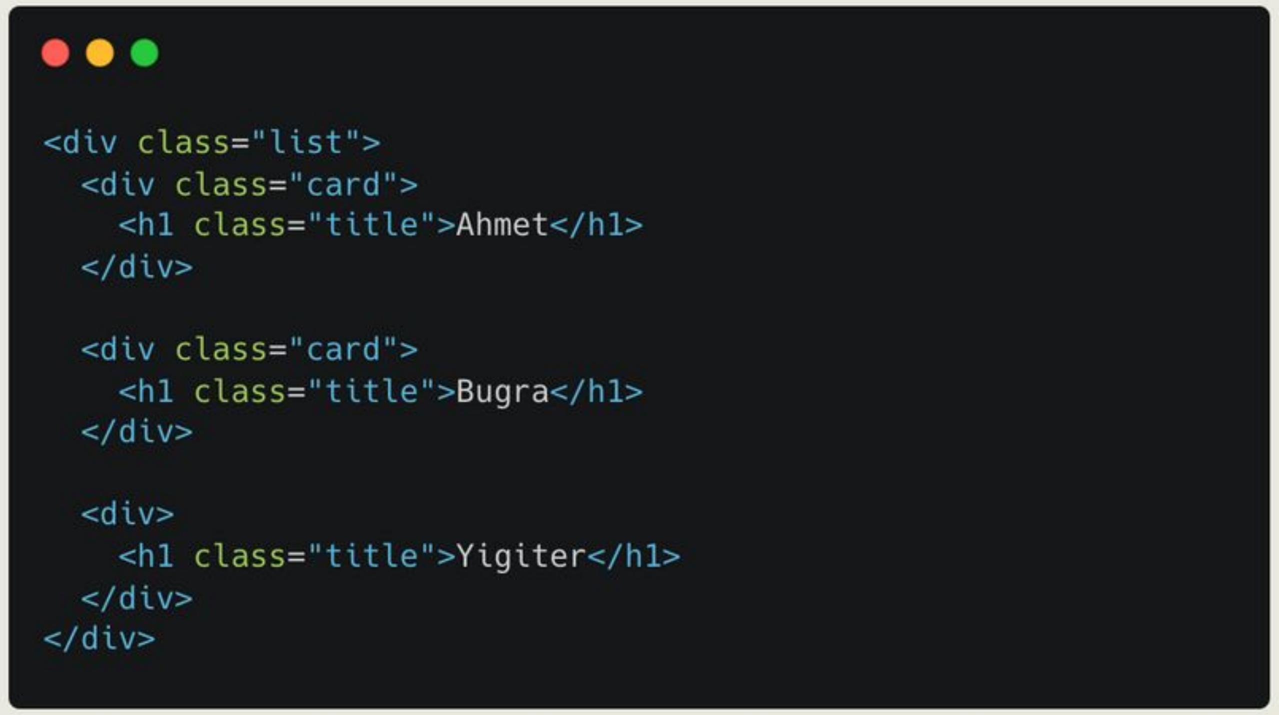
## SCOPED CSS

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.

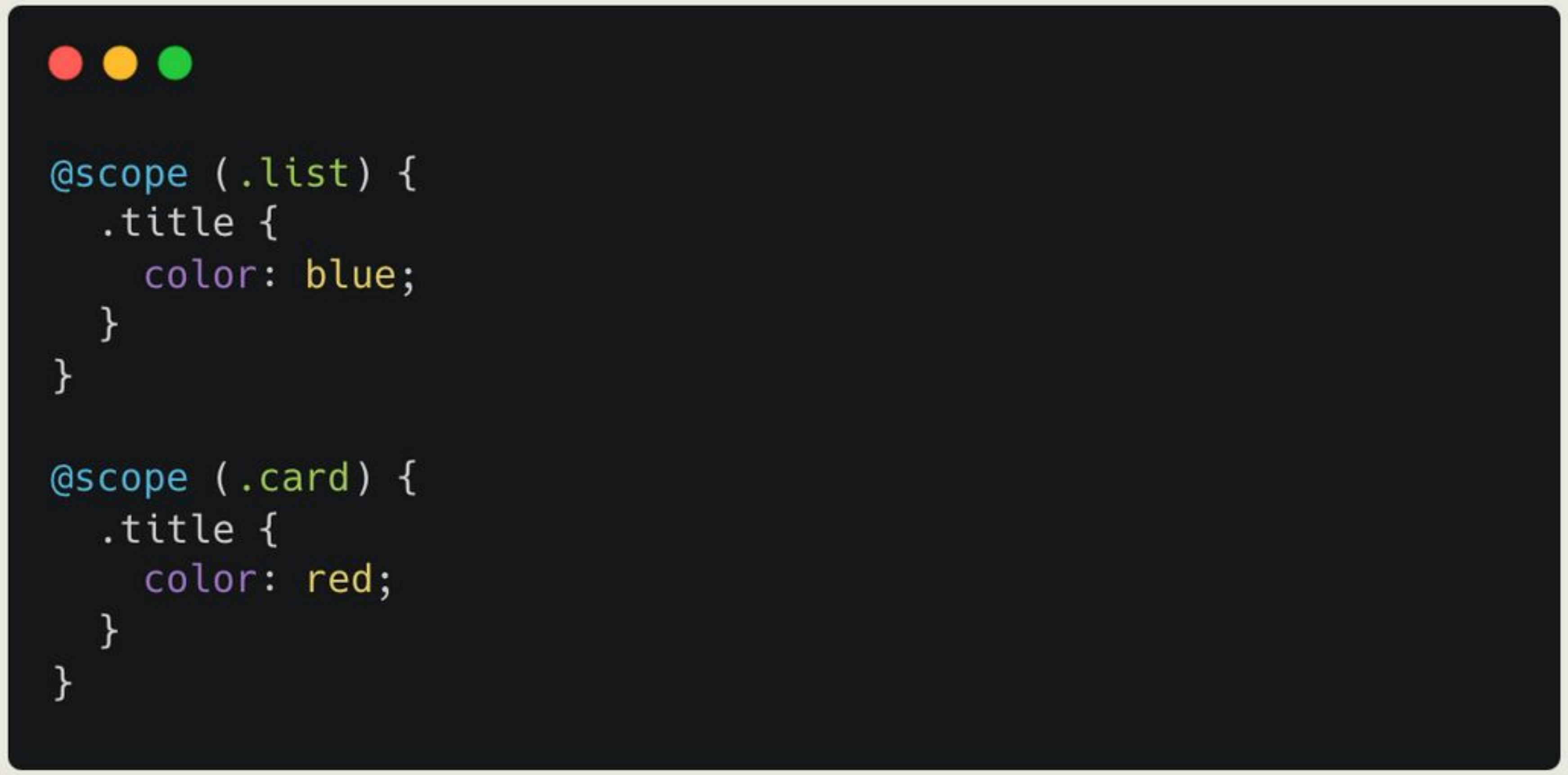


```css
@scope (.card) {
  .title {
    font-weight: bold;
  }
}
```

## LET'S MAKE DEMO!



```html
<div class="list">
  <div class="card">
    <h1 class="title">Ahmet</h1>
  </div>

  <div class="card">
    <h1 class="title">Bugra</h1>
  </div>

  <div>
    <h1 class="title">Yigiter</h1>
  </div>
</div>
```



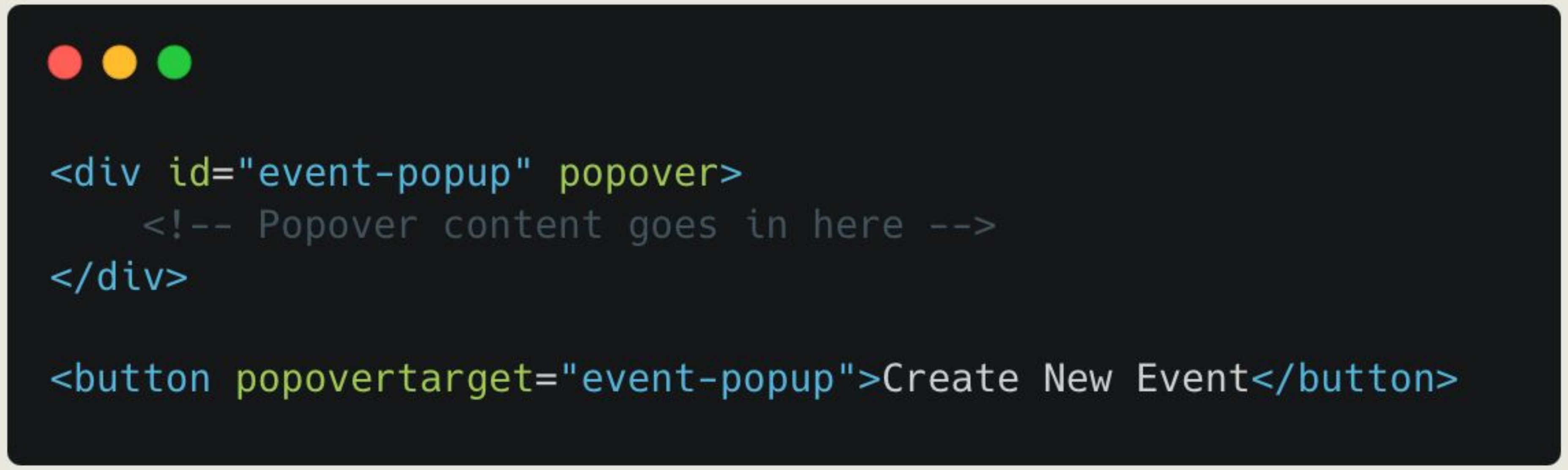
```css
@scope (.list) {
  .title {
    color: blue;
  }
}

@scope (.card) {
  .title {
    color: red;
  }
}
```

## POPOVER

The popover API gives elements some built-in browser-support magic such as:

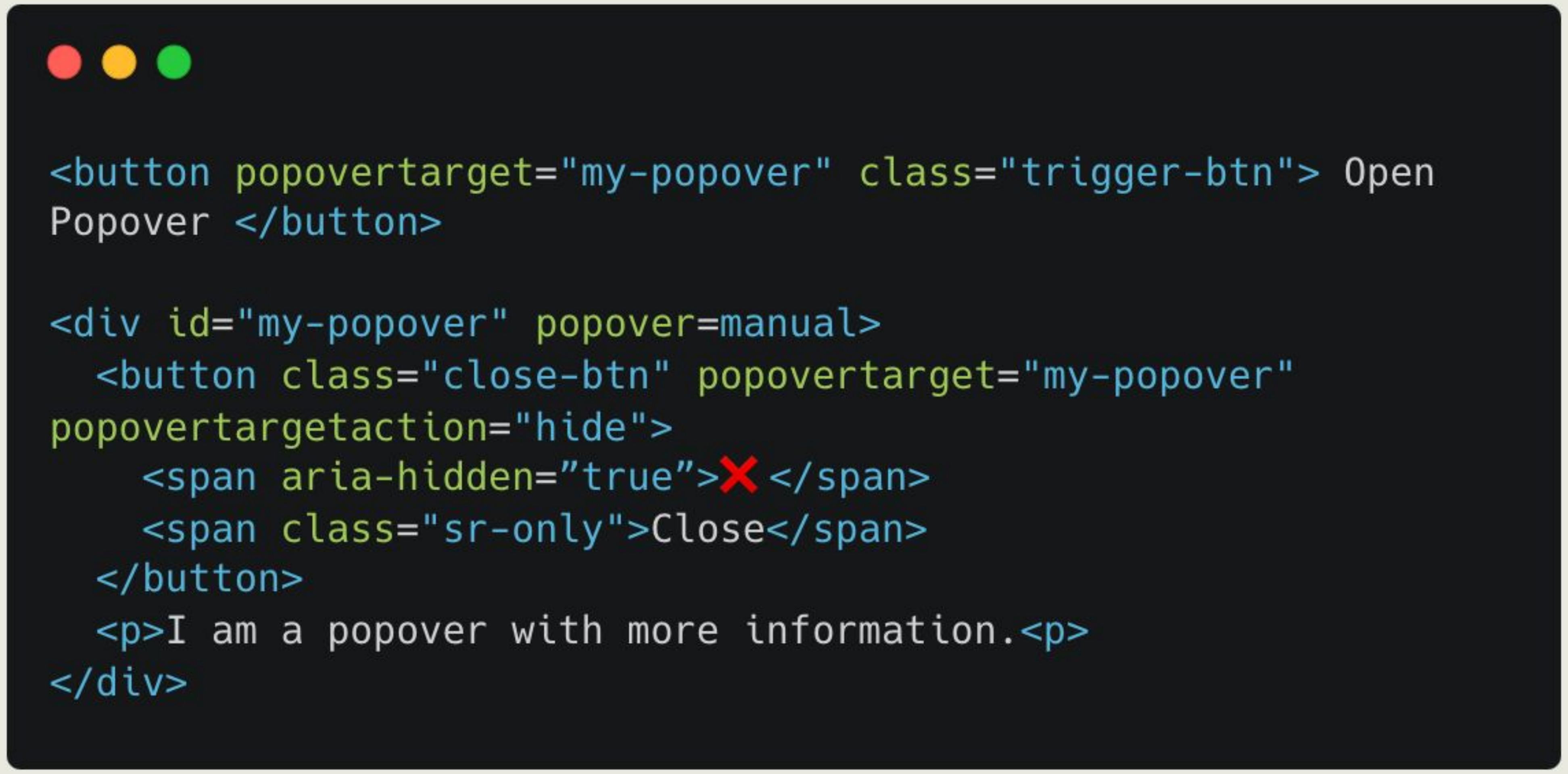
- Support for top-layer, so you don't have to manage z-index. When you open a popover or a dialog, you're promoting that element to a special layer on top of the page.
- Light-dismiss behavior for free in auto popovers, so when you click outside of an element, the popover is dismissed, removed from the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



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

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

# LET'S MAKE DEMO!



```html
<button popovertarget="my-popover" class="trigger-btn"> Open
Popover </button>

<div id="my-popover" popover=manual>
  <button class="close-btn" popovertarget="my-popover"
popovertargetaction="hide">
    <span aria-hidden=”true”>❌ </span>
    <span class="sr-only">Close</span>
  </button>
  <p>I am a popover with more information.<p>
</div>
```

## MORE FEATURES IN 2023

- initial-letter
- Wide-gamut color spaces
- color-mix()
- Cascade layers
- Trigonometric functions
- Individual transform properties
- anchor positioning
- selectmenu
- Discrete property transitions
- Scroll-driven animations
- View transitions

End of Presentation
ANY QUESTIONS?
memecreator.org

You can find my accounts from https://yigiter.dev/