Modern CSS

Christian Burtscher, Jonas Glaser, Marcus Gugacs, and Eva Haring

706.041 Information Architecture and Web Usability 3VU WS 2025/2026
Graz University of Technology

19 Jan 2026

Abstract

This survey explores modern CSS, providing an overview of its evolution and current state. The term “Modern CSS” is sometimes used to encompass the multitude of innovative new CSS features introduced over the past five years or so. These largely emerged as a result of the Interop effort to improve the cross-platform interoperability of the web, which started in 2021. This survey describes 20 of the more important and useful features of Modern CSS.

A significant component of this survey is the accompanying demo web site, NoNewNews, which serves as a practical example of Modern CSS in action, in the context of a news web site. The web site showcases how various Modern CSS techniques are applied to create a visually appealing and functional user experience.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

Over the past decade, the World Wide Web has evolved from a collection of static hypertext documents into a platform for complex, application-like experiences. As user expectations regarding interactivity and responsiveness have risen, so too has the complexity of the underlying technologies. Historically, developers were often forced to rely on heavy JavaScript libraries or brittle CSS workarounds to achieve advanced layouts and state-dependent styling. While effective in the short term, these solutions often introduced performance bottlenecks, accessibility issues, and increased maintenance overhead.

However, the web platform is currently undergoing a significant paradigm shift. Modern CSS has grown beyond a simple styling language into a powerful tool for layout and logic. Features that previously required imperative JavaScript, such as scroll-linked animations, content-aware styling, or complex grid arrangements, are now implemented natively within the browser's rendering engine. This shift enables better performance by offloading computational tasks to the browser, resulting in a smoother user experience.

This survey explains and elaborates on "Modern CSS", describing 20 new and useful features of CSS in detail. The survey is accompanied by a demo web site, which serves as a practical example of modern CSS in action, in the context of a news web site.

1.1 Defining "Modern CSS"

The term "Modern CSS" is sometimes used to encompass the multitude of innovative new CSS features introduced over the past five years or so. These largely emerged as a result of the Interop effort to align the implementation efforts of the major browser vendors, which started in 2021 [WPT 2001; Andrew 2023].

Beyond the reduction of scripts, the term also reflects a change in the standardization process. Unlike the monolithic release of CSS 2.1, CSS since then has been defined by a continuous stream of updates to independent modules. The Interop project defines goals for browser developers in the coming year [WPT 2005], such that particular features of HTML, CSS, JavaScript, and the JavaScript Web APIs become available at around the same time.

1.2 Defining "Baseline"

A particular feature of HTML, CSS, or a Web API becomes "Baseline", once it is supported by a core set of web browsers, namely Chrome (desktop and Android), Firefox (desktop and Android), Safari (macOS and iOS), and Edge (desktop) [web.dev 2025]. This marks the critical threshold where a feature transitions from experimental to production-ready for modern environments. A Baseline feature is said to be *Newly Available* if is has been available for less than 30 months, and *Widely Available* thereafter. Features that do not meet the Baseline criteria are said to have *Limited Availability*.

Technologies classified as *Experimental* represent nascent capabilities that are not yet stable enough for widespread production use. According to MDN [2025j], a feature is considered Experimental, if it is implemented by only one major browser engine or relies on user-configured flags to function, due to the high risk of breaking changes in the underlying specification.

Chapter 2

Features of Modern CSS

Web development is a constantly changing landscape. This makes it hard for professionals and beginners alike to keep up with the latest and greatest tools, features, and frameworks. This chapter provides descriptions of 20 new and particularly interesting features of Modern CSS at the time of writing. The features are grouped into six categories, based on the official Mozilla developer CSS reference [MDN 2025i], namely: Layouts, Properties, Pseudo-Classes, @-Rules, Functions, and Other. Within each category, features are ordered chronologically (based on baseline support), each feature having its own dedicated section.

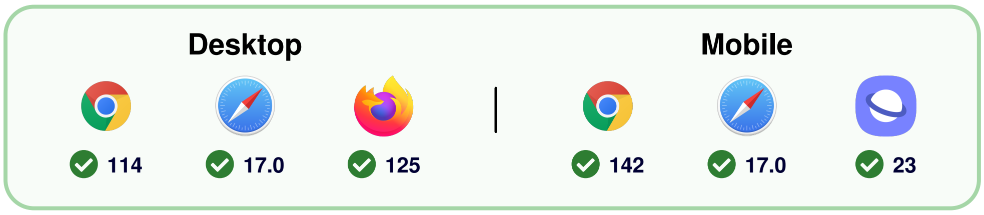
Two infographics are shown for each feature. The first infographic, the browser support infographic, concisely shows support for the feature by the most popular desktop and mobile browsers. Support information is taken from the well-trusted resource Can I use [CIU 2025a], and is used under the terms of a Creative Commons Attribution 4.0 International License. The browser icons for Safari, Firefox, Chrome, and Samsung Internet browsers were sourced from Stockio [2025], Freepik [2025], JJ [2025], and Samsung [2025], respectively. The second infographic, the baseline availability infographic, documents the feature's baseline status. It contains one or more ⓒ buttons to visit the corresponding Can I use page(s), as well as an ⓘ button to visit documentation for the chosen feature.

Each feature is then briefly described and a code snippet is provided to demonstrate its usage. The code examples can be found on GitHub [Burtscher et al. 2025b]. Where it makes sense, an illustation of the result is also included. ⓘ buttons link to further, external examples.

2.1 Layouts

Describes Modern CSS features that involve layouts and UI.

2.1.1 popover-open


Desktop
114
17.0
125
Mobile
142
17.0
23

Baseline - April 2024*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

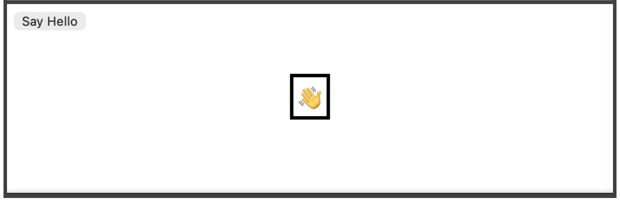
Popovers – controlled by HTML or JavaScript – are a common pattern to draw the user's attention to specific information, by placing content on top of other content [MDN 2025t]. Popovers can either be shown or hidden, most commonly triggered by a button click [Graham 2024]. The Popover API itself is not a CSS feature, but the `:popover-open` CSS pseudo-class can be used to change the appearance of a popover element that is in the showing state. This features allows developers to override the default styles of a popover element when it is displayed to the user [MDN 2025a].

This feature aims to provide an easy and flexible way to style and position popovers in their open state. It allows developers to override and take precedence over the default styles. As shown in Listing 2.1, this example is a slightly modified version of the version shown on CSS-tricks [Graham 2024].

Example Code

```
[popover] {
  position: fixed;
  width: fit-content;
  height: fit-content;
  color: canvastext;
  background-color: canvas;
  inset: 0em;
  margin: auto;
  padding: 0.25em;
  overflow: auto;
}
:popover-open {
  background-color: hsl(25 100%
    50%);
  color: hsl(300 50% 3% / .85);
  padding: 1.5rem;
  width: 25ch;
}
```

Example Result



Listing 2.1: Code example: `:popover-open`.

2.1.2 Masonry Layouts

Desktop | **Mobile**

✗ ⚑ ⚑ | ✗ ✗ ✗

Experimental - Barely available*

This feature is in early development and does not work across most major browsers. There is no information regarding the Baseline release yet.

* This feature only works in Safari's Technology Preview or with appropriately enabled development flags.

Masonry layout introduces a new, more dynamic layout method to make organization of elements in a grid more natural and less strict. Instead of a fixed grid with unwanted gaps, items automatically fill up gaps for an easy-on-the-eye look. The masonry layout is achieved by combining one strict grid axis and one masonry axis. Items are filled according to the masonry algorithm which loads items into the column with the most room, resulting in a tightly packed, space-saving grid [MDN 2025q]

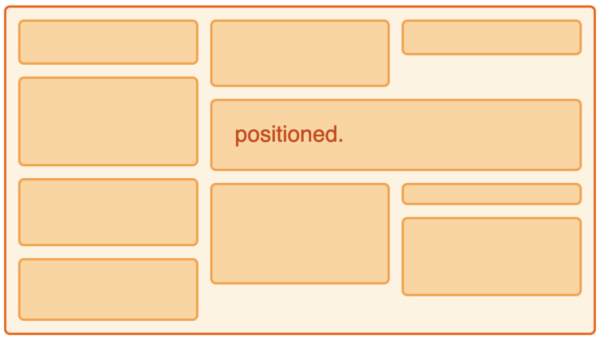
This feature promises to enable Pinterest-style layouts, without the use of JavaScript. Irregular, flexible and flowing grids are often better suited when not all items have the same size and layout [Shoyombo 2025a]. Listing 2.2 shows a row-wise masonry layout from MDN [2025q] that dynamically fills the available width, except where elements are explicitly positioned using `positioned`.

Example Code

```
.grid {
  display: grid;
  gap: 0.65rem;
  grid-template-columns: repeat(
    auto-fill, minmax(7.5rem, 1fr)
    );
  grid-template-rows: masonry;
}

.positioned {
  padding: 1em;
  grid-column: 2 / 4;
}
```

Example Result



Listing 2.2: Code example: Masonry layouts.

2.2 Properties

Describes Modern CSS features that are CSS properties.

2.2.1 text-wrap

Desktop				Mobile		
✓ 114	✓ 17.4	✓ 124		✓ 142	✓ 17.4	✓ 23

Baseline - March 2024*

This feature is well-established and works across many devices and browsers.

* Some parts may have a varying level of support in various browsers.

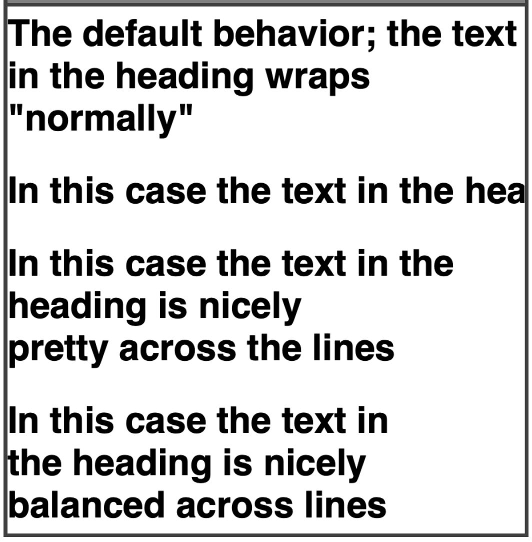
The `text-wrap` property allows developers to improve line breaks and line wraps in text, or turn them off completely. Available options include `wrap`, which causes the text to wrap wherever it would overflow, and `pretty` which does the same as wrap but with another algorithm that values layout over computational speed. Furthermore, `stable` works similarly to `wrap`, but whenever the user is editing the text, the lines before will not be affected by the changes of content. The complete opposite is `nowrap` which causes the text to not wrap at all. Finally, `balance` causes the number of characters to be balanced on each line.

The main purpose of the `text-wrap` shorthand is to simplify the inclusion of typographical content. Developers should make use of it to simplify the required CSS code and improve readability on web sites. Even though the `text-wrap` property has been available since Baseline 2024, individual parts of this feature may not be supported by every version of every major browser. The example in Listing 2.3 is a slightly modified version of the one shown by MDN [2025w].

Example Code

```css
.wrap {
  text-wrap: wrap;
}
.nowrap {
  text-wrap: nowrap;
}
.pretty {
  text-wrap: pretty;
}
.balance {
  text-wrap: balance;
}
h2 {
  font-size: 2rem;
  font-family: sans-serif;
}
```

Example Result



The default behavior; the text in the heading wraps "normally"

In this case the text in the hea

In this case the text in the heading is nicely pretty across the lines

In this case the text in the heading is nicely balanced across lines

Listing 2.3: Code example: `text-wrap`.

2.2.2 field-sizing

Desktop			Mobile		
✓ 123	✗	✗	✓ 142	✗	✓ 27

Experimental - Barely available*

This feature is in early development and does not work across most major browsers. There is no information regarding the Baseline release yet.

* Due to its limited availability, this feature is not yet interoperable and its specification may change.

The `field-sizing` property is a new CSS feature that allows form controls, such as textareas and inputs, to automatically resize based on their content [MDN 2025k]. By setting `field-sizing: content`, an element that traditionally has a fixed default size will instead grow or shrink to fit the text entered by the user. This provides a native, CSS-only way to create form fields that adapt to their value, improving layout consistency and user experience without requiring manual resizing [Nguyen and Warlow 2025].

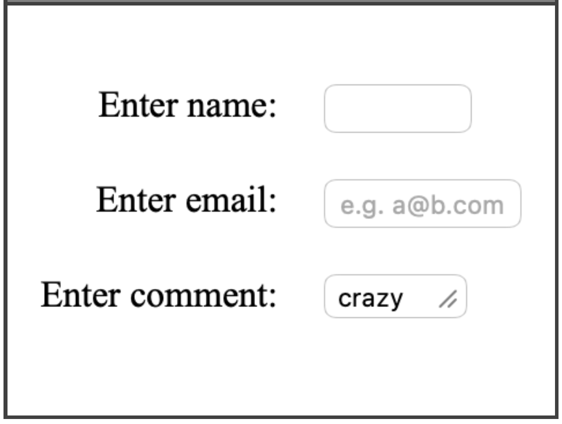
This feature addresses the long-standing UI problem of form fields being either too large or too small for their content. Previously, developers relied on JavaScript to create auto-growing textareas to prevent scrollbars or wasted space [Coyier 2025]. As shown in Listing 2.4, with `field-sizing`, this behavior becomes a simple CSS declaration, eliminating the need for complex scripts. It provides a robust, built-in solution for a very common design pattern, directly improving the usability of web forms.

Example Code

```css
input,
textarea {
  field-sizing: content;
  min-width: 3.25rem;
  max-width: 22rem;
}

label {
  width: 9.50rem;
  margin-right: 1.25rem;
  text-align: right;
}
```

Example Result



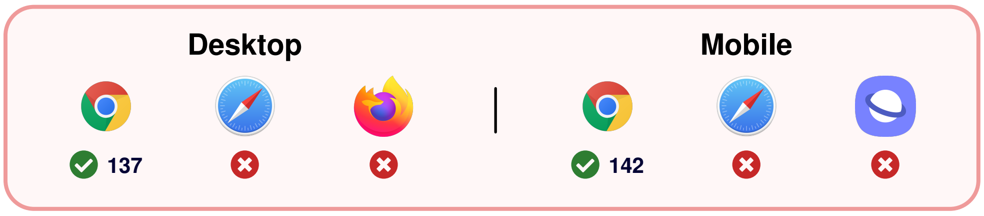
Enter name:

Enter email: e.g. a@b.com

Enter comment: crazy

Listing 2.4: Code example: `field-sizing`.

2.2.3 reading-flow


Desktop
137
Mobile
142

Experimental - Barely available*

This feature is in early development and does not work across most major browsers. There is no information regarding the Baseline release yet.

* Due to its limited availability, this feature is not yet interoperable and its specification may change.

The `reading-flow` property is a new CSS accessibility feature designed to control the logical order of content within a layout [MDN 2025u]. It allows developers to specify how elements in a grid or flex container should be navigated with a keyboard (using the Tab key) and read by assistive technologies.

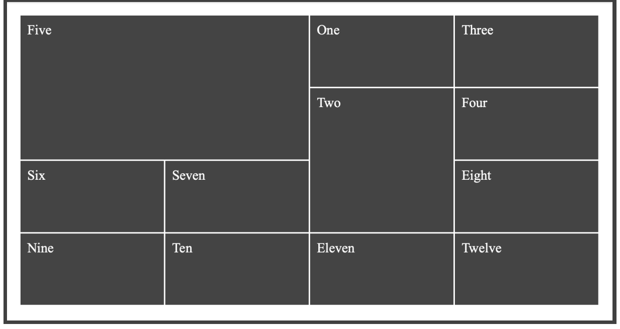
By setting values such as `grid-rows` or `grid-columns`, the navigation sequence can be made to follow the visual arrangement of items, rather than their original order in the HTML source code [Etemad and Atkins 2025].

This feature solves a critical accessibility issue, where a layout's visual order differs from its DOM order, creating a confusing experience for users of keyboards and screen readers. As demonstrated in Listing 2.5, instead of requiring complex JavaScript, `reading-flow` provides a simple, CSS-native solution to ensure a logical and accessible flow for all users [Andrew 2024].

Example Code

```
.wrapper {
  display: grid;
  grid-template-columns: repeat
    (4, 1fr);
  grid-auto-rows: 6.25rem;
}
.wrapper a:nth-child(2) {
  grid-column: 3;
  grid-row: 2 / 4;
}
.wrapper a:nth-child(5) {
  grid-column: 1 / 3;
  grid-row: 1 / 3;
}
.wrapper {
  reading-flow: grid-rows;
}
```

Example Result



Listing 2.5: Code example: `reading-flow`.

2.2.4 interpolate-size

Desktop			Mobile		
✓ 129	✗	✗	✓ 142	✗	✓ 28

Experimental - Barely available*

This feature is in early development and does not work across most major browsers. There is no information regarding the Baseline release yet.

* Due to its limited availability, this feature is not yet interoperable and its specification may change.

Animate to Auto is a new CSS capability that enables developers to smoothly animate elements between fixed size values and intrinsic sizing keywords such as `auto`, `min-content`, `max-content`, or `fit-content` [Coyier 2025]. This is achieved primarily through the `interpolate-size: allow-keywords` property, and the `calc-size()` function for performing calculations on intrinsic sizes [Baron and Atkins 2024]. The feature works universally across any CSS property that accepts size values – not just height, but also width, padding, margin, and other sizing properties—making it a versatile solution for creating dynamic, content-aware animations.

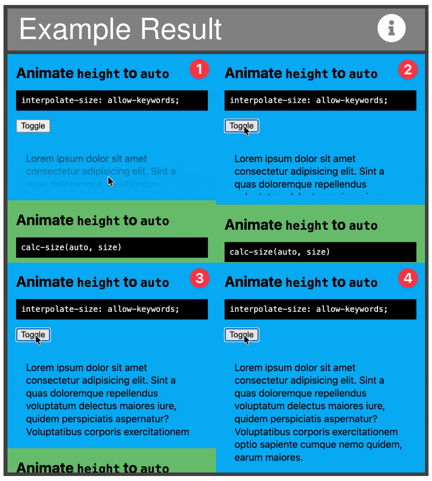
This feature solves one of CSS' most persistent limitations: the inability to animate to or from auto values [Atkins and Etemad 2024]. Previously, expand-and-collapse animations required problematic workarounds like JavaScript solutions that measured element heights off-screen [Coyier 2025]. As illustrated in Listing 2.6, with this freshly introduced feature, developers can now write natural CSS that transitions directly from `height: 0` to `height: auto`, making such animations possible natively without JavaScript.

Example Code

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

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

Listing 2.6: Code example: `interpolate-size`.


Example Result

Animate height to auto 1
interpolate-size: allow-keywords;
Toggle
Lorem ipsum dolor sit amet consectetur adipisicing elit. Sint a quas doloremque repellendus

Animate height to auto 2
interpolate-size: allow-keywords;
Toggle
Lorem ipsum dolor sit amet consectetur adipisicing elit. Sint a quas doloremque repellendus

Animate height to auto
calc-size(auto, size)

Animate height to auto
calc-size(auto, size)

Animate height to auto 3
interpolate-size: allow-keywords;
Toggle
Lorem ipsum dolor sit amet consectetur adipisicing elit. Sint a quas doloremque repellendus voluptatum delectus maiores iure, quidem perspiciatis aspernatur? Voluptatibus corporis exercitationem

Animate height to auto 4
interpolate-size: allow-keywords;
Toggle
Lorem ipsum dolor sit amet consectetur adipisicing elit. Sint a quas doloremque repellendus voluptatum delectus maiores iure, quidem perspiciatis aspernatur? Voluptatibus corporis exercitationem optio sapiente cumque nemo quidem, earum maiores.

Animate height to auto

2.3 Pseudo-Classes

Describes modern CSS features that are pseudo-classes.

2.3.1 has()

Desktop

✓ 105 ✓ 15.4 ✓ 121

Mobile

✓ 142 ✓ 15.4 ✓ 20

Baseline - December 2023*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

The CSS pseudo-class `:has()` is a powerful selector, often referred to as the parent selector. Its core function is to select an element based on the presence of specific descendant elements it contains or subsequent siblings that follow it. This was a long-awaited feature, because it allows developers, for the first time, to apply styling rules to a parent or ancestor element depending on its content. This utility extends beyond direct children to any element within its DOM subtree, providing a new level of contextual styling capability directly within CSS [MDN 2025l].

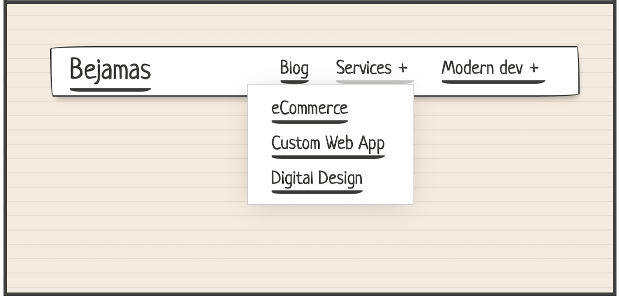
Traditionally, CSS selectors could only target elements downwards (children) or sideways (siblings) in the DOM tree. The primary motivation for :has() was to break this directional limitation, allowing selections to be made upwards. This opens up dynamic styling possibilities that previously required JavaScript. Before `:has()`, developers had to write scripts to check for child elements and then add classes to the parent. As shown in Listing 2.7, the purpose of `:has()` is to provide a declarative and performant native CSS solution, leading to cleaner code and enabling the creation of more robust, self-adapting components.

Example Code

```css
button:has(.icon) {
  display: flex;
  gap: 0.5rem;
}

nav li:has(ul) > a::after {
  content: "+";
  margin-inline: 0.5rem;
}
```

Example Result



Listing 2.7: Code example: `:has()`.

2.4 @-Rules

Describes Modern CSS features that are @-rules.

2.4.1 @supports

Desktop				Mobile		
✓ 4	✓ 4	✓ 49		✓ 142	✓ 3.2	✓ 4

Baseline - September 2015*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

Using the `@supports` CSS at-rule enables developers to apply CSS declarations based on whether a browser supports a specific feature. To define more precise support rules, multiple conditions can be combined by conjunctions (and), disjunctions (or), and even negations. These support conditions can follow either a property-value syntax or a function syntax [MDN 2025d].

This feature has been available for a long time. It is widely used [Devographics 2025], and can be very helpful when experimenting with new features that are not yet supported in all browsers. As demonstrated in Listing 2.8, it allows developers to use these new features without causing errors.

Example Code

```
.flex-container > * {
  padding: 0.3em;
  list-style-type: none;
  text-shadow: 0 0 0.125rem red;
  float: left;
}

@supports (display: flex) {
  .flex-container > * {
  text-shadow: 0 0 0.125rem blue;
  float: none;
  }

  .flex-container {
  display: flex;
  }
}
```

Listing 2.8: Code example: `@supports`.

2.4.2 @property

Desktop			Mobile		
✓ 85	✓ 16.4	✓ 128	✓ 142	✓ 16.4	✓ 14.0

Baseline - July 2024*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

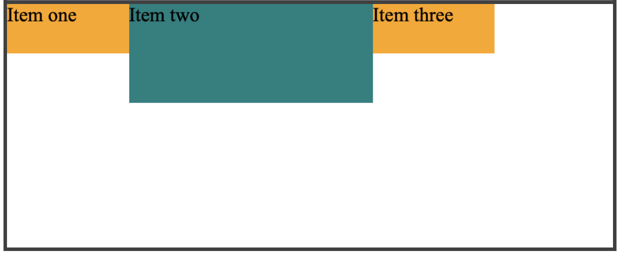
With the introduction of the `@property` at-rule, CSS now allows the definition of custom properties without the need for JavaScript. This results in CSS-native support for typed variables, default values, and inheritance. Given valid code as a prerequisite, the `@property` rule will register the custom property [MDN 2025c]. The example shown in Listing 2.9 demonstrates the usage of initial values, parent object overwrites, manual usage of initial values, and handling of invalid types. The example is a slightly modified version of the example shown by MDN [2025c].

CSS variables are an important feature, for which developers have had to use custom CSS preprocessors for. Now, custom properties in modern CSS are even more powerful than preprocessor variables [Albert 2024]. Like many of the modern CSS features, the `@property` at-rule is powered by the Houdini APIs, granting developers low-level access to the CSS engine [MDN 2025m].

Example Code

```css
@property --item-size {
  syntax: "<percentage>";
  inherits: true;
  initial-value: 40%;
}
.container {
  /* ... parent property value */
  --item-size: 20%;
}
.two {
  --item-size: initial;
  background-color: teal;
}
.three {
  /* invalid value */
  --item-size: 62rem;
}
```

Example Result



Listing 2.9: Code example: `@property`.

2.4.3 Container Size Queries

Desktop			Mobile		
✓ 106	✓ 16.0	✓ 110	✓ 142	✓ 16.0	✓ 20

Baseline - October 2025*

This feature is well-established and works across many devices and browser versions.

* This feature is considered safe to use in production environments.

Container size queries are a central feature of modern responsive design that allow styles to be applied to an element based on the size of its parent container, rather than relying on the size of the entire browser window. To utilize this, an element must be explicitly declared as a size query container, typically with the CSS property `container-type: inline-size`. Subsequently, the `@container` rule can be used to define conditional styles that activate when the container meets specific size characteristics, such as `min-width`, `max-height`, or `aspect-ratio`. This enables components to dynamically adapt their layout and appearance to the space available to them [MDN 2025f].

The primary motivation for container size queries was to create a truly component-based approach to web design. Traditional media queries are global and lack context, which complicates the reusability of components. The purpose of container queries is to enable components to define their own responsive rules, allowing them to display correctly in any layout, whether a wide main column or a narrow sidebar. As shown in Listing 2.10, components can adapt their styles based on their container's dimensions rather than the viewport size.

Example Code

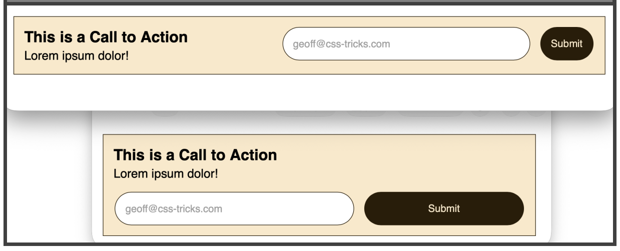
```
@container cta (width > 50rem) {
  .cta-layout {
  justify-content: space-between;
  }
}

@container cta (width < 50rem) {
  .cta-layout {
  flex-direction: column;
  }

  .cta-layout input,
  .cta-layout button {
  flex: 1 1 auto;
  }
}
```

Listing 2.10: Code example: Container size queries.

Example Result



2.4.4 Container Style Queries

Desktop			**Mobile**		
! 111	! 18.0	✗	! 142	! 18.0	! 22

Experimental - Partially available*

This feature is not yet fully considered baseline as it lacks universal browser support. Even supported browsers only partially allow the usage of this feature.

*As the specification is still in development, syntax and behavior are subject to change.

Container style queries expand upon the concept of container queries, moving beyond simple size-based adjustments. While traditional container queries adapt the layout based on a container's width or height, style queries allow styles to be applied based on the computed CSS values of the container itself. So, instead of asking, "How wide is the container?", style queries ask, "What style does the container have?". Currently, support is primarily limited to querying CSS custom properties. This makes it possible to change the appearance of child elements depending on the value set for a specific custom property on the container. As demonstrated in Listing 2.11, a component can automatically adjust its styling based on custom property values.

The motivation behind container style queries is to create more context-sensitive components. The main purpose is to couple the styling of child elements to the thematic or state-based style characteristics of the parent container. For example, a component could automatically adjust its color scheme depending on whether its container has a `--theme: dark` or `--theme: light` custom property set. In the future, this will be extended to any CSS property, allowing reactions to characteristics like `background-color` or `display`. This reduces the need for additional CSS classes or JavaScript to manage states and leads to more modular and maintainable designs [MDN 2025f].

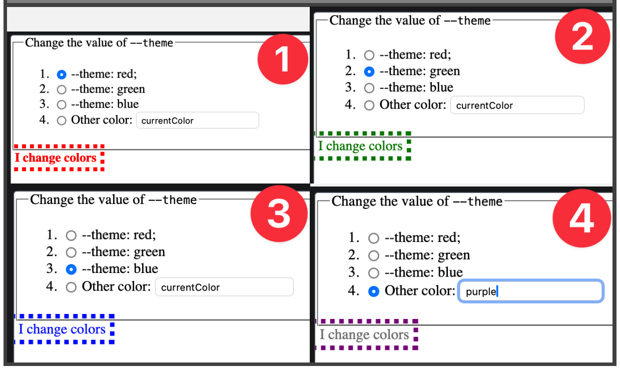
Example Code

```
@container style(--theme: green)
    or style(--theme: blue) or
    style(--theme: red) {
  output {
  color: var(--theme);
  }
}

@container style(--theme: red) {
  output {
  font-weight: bold;
  }
}
```

Listing 2.11: Code example: Container style queries.

Example Result



Change the value of --theme
1. ● --theme: red;
2. ○ --theme: green
3. ○ --theme: blue
4. ○ Other color: currentColor

I change colors

Change the value of --theme
1. ○ --theme: red;
2. ● --theme: green
3. ○ --theme: blue
4. ○ Other color: currentColor

I change colors

Change the value of --theme
1. ○ --theme: red;
2. ○ --theme: green
3. ● --theme: blue
4. ○ Other color: currentColor

I change colors

Change the value of --theme
1. ○ --theme: red;
2. ○ --theme: green
3. ○ --theme: blue
4. ● Other color: purple

I change colors

2.4.5 @function

Desktop				Mobile		
✓ 139	✗	✗	|	✓ 142	✗	✗

Experimental - Barely available*

This feature is in early development and does not work across most major browsers. There is no information regarding the Baseline release yet.

* Due to its limited availability, this feature is not yet interoperable and its specification may change.

The `@function` at-rule introduces the ability to define custom, reusable functions directly within CSS [MDN 2025b]. These functions can accept arguments, perform calculations or logic, and return a single value to be used in any CSS property. Once defined, a custom function is called using a dashed-ident syntax, such as `--my-function(arg1, arg2)`. This allows developers to encapsulate complex or repetitive logic into a clean, callable unit, making stylesheets more modular and readable. As illustrated in Listing 2.12, custom functions can perform calculations and return values for use in any CSS property [Suzanne and Atkins 2025].

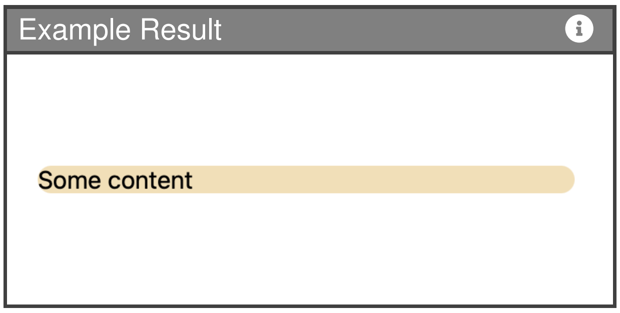
This feature addresses a major gap in native CSS: the lack of dynamic, reusable logic. While CSS custom properties allow for storable values, they are static. For years, developers have relied on preprocessors to create functions for tasks like color manipulation or complex layout calculations [Suzanne 2023]. The `@function` rule brings this essential capability directly into the browser. It eliminates the dependency on external build tools for dynamic styling, reduces code duplication, and empowers developers to write more powerful and maintainable CSS natively.

Example Code

```css
@function --double(--value) {
  result: calc(var(--value) * 2);
}

p {
  --base-spacing: 0.625rem;
  border-radius: var(
    --base-spacing);
  padding: --double(var(
    --base-spacing));
  width: 50%;
  background-color: wheat;
}
```

Example Result


Some content

Listing 2.12: Code example: `@function`.

2.5 Functions

Describes Modern CSS features that are CSS functions.

2.5.1 linear()

Desktop			Mobile		
✓ 113	✓ 17.2	✓ 112	✓ 142	✓ 17.2	✓ 23

Baseline - December 2023*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

The CSS function `linear()` is an easing function used to control the timing of CSS animations and transitions. Its primary purpose is to create a transition curve that moves at a constant speed between multiple defined points. Unlike a simple linear animation, which only connects a start and an end point at a uniform speed, `linear()` allows for the definition of multiple intermediate points. This enables developers to approximate complex animation curves through a sequence of straight line segments, thereby gaining more precise control over the animation's progression [MDN 2025p].

The motivation behind the development of `linear()` was to provide developers with a tool to approximate complex animation curves directly in CSS, without having to resort to JavaScript. Many natural movements (like a bouncing ball or a spring effect) do not follow simple mathematical curves such as ease-in or ease-out. With `linear()`, developers can replicate such complex curves through a sequence of many small, straight line segments. The more points defined, the more accurately the desired curve can be approximated. As shown in Listing 2.13, different point sequences create distinctly different animation behaviors.

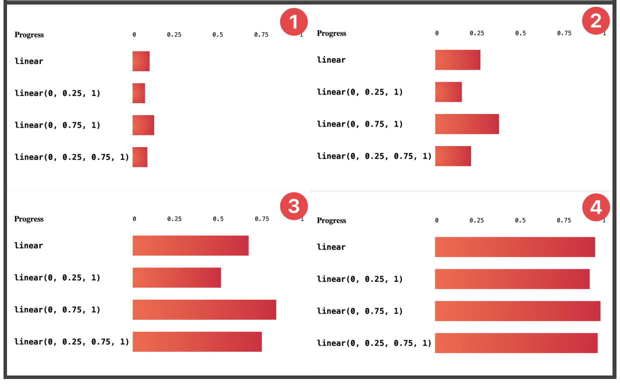
Example Code

```
.bar-1 {
  animation-timing-function:
    linear(0, 0.25, 1);
}

.bar-2 {
  animation-timing-function:
    linear(0, 0.75, 1);
}

.bar-3 {
  animation-timing-function:
    linear(0, 0.25, 0.75, 1);
}
```

Example Result



Listing 2.13: Code example: `linear()`.

2.5.2 light-dark()

Desktop				Mobile		
✓ 123	✓ 17.5	✓ 120	|	✓ 142	✓ 17.5	✓ 27

Baseline - May 2024*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

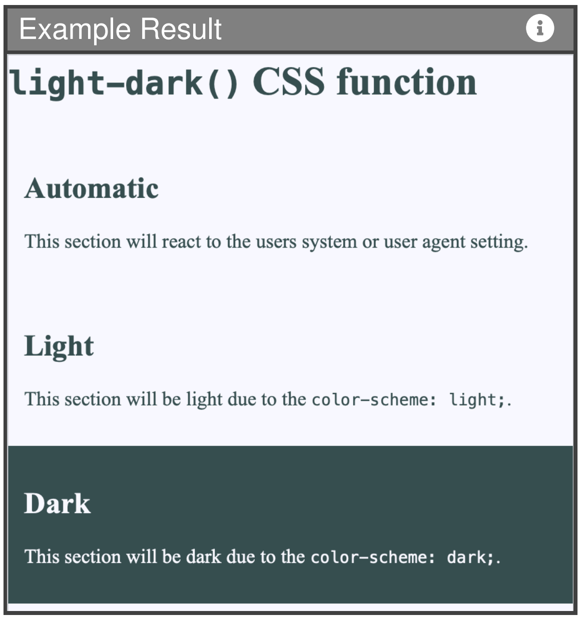
Many popular web sites provide the user with a choice to change between light and dark modes. Some users prefer the dark text on a light background, while others prefer light text on a dark background. The `light-dark()` function allows properties to be assigned two different colors [MDN 2025o]. Either light or dark preference can then be manually enforced or decided based on either the operating system settings or via the user agent settings. Listing 2.14 shows a slightly modified version of the approach by MDN MDN [2025o].

The purpose of this function is to provide more choice to both developers and visitors of web sites. Additionally, it reduces the effort for the developers to implement two themes by not having to employ media queries.

Example Code

```
:root {
  color-scheme: light dark;
  --light-bg: ghostwhite;
  --light-color: darkslategray;
  --dark-bg: darkslategray;
  --dark-color: ghostwhite;
}
* {
  background-color: light-dark(
    var(--light-bg), var(--dark-bg
    ));
  color: light-dark(var(
    --light-color), var(
    --dark-color));
}
.light { color-scheme: light; }
.dark { color-scheme: dark; }
```

Example Result



`light-dark()` CSS function

Automatic

This section will react to the users system or user agent setting.

Light

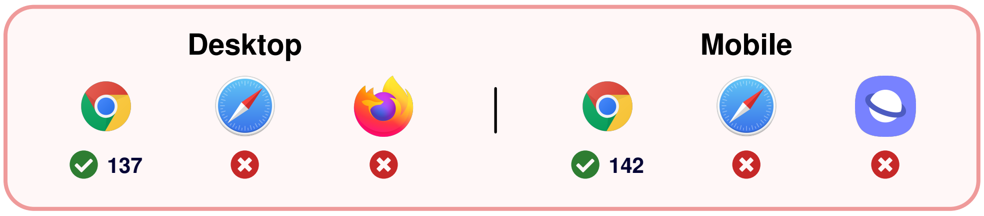
This section will be light due to the `color-scheme: light;`.

Dark

This section will be dark due to the `color-scheme: dark;`.

Listing 2.14: Code example: `light-dark()`.

2.5.3 if()



Desktop			Mobile		
✓ 137	✗	✗	✓ 142	✗	✗

Experimental - Barely available*

This feature is in early development and does not work across most major browsers. There is no information regarding the Baseline release yet.

* Due to its limited availability, this feature is not yet interoperable and its specification may change.

With the introduction of the CSS `if()` function, it is no longer necessary to set each property value manually, but they can be set depending on a conditional test inside of any property. Similar to if-else-statements, the `if()` function allows to specify zero or more semi-colon-separated condition-value pairs. In the case of no true condition, an else case should be defined or a fallback value must be set to avoid problems with non-supporting browsers. There are three types of if-test: style, media, and feature queries. It is also possible to nest `if()` functions for more advanced usage [MDN 2025n].

This feature should be used if property-conditional behavior is needed and adaptable, self-contained properties are the goal. The `if()` function aims to complement existing media queries, container style queries, and custom property toggles as each of them still provides optimal usage for some cases. The possibility to nest several if-conditions increases the modularity and complexity of properties. Listing 2.15 demonstrates nested conditional logic for theme-based color selection [Akinyemi 2025].

Example Code

```
color: if(
  style(--scheme: ice):
  if(
  media(prefers-color-scheme:
    dark): #caf0f8;
  else: #03045e;
  );
  style(--scheme: fire):
  if(
  media(prefers-color-scheme:
    dark): #ffc971;
  else: #621708;
  );
  else: black
);
```

Listing 2.15: Code example: `if()`.

2.5.4 clamp()

Desktop				Mobile		
✓ 79	✓ 13.1	✓ 75		✓ 142	✓ 13.4	✓ 12

Baseline - June 2020*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

The CSS `clamp()` function is a powerful tool for fluid design, enabling a CSS value to be kept within a defined range. The function accepts three arguments: a minimum value, a preferred value, and a maximum value. The computed value will always be the preferred value as long as it falls between the minimum and maximum values. If the preferred value drops below the minimum, the minimum value is used; if it rises above the maximum, the maximum value is used. This is particularly useful for properties such as `font-size` or `width`, to ensure they adapt to the viewport, but never become too small or too large [MDN 2025e].

The primary motivation behind `clamp()` was to simplify responsive and fluid design, particularly concerning typography. Previously, developers had to write complex media queries to prevent viewport-relative units (such as vw) from resulting in extreme values on very small or very large screens. The purpose of `clamp()` is to encapsulate this logic within a single, declarative line. Instead of defining multiple breakpoints, `clamp()` allows for setting a flexible value that scales smoothly, but is clamped at the ends of the range. As shown in Listing 2.16, this results in cleaner, more maintainable code and a smoother user experience across a variety of device sizes..

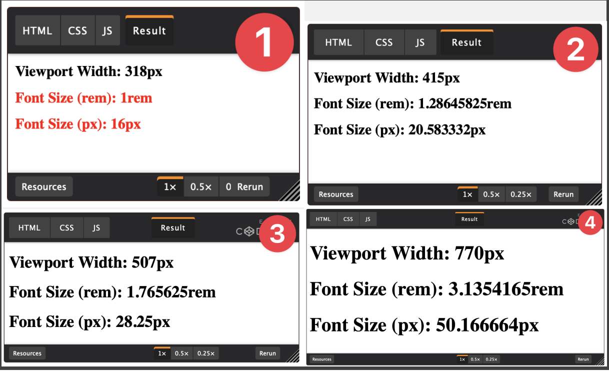
Example Code

```
h1{
  font-size: clamp(1rem, -0.875
    rem + 8.333333vw, 3.5rem);
}

.red{
  color:red;
}
```

Listing 2.16: Code example: `clamp()`.

Example Result



Viewport Width: 318px
Font Size (rem): 1rem
Font Size (px): 16px

Viewport Width: 415px
Font Size (rem): 1.28645825rem
Font Size (px): 20.583332px

Viewport Width: 507px
Font Size (rem): 1.765625rem
Font Size (px): 28.25px

Viewport Width: 770px
Font Size (rem): 3.1354165rem
Font Size (px): 50.166664px

2.5.5 min()

Desktop			Mobile		
✓ 79	✓ 13.1	✓ 75	✓ 142	✓ 13.4	✓ 12

Baseline - June 2020*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

The CSS function `min()` compares a list of comma-separated values and applies the smallest (the minimum) value as the final CSS value. It accepts various units and can even contain mathematical expressions. A classic use case is defining element widths: with `width: min(100%, 60rem)`, the browser is instructed to make the element 60 rem wide, unless the available space (100%) is less than 60 rem. In that case, the percentage value wins. This effectively means: the element is at most 60 rem wide, but automatically adapts on smaller screens [MDN 2025s]

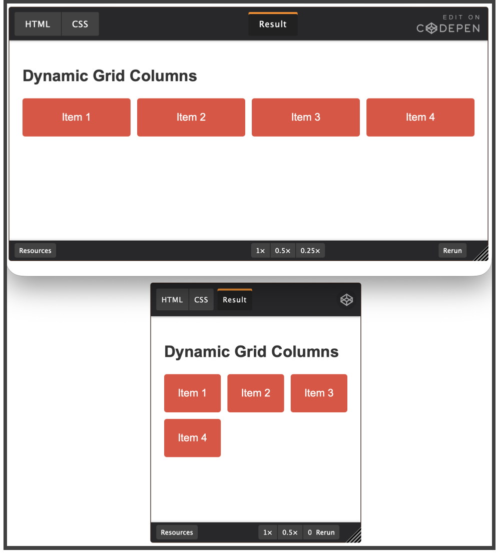
The main purpose of `min()` is to concisely set upper limits (caps) for responsive values. The motivation behind it was to reduce the dependency on media queries and to simplify the often cumbersome combination of properties. Traditionally, one often had to define `width: 100%` and `max-width: 60rem` separately to achieve the same effect. `min()` combines this logic into a single expression that can be used anywhere – not just for widths, but also for margins, font sizes, or padding. As demonstrated in Listing 2.17, this enables a fluid design that automatically constrains itself without the need to define complex breakpoints.

Example Code

```
.grid {
  display: grid;
  grid-template-columns: repeat(
    auto-fit, minmax(min(15rem,
    25%), 1fr));
  gap: 1rem;
}
```

Listing 2.17: Code example: `min()`.

Example Result



2.5.6 max()

Desktop

✓ 79 ✓ 13.1 ✓ 75

Mobile

✓ 142 ✓ 13.4 ✓ 12

Baseline - June 2020*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

The CSS `max()` function compares a list of comma-separated values and applies the largest (the maximum) value as the final CSS value. It accepts various units and mathematical expressions. A common, and at first glance counterintuitive, use case is to ensure a minimum size. With a rule like `width: max(50vw, 30rem)`, the browser is instructed to choose the larger of the two values. If 50% of the viewport width is smaller than 30 rem (for example, on a smartphone), the fixed value wins. This effectively means that the element is at least 30 rem wide, but it grows as soon as the viewport is large enough [MDN 2025r].

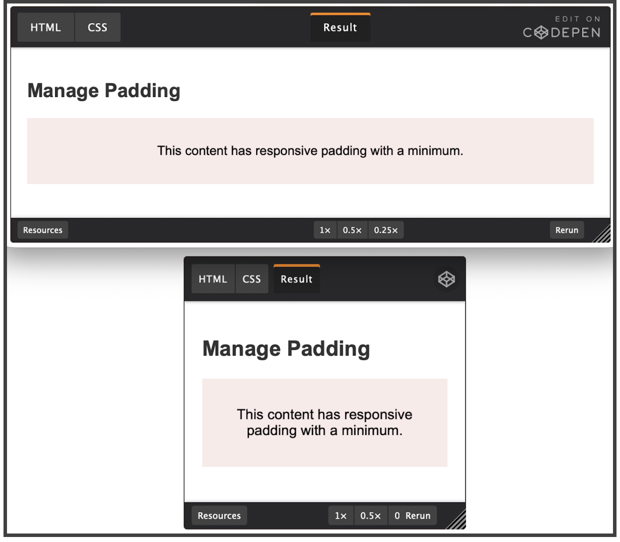
The main purpose of `max()` is to set lower limits (floors) for responsive values. One can think of it as: "Be at least as large as this value". The motivation behind it was to ensure accessibility and usability in fluid layouts without needing complex media queries. While properties like `min-width` solve this for widths, `max()` brings this logic to any CSS property. An important use case is typography: `font-size: max(5vw, 1rem)` guarantees that a font scales with the screen width, but never becomes so small that it is illegible. As illustrated in Listing 2.18, it acts as a safety net against excessive shrinking.

Example Code

```
:root {
  --content-width: 80rem; /* Max
    content width */
}

.inner-content {
  max-width: var(--content-width)
    ;
  margin: 0 auto;
  padding-left: max(2rem, calc
    ((100vw - var(--content-width)
    ) / 2));
  padding-right: max(2rem, calc
    ((100vw - var(--content-width)
    ) / 2));
}
```

Example Result



Manage Padding

This content has responsive padding with a minimum.

Manage Padding

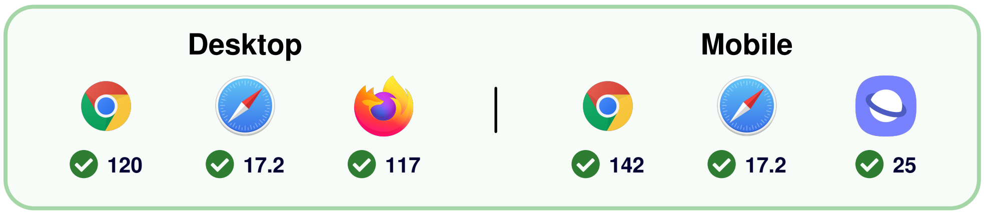
This content has responsive padding with a minimum.

Listing 2.18: Code example: `max()`.

2.6 Other

Other features of Modern CSS.

2.6.1 CSS Nesting



Baseline - December 2023*

This feature is well-established and works across many devices and browsers.

* This feature is considered safe to use in production environments.

The CSS nesting module does not provide much additional value to end users of web sites in comparison to many of the other modern CSS features. However, it does allow developers to write their CSS code in a more compact manner and also more intuitively at times. In general, CSS nesting allows nesting of rules within one another. The example in Listing 2.19 was inspired by the explainer of the corresponding W3C Editor's Draft [Atkins-Bittner 2025].

The main purposes of CSS nesting are increased readability, modularity, and maintainability of code. An additional advantage, compared to nesting implemented through preprocessors, is that it can be parsed natively by the browser and does not have to be compiled by a preprocessor beforehand [MDN 2025h].

Example Code

```
/* Without CSS Nesting: */
.foo {
  background-color: red;
}
.foo .bar {
  font-size: 2.0rem;
}

/* Now it is possible to express
    the same logic more compactly:
     */
.foo {
  background-color: red;
  .bar {
  font-size: 2.0rem;
  }
}
```

Listing 2.19: Code example: CSS nesting.

2.6.2 Scroll-Driven Animations

Desktop				Mobile		
✓ 115	✓ 26.0	⚑		✓ 142	✓ 26.0	✓ 23

Experimental - Partially available*

This feature is almost well-established and works across many devices and browsers already. It has been available in nearly all baseline browsers since September 2025. For Firefox, this feature is not yet available.

* This feature can be enabled for development purposes in Firefox via its experimental features setting.

CSS scroll-driven animations are a modern CSS feature that allows animations to be linked directly to a user's scroll progress. Instead of an animation running over a fixed period (e.g. 3 seconds), its progress is controlled by scrolling within a container. This opens up new, high-performance possibilities for interactive web experiences, such as parallax effects, progress indicators, or fading in elements as they enter the viewport [MDN 2025v].

Traditionally, such effects had to be implemented with JavaScript, which can often lead to performance issues as the script runs on the browser's main thread. With CSS scroll-driven animations, these calculations are optimized by the browser and often run independently of the main thread, resulting in smoother animations. Listing 2.20 demonstrates how animation timelines can be linked to scroll progress.

Example Code

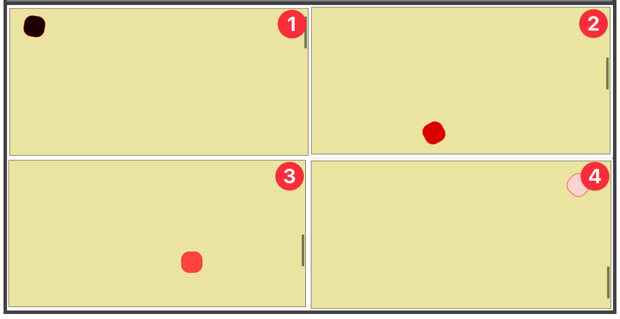
```
main {
  scroll-timeline:
    --main-timeline;
}

div {
  animation: background-animation
     linear;
  animation-timeline: scroll(
    nearest inline);
}

div::after {
  animation: shape-animation
    linear;
  animation-timeline:
    --main-timeline;
}
```

Listing 2.20: Code example: Scroll-driven animations.

Example Result



Chapter 3

NoNewNews Prototype

A prototype demonstration web site, NoNewNews, was developed to illustrate some of the more interesting features of Modern CSS. The web site resembles a typical news(paper) web site, such as The Boston Globe [Boston Globe 2025]. In addition to making use of as many modern CSS features as possible, the web site was made as responsive as possible. The complete code for the NoNewNews web site prototype can be found on GitHub [Burtscher et al. 2025a].

Web sites are generally based on the three foundational web technologies: HTML for structure, CSS for styling, and JavaScript for logic and behavior. In this case, TypeScript [Microsoft 2025] was chosen to work in a type-safe environment. To simplify the development process and focus on the goal of showcasing modern CSS features, SvelteKit [Svelte 2025] was chosen as the JavaScript framework. Svelte is known for simplifying the breakdown of code into components, allowing team members to work on different components independently.

NoNewNews has a navigation bar at the top, followed by the main content area underneath. The main content is structured in a grid comprised of varying cell sizes and positions. Following this section of the latest articles are sections for the various categories as well as some more information about the columnists. The bottom of the page contains a footer with various additional functionality, like creating an account, subscribing to the newspaper, and more.

3.1 Navbar

The navbar is at the top of the NoNewNews web site, as shown in Figures 3.1 and Figure 3.2 for wider and narrower screen widths, respectively. The code comprising this Svelte component can be found in the file `src/lib/components/Navbar.svelte` of the repository. CSS Properties (see Section 2.4.2) are used to simplify the addition of web site themes. One example is using `background-color: var(--navbar-background);` to avoid having to redefine the colors for light and dark mode (see Section 2.5.2) each time. For increased responsiveness, clamp (see Section 2.5.4) was used to specify the minimum, preferred, and maximum sizes for both font and container sizes. Additionally, container size queries (see Section 2.4.3) are used to swap to a mobile-optimized layout and use the available space as effectively as possible.

To avoid the navigation links from wrapping to the next line, the navbar sets `text-wrap: nowrap;`. This forces text to not wrap at all (see Section 2.2.1), even though the layout may become too small, but this behavior is intended in this case due to additional handling through container size queries. Internally, the Navbar component makes use of CSS nesting (see Section 2.6.1) to increase readability, maintainability, and adaptability of the styling-related code.

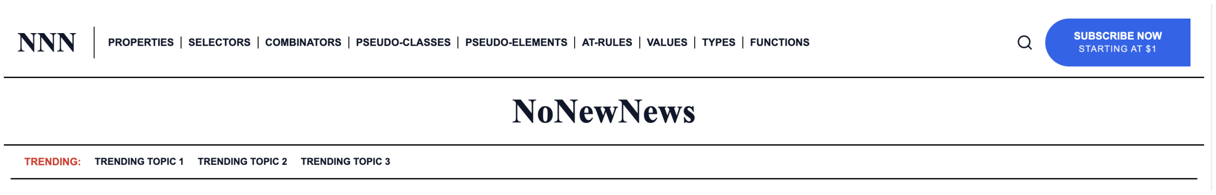

NNN
PROPERTIES | SELECTORS | COMBINATORS | PSEUDO-CLASSES | PSEUDO-ELEMENTS | AT-RULES | VALUES | TYPES | FUNCTIONS
SUBSCRIBE NOW
STARTING AT $1
NoNewNews
TRENDING: TRENDING TOPIC 1 TRENDING TOPIC 2 TRENDING TOPIC 3

Figure 3.1: NoNewNews: The navigation bar at the top of the web site.

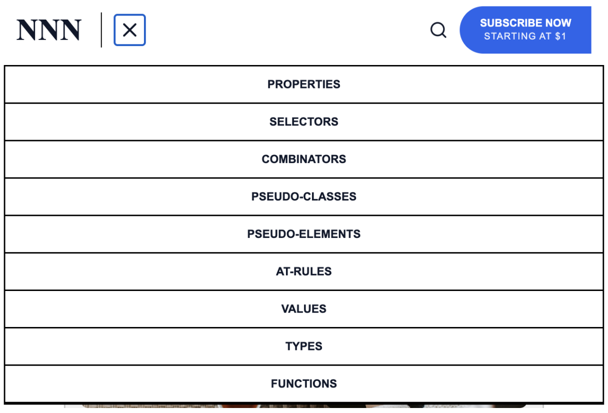


Figure 3.2: NoNewNews: The navigation bar on a narrower screen, with the corresponding burger menu opened to view the navigation links.

3.2 Main Content

The main content is divided into an article component and the actual home page route, located in the files `src/lib/components/Article.svelte` and `src/routes/+page.svelte` of the repository. A masonry grid layout (see Section 2.1.2) is used to position the articles on the home page, as can be seen in Figure 3.3. Note that only a few selected browsers, with corresponding feature flags enabled, support masonry layout at time of writing this survey paper.

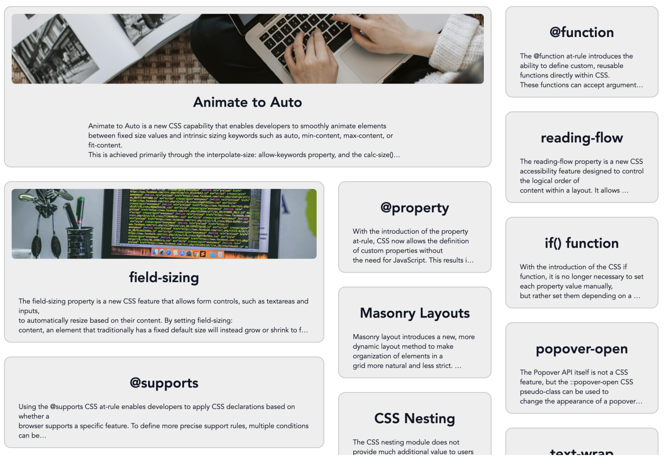
An `@supports` rule was used to first check for browser support (see Section 2.4.1). If the browser cannot utilize masonry, a more traditional grid with potential gaps between cells is shown. Should the reader click on any of the displayed articles, the popover API (see Section 2.1.1) handles the foreground display of the article. While reading through an article, scroll-driven animations (see Section 2.6.2) indicate the reading progress using a blue progress bar at the top of the popup. This can be seen in Figure 3.4.

The home page route also makes sure to define various color variables for the differing themes, such as: `--page-background: light-dark(#ffffff, #000000);` for the page background color. Furthermore, the popover behavior is deeply integrated both in the article component and the page route, as the applied styling when the popover is opened is applied through the article component, but disabling of the mouse behavior is done through the page route. This is done using a `:has` selector (see Section 2.3.1) to detect when the popover is open, like for example `:global(body:has([popover]:popover-open))`.

3.3 Footer

The Svelte component for the footer is located in the file `src/lib/components/footer.svelte` of the repository. Figure 3.5 shows how the footer looks in a wider desktop browser window. The mobile layout looks similar, but with closer spacing.

CSS custom properties are defined for the various colors used (see Section 2.4.2). Properties allow the specification of syntax `syntax: "<color>";`, default value `initial-value: #f8f9fa;`, and inheritance


Animate to Auto

Animate to Auto is a new CSS capability that enables developers to smoothly animate elements
between fixed size values and intrinsic sizing keywords such as auto, min-content, max-content, or
fit-content.
This is achieved primarily through the interpolate-size: allow-keywords property, and the calc-size()…

field-sizing

The field-sizing property is a new CSS feature that allows form controls, such as textareas and
inputs,
to automatically resize based on their content. By setting field-sizing:
content, an element that traditionally has a fixed default size will instead grow or shrink to f…

@supports

Using the @supports CSS at-rule enables developers to apply CSS declarations based on
whether a
browser supports a specific feature. To define more precise support rules, multiple conditions
can be…

@property

With the introduction of the property
at-rule, CSS now allows the definition
of custom properties without
the need for JavaScript. This results i…

Masonry Layouts

Masonry layout introduces a new, more
dynamic layout method to make
organization of elements in a
grid more natural and less strict. …

CSS Nesting

The CSS nesting module does not
provide much additional value to users

@function

The @function at-rule introduces the
ability to define custom, reusable
functions directly within CSS.
These functions can accept argument…

reading-flow

The reading-flow property is a new CSS
accessibility feature designed to control
the logical order of
content within a layout. It allows …

if() function

With the introduction of the CSS if
function, it is no longer necessary to set
each property value manually,
but rather set them depending on a …

popover-open

The Popover API itself is not a CSS
feature, but the ::popover-open CSS
pseudo-class can be used to
change the appearance of a popover…

Figure 3.3: NoNewNews: The main content of the home page, showing articles placed using masonry layout in supported browsers.

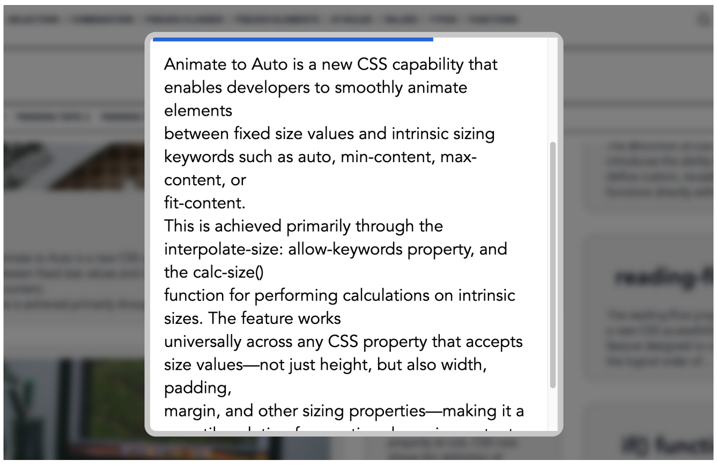

Animate to Auto is a new CSS capability that
enables developers to smoothly animate
elements
between fixed size values and intrinsic sizing
keywords such as auto, min-content, max-
content, or
fit-content.
This is achieved primarily through the
interpolate-size: allow-keywords property, and
the calc-size()
function for performing calculations on intrinsic
sizes. The feature works
universally across any CSS property that accepts
size values—not just height, but also width,
padding,
margin, and other sizing properties—making it a

Figure 3.4: NoNewNews: A news article in a popover window, with a reading progress bar.

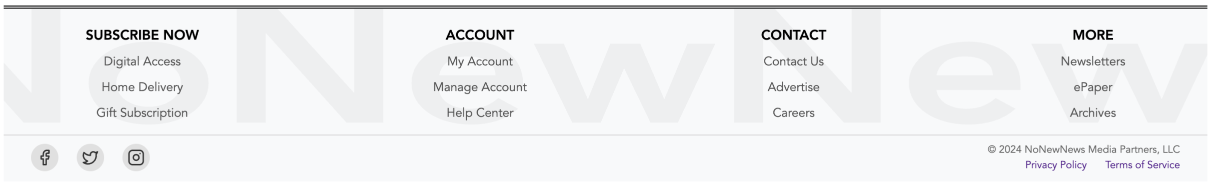


Figure 3.5: NoNewNews: The footer on a wider screen.

`inherits: false;` if required. Instead of `clamp()`, the footer makes use of the `min()` and `max()` functions to optimize the layout, by, for example, calculating padding accordingly: `left: calc(-1 * min(2rem, 5vw));`.

Chapter 4

Concluding Remarks

Modern CSS refers to the current state of the CSS ecosystem. During the last decade, CSS evolved dramatically with countless new features. Many of these, such as variables, nesting, functions, and conditionals were inspired by a previous generation of CSS preprocessors. Of the main preprocessor features, only mixins remain to be implemented directly in CSS.

Nowadays, the layout of web sites is highly flexible, with developers just needing to make use of CSS by styling and enabling popovers to direct the user's attention or creating dynamic grid-style layouts in pure CSS. Additionally, properties like `text-wrap` improve styling of typographical content, while Animate to Auto helps with animations, `field-sizing` with the layout of forms, and `reading-flow` to be able to improve accessibility. The aim to reduce JavaScript dependence is further targeted by adding `@function` and `@property` at-rules to add logic and typed variables to CSS. Furthermore, if-functions can now be written in CSS directly.

With the increased number of features supported, the `@supports` at-rule enables checking for browser-side support of specific features. Container size and container style queries allow conditional styling based on size or styling like `background-color`, respectively. CSS nesting allows developers to write more readable, modular, and maintainable code. Light and dark modes are easy to implement with the `light-dark()` function. Pseudo-classes simplify styling rules based on descendant elements. And, finally, the `linear()` function allows the creation of CSS-controlled transition curves, whereas scroll-driven animations allow animations to be purely linked to the user's scrolling progression.

This survey paper presented 20 of the more interesting features of Modern CSS, each accompanied by an example code snippet, (often) an illustration, and pointers to further examples and resources. The NoNewNews web site prototype provides a practical demonstration of how features of Modern CSS can be used to build a modern news web site.

Bibliography

Akinyemi, Ikeh [2025]. *Should you use if() functions in CSS?* Dec 2025. https://blog.logrocket.com/css-if-function-conditional-styling-2025/#if-vs-current-css-approaches (cited on page 18).

Albert, Quentin [2024]. *Do We Still Need CSS Preprocessors in 2024?* Feb 2024. https://beyond-the-cascade.com/do-we-still-need-css-preprocessors-in-2024/ (cited on page 12).

Andrew, Rachel [2023]. *Interop 2024 Brings More Features to Baseline*. Dec 2023. https://web.dev/blog/interop-2024-wrapup (cited on page 1).

Andrew, Rachel [2024]. *Following Grid and Flex Layouts with reading-flow*. W3C, Sep 2024. https://w3.org/2024/09/TPAC/demo-css.html (cited on page 8).

Atkins, Tab and Elika J. Etemad [2024]. *CSS Values and Units Module Level 5*. First Public Working Draft. W3C, Nov 2024. https://w3.org/TR/css-values-5/ (cited on page 9).

Atkins-Bittner, Tab [2025]. *CSS Nesting Module Level 1*. W3C, Nov 2025. https://drafts.csswg.org/css-nesting-1/#nest-selector (cited on page 22).

Baron, L. David and Tab Atkins [2024]. *Explainer for calc-size() and interpolate-size*. W3C CSSWG Drafts Repository. Aug 2024. https://github.com/w3c/csswg-drafts/blob/main/css-values-5/calc-size-explainer.md (cited on page 9).

Bejamas [2025]. *Learn CSS :has() Selector by Examples: 5 Top Use Cases*. Web Knowledge Hub, May 2025. https://bejamas.com/hub/guides/learn-css-has-selector-by-examples-top-use-cases (cited on page 10).

Boston Globe [2025]. *The Boston Globe*. 29 Nov 2025. https://bostonglobe.com/ (cited on page 25).

Burtscher, Christian, Jonas Glaser, Marcus Gugacs and Eva Haring [2025a]. *Mockup of a Responsive Newspaper Website*. Nov 2025. https://github.com/gugacs/nonewnews (cited on page 25).

Burtscher, Christian, Jonas Glaser, Marcus Gugacs and Eva Haring [2025b]. *Modern CSS Features GitHub*. Dec 2025. https://github.com/evaharing/Modern-CSS-features (cited on page 3).

CfD [2023]. *Scroll-driven animations*. Chrome for Developers, May 2023. https://developer.chrome.com/docs/css-ui/scroll-driven-animations (cited on page 23).

CfD [2024]. *Animate to height: auto; (and Other Intrinsic Sizing Keywords)*. Chrome for Developers, Sep 2024. https://developer.chrome.com/docs/css-ui/animate-to-height-auto (cited on page 9).

CfD [2025]. *Use CSS reading-flow for Logical Sequential Focus Navigation*. Chrome for Developers, May 2025. https://developer.chrome.com/blog/reading-flow (cited on page 8).

CIU [2025a]. *Can I use... Browser support tables for modern web technologies*. Can I use, 18 Nov 2025. https://caniuse.com/ (cited on page 3).

CIU [2025b]. *Clamp()*. Can I use, 18 Nov 2025. https://caniuse.com/?search=clamp (cited on pages 19–21).

CIU [2025c]. *CSS At-Rule: @function*. Can I use, 06 Nov 2025. https://caniuse.com/?search=%40function (cited on page 15).

CIU [2025d]. *CSS At-Rule: @property*. Can I use, 10 Nov 2025. https://caniuse.com/?search=@property (cited on page 12).

CIU [2025e]. *CSS At-Rule: @supports (Compatibility Prefix)*. Can I use, 10 Nov 2025. https://caniuse.com/?search=%40supports (cited on page 11).

CIU [2025f]. *CSS Function: calc-size()*. Can I use, 05 Nov 2025. https://caniuse.com/?search=calc-size%28%29 (cited on page 9).

CIU [2025g]. *CSS Function: if()*. Can I use, 10 Nov 2025. https://caniuse.com/?search=css+if%28%29 (cited on page 18).

CIU [2025h]. *CSS Nesting*. Can I use, 10 Nov 2025. https://caniuse.com/css-nesting (cited on page 22).

CIU [2025i]. *CSS Property: field-sizing*. Can I use, 06 Nov 2025. https://caniuse.com/?search=field-sizing (cited on page 7).

CIU [2025j]. *CSS Property: grid-template-rows: masonry*. Can I use, 10 Nov 2025. https://caniuse.com/?search=masonry (cited on page 5).

CIU [2025k]. *CSS Property: interpolate-size: allow-keywords*. Can I use, 05 Nov 2025. https://caniuse.com/?search=interpolate-size%3A+allow-keywords (cited on page 9).

CIU [2025l]. *CSS Property: light-dark*. Can I use, 10 Nov 2025. https://caniuse.com/?search=light-dark (cited on page 17).

CIU [2025m]. *CSS Property: reading-flow*. Can I use, 06 Nov 2025. https://caniuse.com/?search=reading-flow (cited on page 8).

CIU [2025n]. *CSS Property: text-wrap*. Can I use, 10 Nov 2025. https://caniuse.com/?search=text-wrap (cited on page 6).

CIU [2025o]. *CSS Selector: :has()*. Can I use, 11 Nov 2025. https://caniuse.com/?search=has%28%29 (cited on page 10).

CIU [2025p]. *CSS Selector: :linear()*. Can I use, 11 Nov 2025. https://caniuse.com/?search=linear%28%29 (cited on page 16).

CIU [2025q]. *CSS Selector: :popover-open*. Can I use, 10 Nov 2025. https://caniuse.com/?search=popover-open (cited on page 4).

CIU [2025r]. *CSS Selector: :Scroll-Driven Animations*. Can I use, 11 Nov 2025. https://caniuse.com/?search=scroll+driven+animations (cited on page 23).

CIU [2025s]. *CSS Selector: Container Size Queries*. Can I use, 11 Nov 2025. https://caniuse.com/?search=container+size+queries (cited on page 13).

CIU [2025t]. *CSS Selector: Container Style Queries*. Can I use, 11 Nov 2025. https://caniuse.com/?search=container+style+queries (cited on page 14).

Coyier, Chris [2025]. *What You Need to Know about Modern CSS (2025 Edition)*. Frontend Masters, Sep 2025. https://frontendmasters.com/blog/what-you-need-to-know-about-modern-css-2025-edition/ (cited on pages 7, 9).

CSS-Tricks [2024]. *Container Size Queries*. Jun 2024. https://css-tricks.com/css-container-queries/#container-queries-properties-amp-values (cited on page 13).

CSS-Tricks [2025]. *linear()*. Jul 2025. https://css-tricks.com/almanac/functions/l/linear/ (cited on page 16).

Devographics [2025]. *State of CSS: Features*. Nov 2025. https://2025.stateofcss.com/en-US/features/ (cited on page 11).

Etemad, Elika J. and Tab Atkins [2025]. *CSS Display Module Level 4*. Editor's Draft. W3C, Nov 2025. https://w3.org/TR/css-display-4/ (cited on page 8).

Freepik [2025]. *Firefox Browser Logo Icon*. Nov 2025. https://flaticon.com/free-icon/firefox_5968827 (cited on page 3).

Graham, Geoff [2024]. *:popover-open*. Jun 2024. https://css-tricks.com/almanac/pseudo-selectors/p/popover-open/ (cited on page 4).

JJ, Carlos [2025]. *Google Chrome Browser Logo Icon*. Nov 2025. https://iconfinder.com/icons/87865/chrome_icon (cited on page 3).

MDN [2025a]. *:popover-open*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Selectors/:popover-open (cited on page 4).

MDN [2025b]. *@function*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/@function (cited on page 15).

MDN [2025c]. *@property*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/@property (cited on page 12).

MDN [2025d]. *@supports*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/@supports (cited on page 11).

MDN [2025e]. *clamp()*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/clamp (cited on page 19).

MDN [2025f]. *Container Queries*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/At-rules/@container (cited on pages 13–14).

MDN [2025g]. *Container Style Queries*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Containment/Container_size_and_style_queries (cited on page 14).

MDN [2025h]. *CSS Nesting*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Nesting (cited on page 22).

MDN [2025i]. *CSS Reference*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference (cited on page 3).

MDN [2025j]. *Experimental, Deprecated, and Obsolete*. Mozilla Developer Network, Dec 2025. https://developer.mozilla.org/en-US/docs/MDN/Writing_guidelines/Experimental_deprecated_obsolete#experimental (cited on page 2).

MDN [2025k]. *field-sizing*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Properties/field-sizing (cited on page 7).

MDN [2025l]. *has()*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Selectors/:has (cited on page 10).

MDN [2025m]. *Houdini APIs*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/API/Houdini_APIs (cited on page 12).

MDN [2025n]. *if()*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/if (cited on page 18).

MDN [2025o]. *light-dark()*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/color_value/light-dark (cited on page 17).

MDN [2025p]. *linear()*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/easing-function/linear (cited on page 16).

MDN [2025q]. *Masonry layout*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Grid_layout/Masonry_layout (cited on page 5).

MDN [2025r]. *max()*. Mozilla Developer Network, Dec 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/max (cited on page 21).

MDN [2025s]. *min()*. Mozilla Developer Network, Dec 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Values/min (cited on page 20).

MDN [2025t]. *Popover API*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/API/Popover_API#css_features (cited on page 4).

MDN [2025u]. *reading-flow*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Properties/reading-flow (cited on page 8).

MDN [2025v]. *Scroll-Driven Animations*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Scroll-driven_animations (cited on page 23).

MDN [2025w]. *text-wrap*. Mozilla Developer Network, Nov 2025. https://developer.mozilla.org/en-US/docs/Web/CSS/Reference/Properties/text-wrap (cited on page 6).

Microsoft [2025]. *TypeScript is JavaScript with Syntax for Types*. Nov 2025. https://typescriptlang.org/ (cited on page 25).

Nguyen, Tim and Luke Warlow [2025]. *CSS Form Control Sytling Module Level 1*. Editor's Draft. W3C, Oct 2025. https://drafts.csswg.org/css-forms-1/ (cited on page 7).

Rodriguez, Petro [2020]. *Linearly Scale font-size with CSS clamp() Based on the Viewport*. Sep 2020. https://css-tricks.com/linearly-scale-font-size-with-css-clamp-based-on-the-viewport/ (cited on page 19).

Samsung [2025]. *Samsung Internet Logo*. Nov 2025. https://commons.wikimedia.org/wiki/File:Samsung_Internet_logo.svg (cited on page 3).

Shoyombo, Gabriel [2025a]. *Masonry In CSS: Should Grid Evolve Or Stand Aside For A New Module?* May 2025. https://smashingmagazine.com/2025/05/masonry-css-should-grid-evolve-stand-aside-new-module/ (cited on page 5).

Shoyombo, Gabriel [2025b]. *max()*. May 2025. https://css-tricks.com/almanac/functions/m/max/ (cited on page 21).

Shoyombo, Gabriel [2025c]. *min()*. May 2025. https://css-tricks.com/almanac/functions/m/min/ (cited on page 20).

Stockio [2025]. *Safari Browser Logo Icon*. Nov 2025. https://flaticon.com/free-icon/safari_668290 (cited on page 3).

Suzanne, Miriam E. [2023]. *Proposal: Custom CSS Functions and Mixins*. W3C CSSWG Drafts Repository, Issue 9350. Sep 2023. https://github.com/w3c/csswg-drafts/issues/9350 (cited on page 15).

Suzanne, Miriam E. and Tab Atkins [2025]. *CSS Functions and Mixins Module*. Editor's Draft. W3C, May 2025. https://w3.org/TR/css-mixins-1/ (cited on page 15).

Svelte [2025]. *Web Development for the Rest of Us*. Nov 2025. https://svelte.dev/ (cited on page 25).

web.dev [2025]. *Baseline*. 02 Dec 2025. `https://web.dev/baseline` (cited on page 1).

WPT [2001]. *Interop 2021 Dashboard*. The web-platform-tests Project, 2001. `https://wpt.fyi/interop-2021` (cited on page 1).

WPT [2005]. *Interop 2025 Dashboard*. The web-platform-tests Project, 2005. `https://wpt.fyi/interop-2025` (cited on page 1).