



Lecture plan

- 💡 CSS Cascade. CSS Specificity
- 💡 CSS Custom Properties
- 💡 CSS Layouts – Flexbox & Grid
- 💡 CSS Modules & CSS-in-JS
- 💡 Technologies

CSS Cascade



CSS Cascade definition

“ “

The cascade takes an unordered list of declared values for a given property on a given element, sorts them by their declaration's precedence as determined below, and outputs a single cascaded value.

[CSS WG](#)

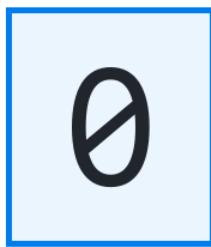
Cascade sorting order

- 👉 Origin and Importance
- 👉 Context
- 👉 Specificity
- 👉 Order of Appearance

[Cascade Sorting Order](#)

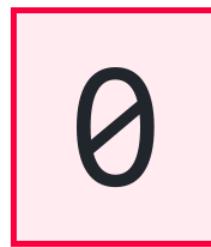
Specificity

Inline Style



Thousands

ID



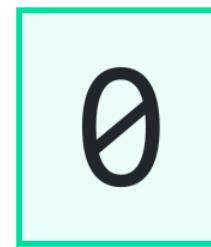
Hundreds

Class



Tens

Element



Ones

Highest



Lowest

Specificity Magnitude

[Specificity Calculator](#)

Order of Appearance

The last declaration in document order wins.

CSS Specificity Quiz 1/8

01. section p {	01. <section class="section" id="section">
02. color: ■ red;	02. <h1>Title</h1>
03. }	03. <p class="p1">Paragraph 1</p>
04. .p1 {	04. <p class="p2">Paragraph 2</p>
05. color: ■ green;	05. </section>
06. }	

💡 The specificity of section p is 0.0.2 and .p1 is 0.1.0

CSS Specificity Quiz 2/8

```
01. .section p {      01. <section class="section" id="section">
02.   color: ■ red;  02.   <h1>Title</h1>
03. }                  03.   <p class="p1">Paragraph 1</p>
04. .p1 {              04.   <p class="p2">Paragraph 2</p>
05.   color: ■ green; 05. </section>
06. }
```

💡 The specificity of **.section1 p** is 0.1.1 and .p1 is 0.1.0

CSS Specificity Quiz 3/8

```
01. #section {          01. <section class="section" id="section">
02.   color: ■ red;    02.   <h1>Title</h1>
03. }                  03.   <p class="p1">Paragraph 1</p>
04. .p1 {             04.   <p class="p2">Paragraph 2</p>
05.   color: ■ green; 05. </section>
06. }
```

💡 p would inherit its parent's color only if there is no color applied directly to paragraphs

CSS Specificity Quiz 4/8

```
01. #section * {      01. <section class="section" id="section">
02.   color: ■ red;  02.   <h1>Title</h1>
03. }                  03.   <p class="p1">Paragraph 1</p>
04. .p1 {             04.   <p class="p2">Paragraph 2</p>
05.   color: ■ green; 05. </section>
06. }
```

💡 The specificity of **#section1 *** is 1.0.0 and .p1 is 0.1.0

CSS Specificity Quiz 5/8

```
01. section * {          01. <section class="section" id="section">
02.   color: ■ red;    02.   <h1>Title</h1>
03. }                  03.   <p class="p1">Paragraph 1</p>
04. p {                04.   <p class="p2">Paragraph 2</p>
05.   color: ■ green; 05. </section>
06. }
```

💡 The specificity of section * is 0.0.1. The specificity of p is also 0.0.1 but it comes later in the stylesheet .

CSS Specificity Quiz 6/8

01. section .p1 {	01. <section class="section" id="section">
02. color: ■ red;	02. <h1>Title</h1>
03. }	03. <p class="p1">Paragraph 1</p>
04. section :nth-child(2) {	04. <p class="p2">Paragraph 2</p>
05. color: ■ green;	05. </section>
06. }	

💡 The specificity of section .p1 is 0.1.1. The specificity of **section :nth-child(2)** is also 0.1.1 but it comes later ¹² in the stylesheet .

CSS Specificity Quiz 7/8

```
01. section > p {      01. <section class="section" id="section">
02.   color: ■ red;    02.   <h1>Title</h1>
03. }                  03.   <p class="p1">Paragraph 1</p>
04. section p {        04.   <p class="p2">Paragraph 2</p>
05.   color: ■ green; 05. </section>
06. }
```

💡 The specificity of section > p is 0.0.2. The specificity of **section p** is also 0.0.2 but it comes later in the stylesheet .

CSS Specificity Quiz 8/8

```
01. .red {          01. <section class="section" id="section">
02.   color: ■ red;  02.   <p class="green red">Paragraph</p>
03. }              03. </section>
04. .green {
05.   color: ■ green;
06. }
```

💡 The specificity of `.red` is 0.1.0. The specificity of `.green` is also 0.1.0 but it comes later in the stylesheet.

CSS Custom Properties

CSS Custom Properties usecases

- 👉 Create complex CSS properties
- 👉 Create color themes
- 👉 Use it with JavaScript (e.g. animation)

CSS Custom Properties

```
01. :root {  
02.   --bg-color: ■ plum; // declare a custom variable  
03.   --base-padding: 0.5rem;  
04. }  
05. .Component {  
06.   background-color: var(--bg-color); // use the custom variable  
07.   padding: calc(2 * var(--base-padding));  
08. }
```



[Practice] CSS Custom Properties

1. Convert it to CSS Custom Properties
2. (optional) Make it responsive - change base font size on different screen sizes

type-scale.com

Comparison to CSS preprocessors

Preprocessors variables are...

- 👉 Both global & local
- 👉 Static
- 👉 Not flexible

CSS Custom Properties are...

- 👉 Global
- 👉 Dynamic
- 👉 Accessible from JS land
- 👉 CSS as API ([example](#))

JavaScript API

```
01. let element = document.querySelector("#element")
02. let styles = getComputedStyle(element)
03. let size = +styles.getPropertyValue("--base-size")
04. element.style.setProperty("--base-size", size + 4)
```

CSS Custom Properties

- 👍 Native solution
- 👍 Obeyed to CSS Cascade
- 👍 Dynamic nature
- 👍 Accessible via JS
- 👍 Could be polyfilled in some base cases
- 👍 Good compilation target
- 🤔 Performance

CSS Layouts

Flexbox & Grid

Responsive Design could be achieved with...

- 👉 Typography
- 👉 Media
- 👉 Meta tag viewport
- 👉 Media Queries
- 👉 CSS Flexbox
- 👉 CSS Grids
- 👉 [CSS Container Queries](#)

CSS Flexbox API

- flex-direction
- justify-content
- align-items
- order
- flex-grow, flex-shrink, flex-basis
- flex-wrap
- align-content

initial state

```
01. <div class="container">  
02.     <div class="item">First</div>  
03.     <div class="item">Second</div>  
04.     <div class="item">Third</div>  
05.     <div class="item">Fourth</div>  
06. </div>
```

initial state

First

Second

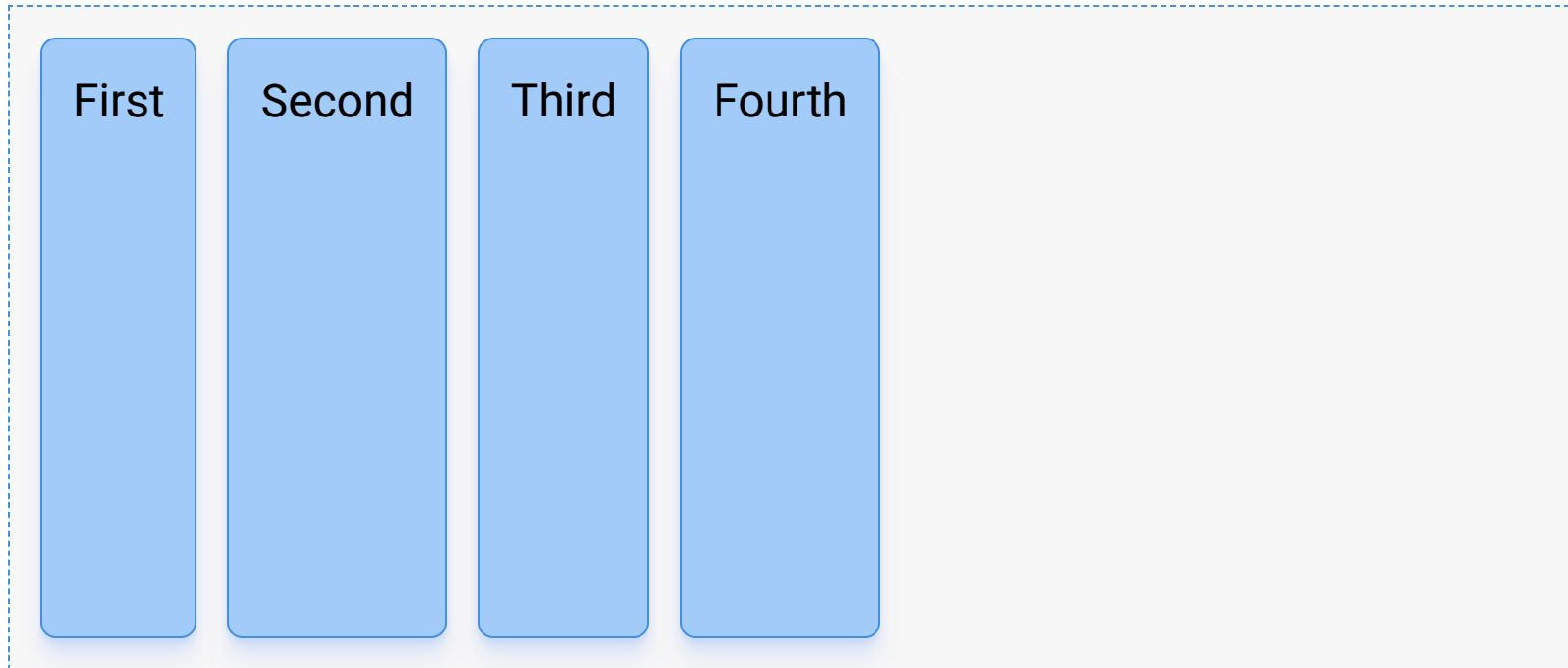
Third

Fourth

display: flex

```
01. .container {  
02.     display: flex;  
03. }
```

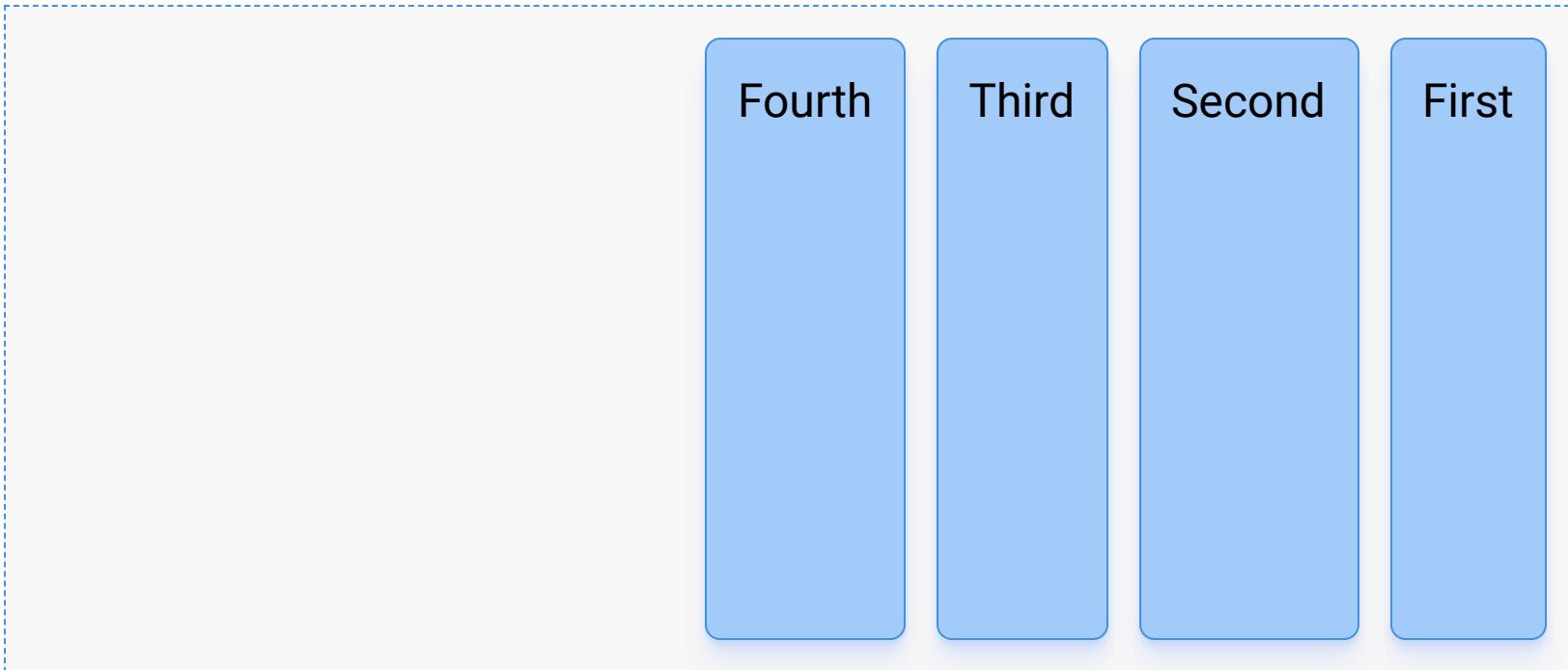
display: flex



flex-direction: row-reverse

```
01. .container {  
02.     display: flex;  
03.     flex-direction: row-reverse;  
04. }
```

flex-direction: row-reverse



flex-direction: column

```
01. .container {  
02.     display: flex;  
03.     flex-direction: column;  
04. }
```

flex-direction: column

First

Second

Third

Fourth

flex-direction: column-reverse

```
01. .container {  
02.     display: flex;  
03.     flex-direction: column-reverse;  
04. }
```

flex-direction: column-reverse

Fourth

Third

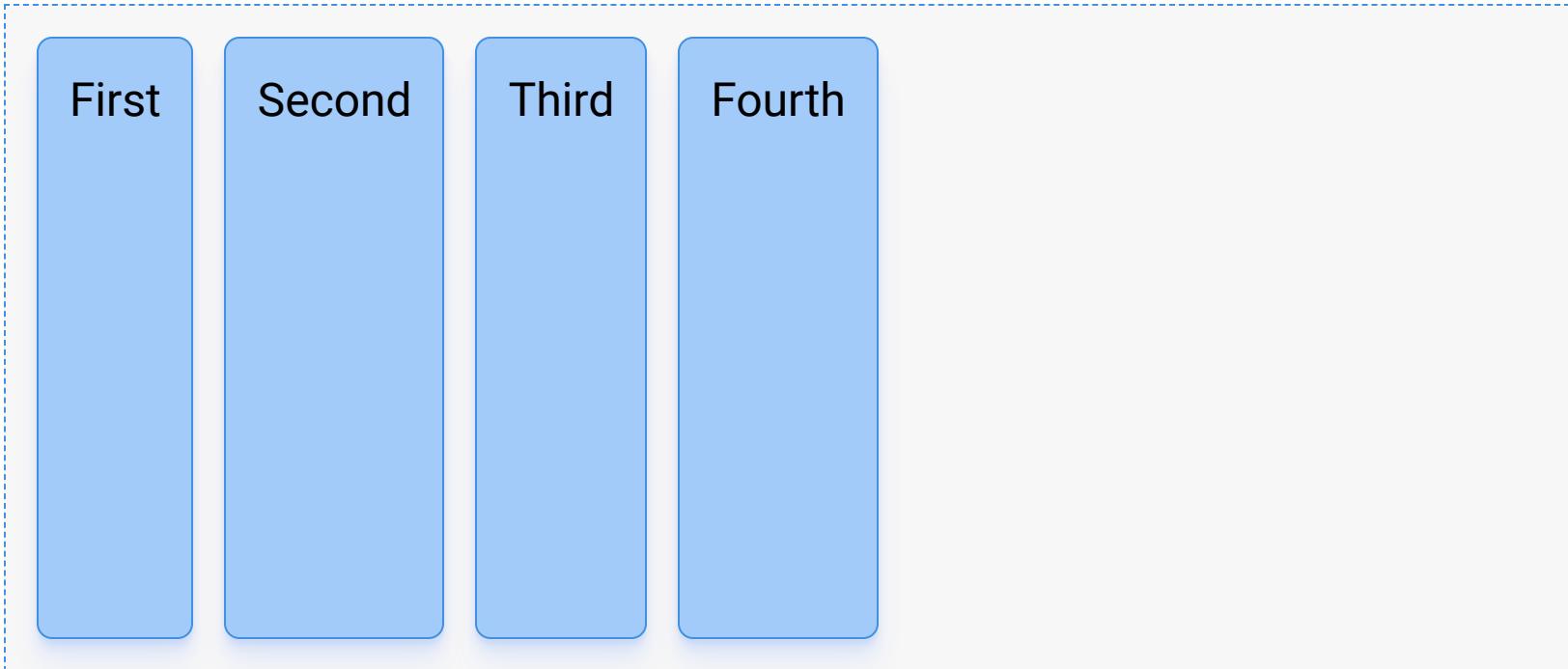
Second

First

justify-content: flex-start (default)

```
01. .container {  
02.     display: flex;  
03.     justify-content: flex-start;  
04. }
```

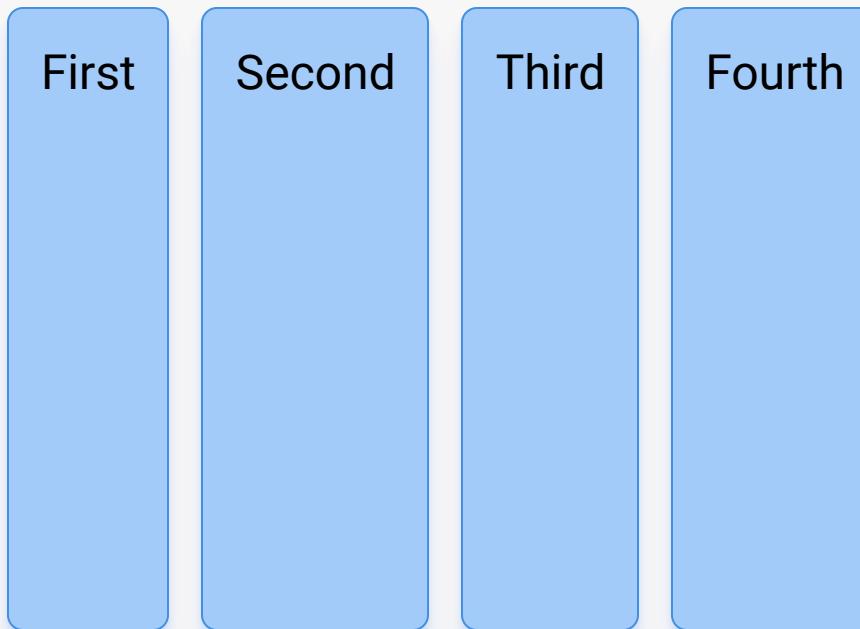
justify-content: flex-start (default)



justify-content: center

```
01. .container {  
02.     display: flex;  
03.     justify-content: center;  
04. }
```

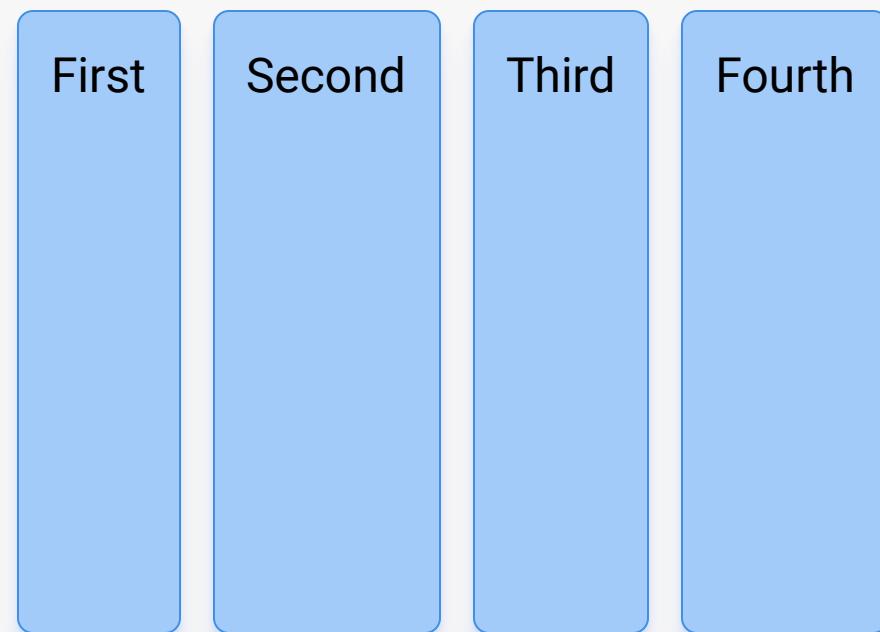
justify-content: center



justify-content: flex-end

```
01. .container {  
02.     display: flex;  
03.     justify-content: flex-end;  
04. }
```

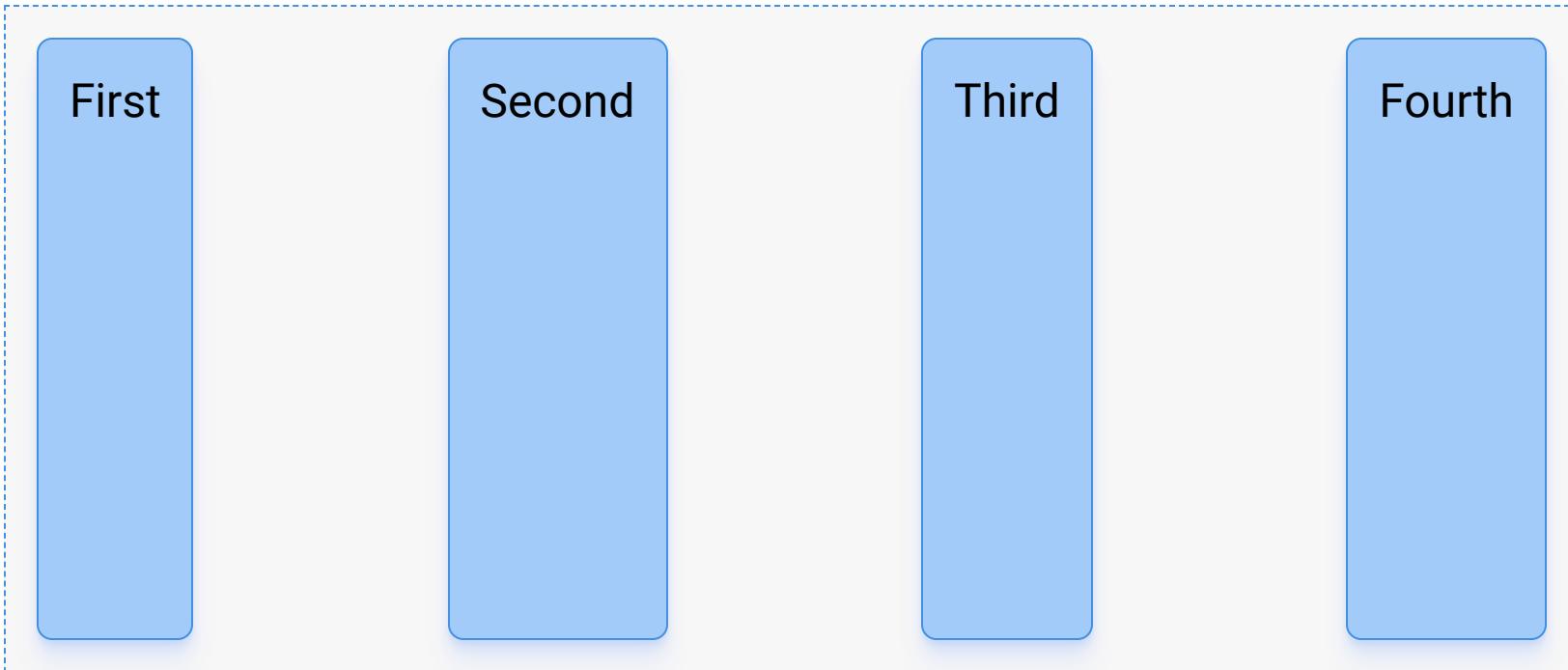
justify-content: flex-end



justify-content: space-between

```
01. .container {  
02.     display: flex;  
03.     justify-content: space-between;  
04. }
```

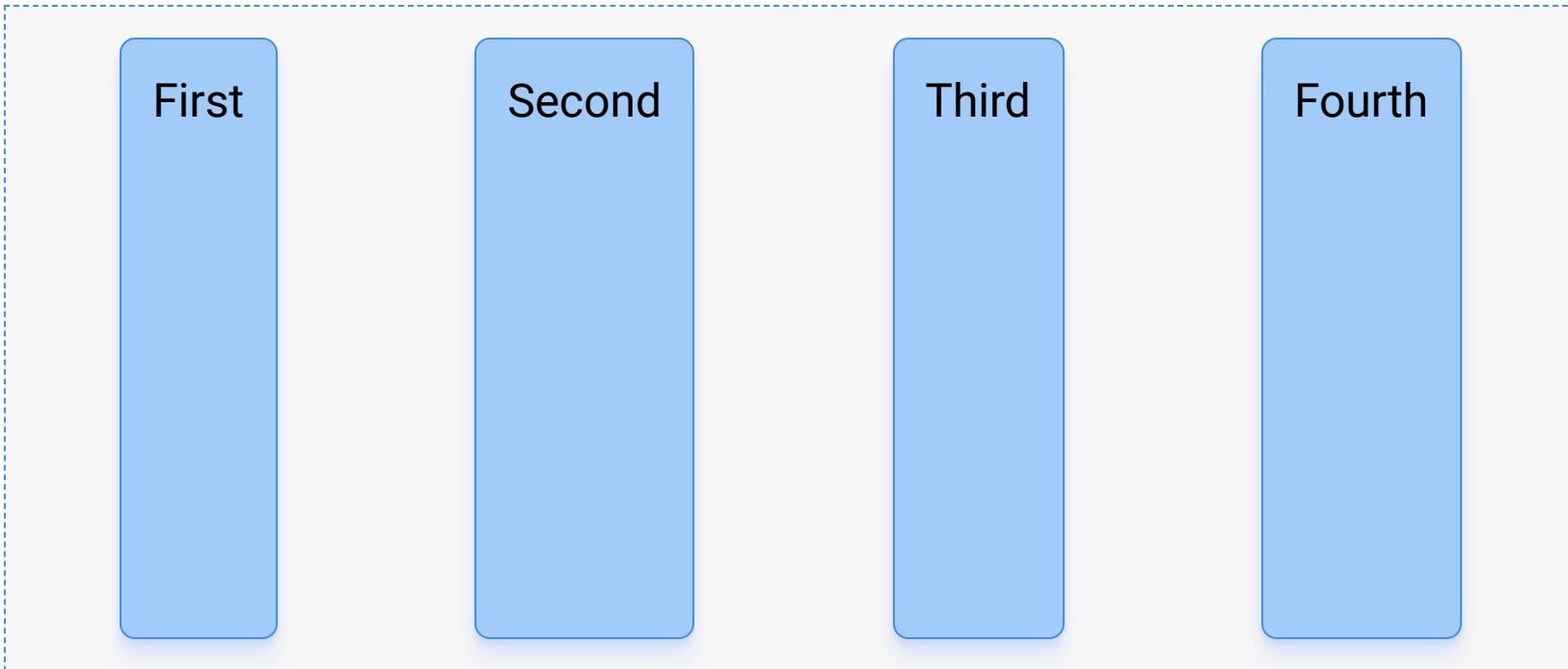
justify-content: space-between



justify-content: space-around

```
01. .container {  
02.     display: flex;  
03.     justify-content: space-around;  
04. }
```

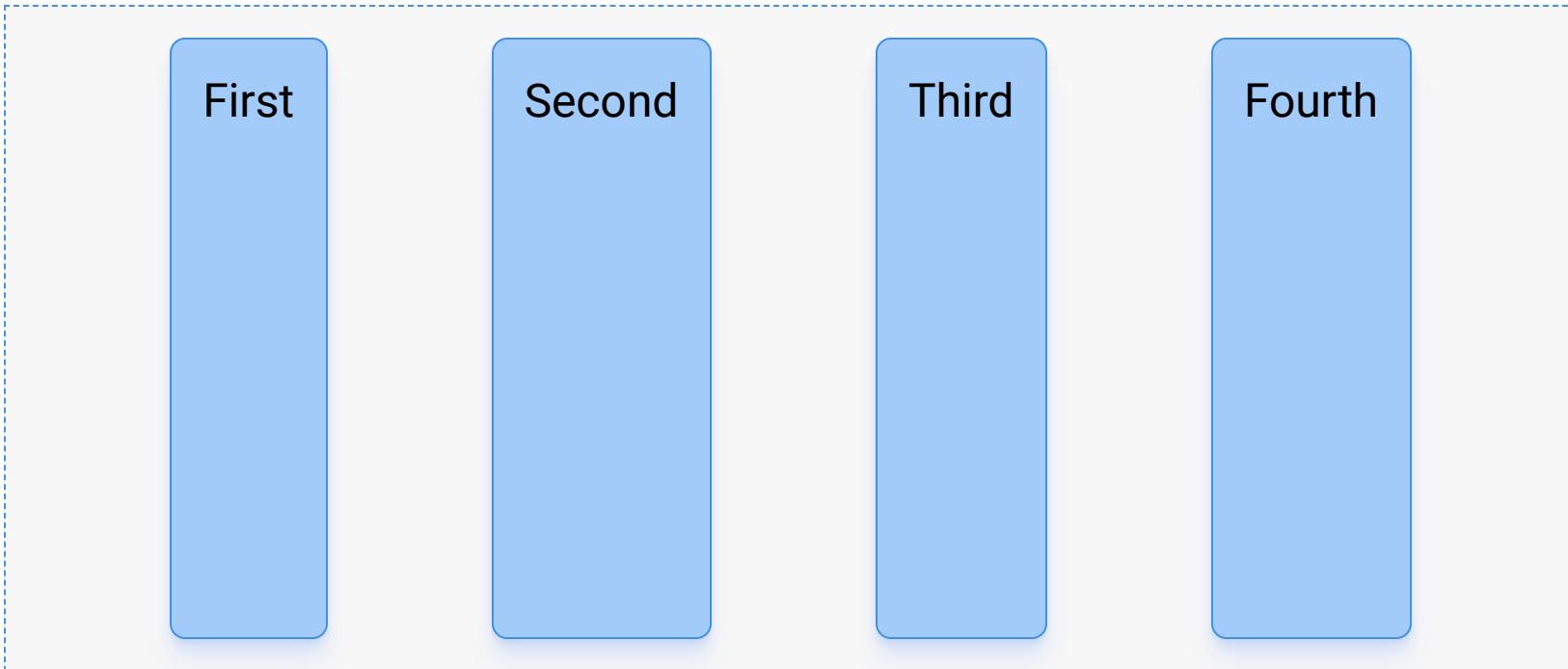
justify-content: space-around



justify-content: space-evenly

```
01. .container {  
02.     display: flex;  
03.     justify-content: space-evenly;  
04. }
```

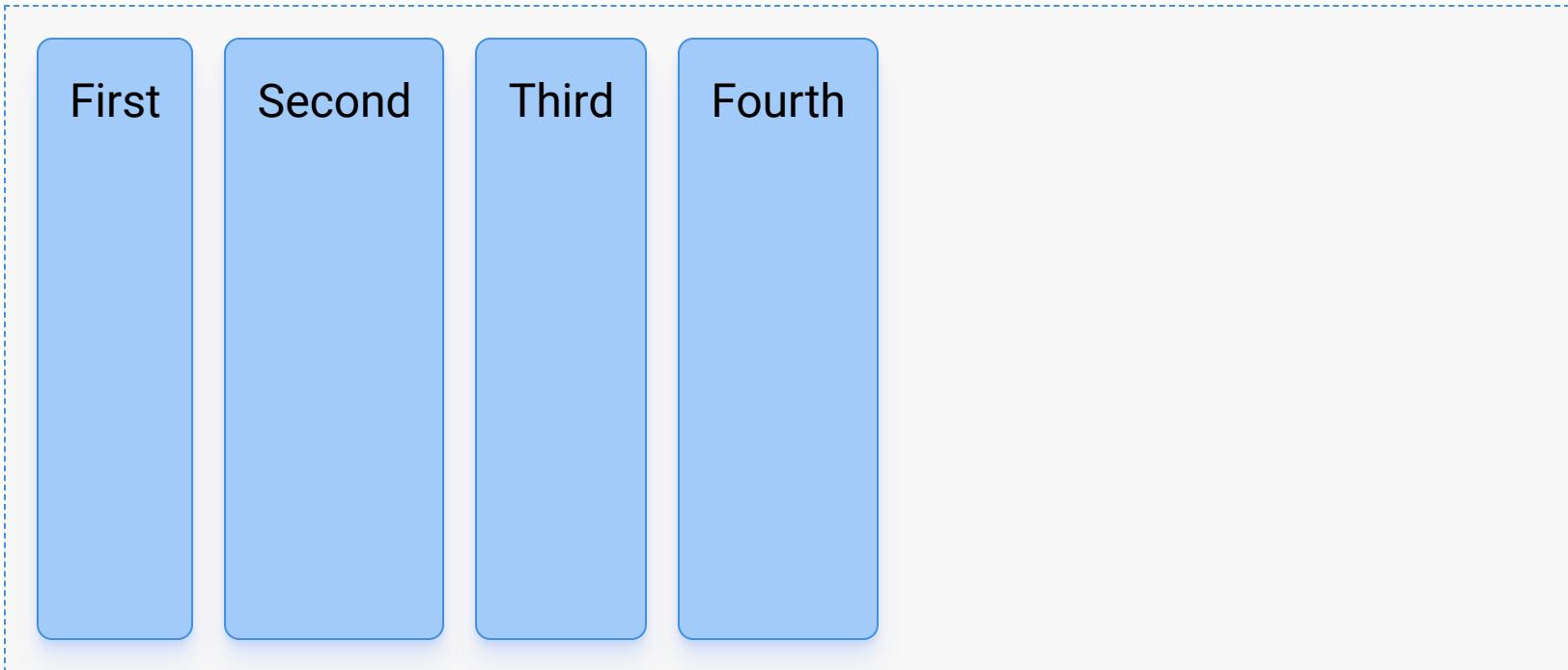
justify-content: space-evenly



align-items: stretch (default)

```
01. .container {  
02.   display: flex;  
03.   align-items: stretch;  
04. }
```

align-items: stretch (default)



align-items: flex-start

```
01. .container {  
02.     display: flex;  
03.     align-items: flex-start;  
04. }
```

align-items: flex-start

First

Second

Third

Fourth

align-items: center

```
01. .container {  
02.     display: flex;  
03.     align-items: center;  
04. }
```

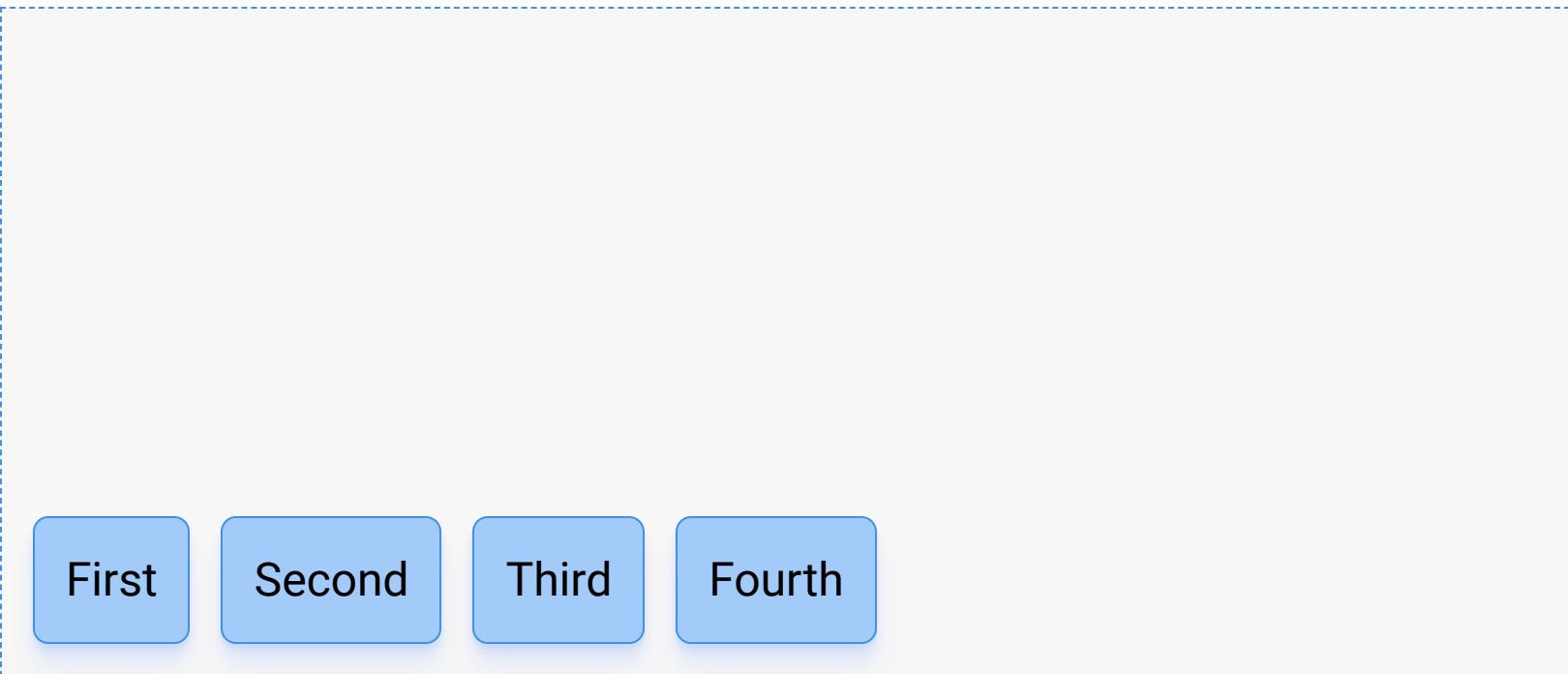
align-items: center



align-items: flex-end

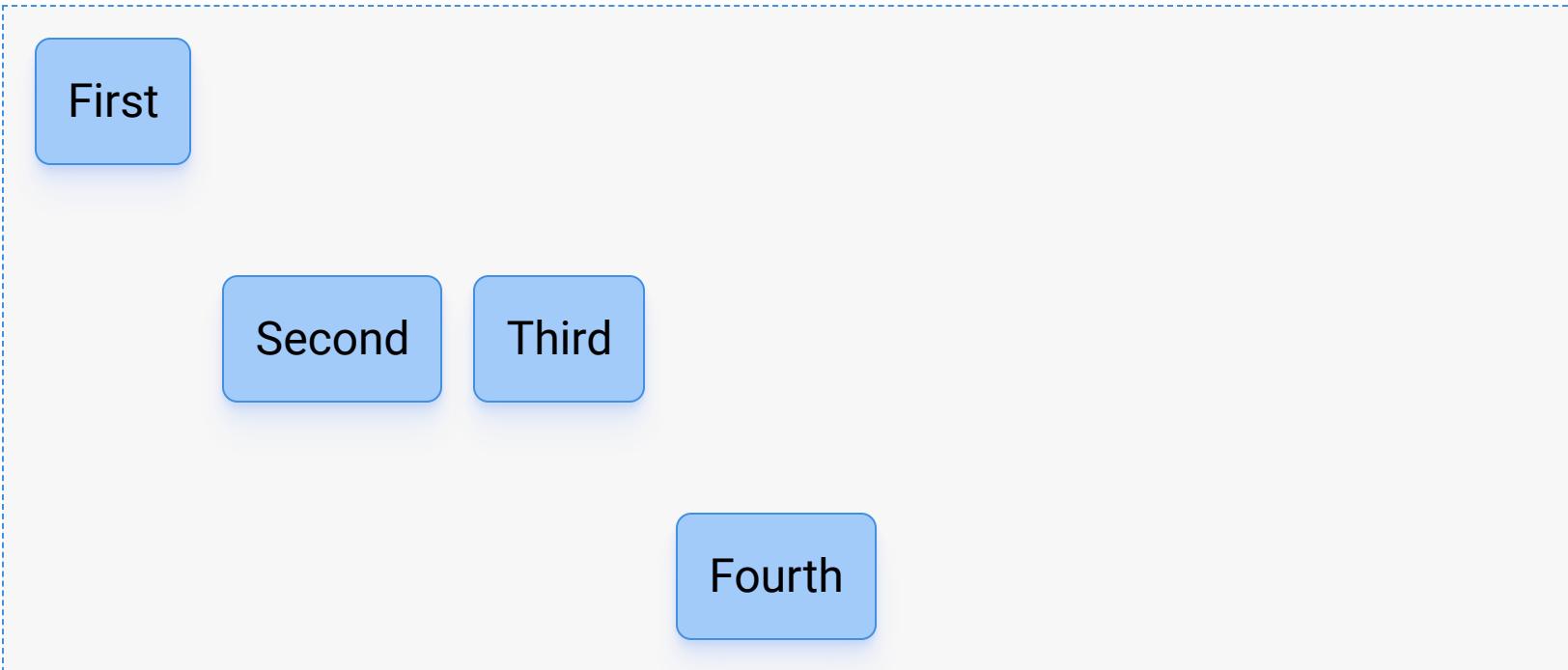
```
01. .container {  
02.     display: flex;  
03.     align-items: flex-end;  
04. }
```

align-items: flex-end



```
01. .container {  
02.     display: flex;  
03.     align-items: center;  
04. }  
05. .item:nth-child(1) {  
06.     align-self: flex-start;  
07. }  
08. .item:nth-child(4) {  
09.     align-self: flex-end;  
10. }
```

align-self



Order

```
01. .item {  
02.     order: 1;  
03. }  
04. .item:nth-child(1) {  
05.     order: 2;  
06. }  
07. .item:nth-child(4) {  
08.     order: -1;  
09. }
```

Order

Fourth

Second

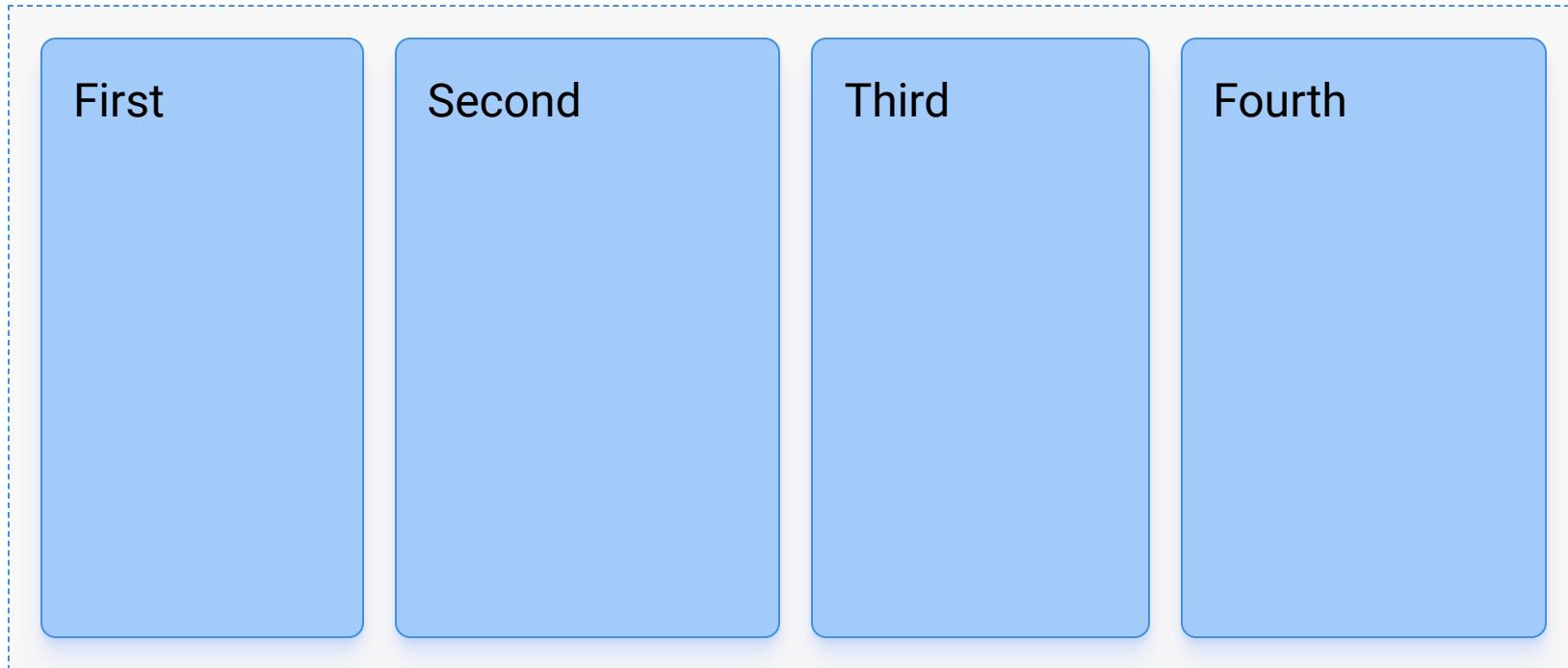
Third

First

flex-grow

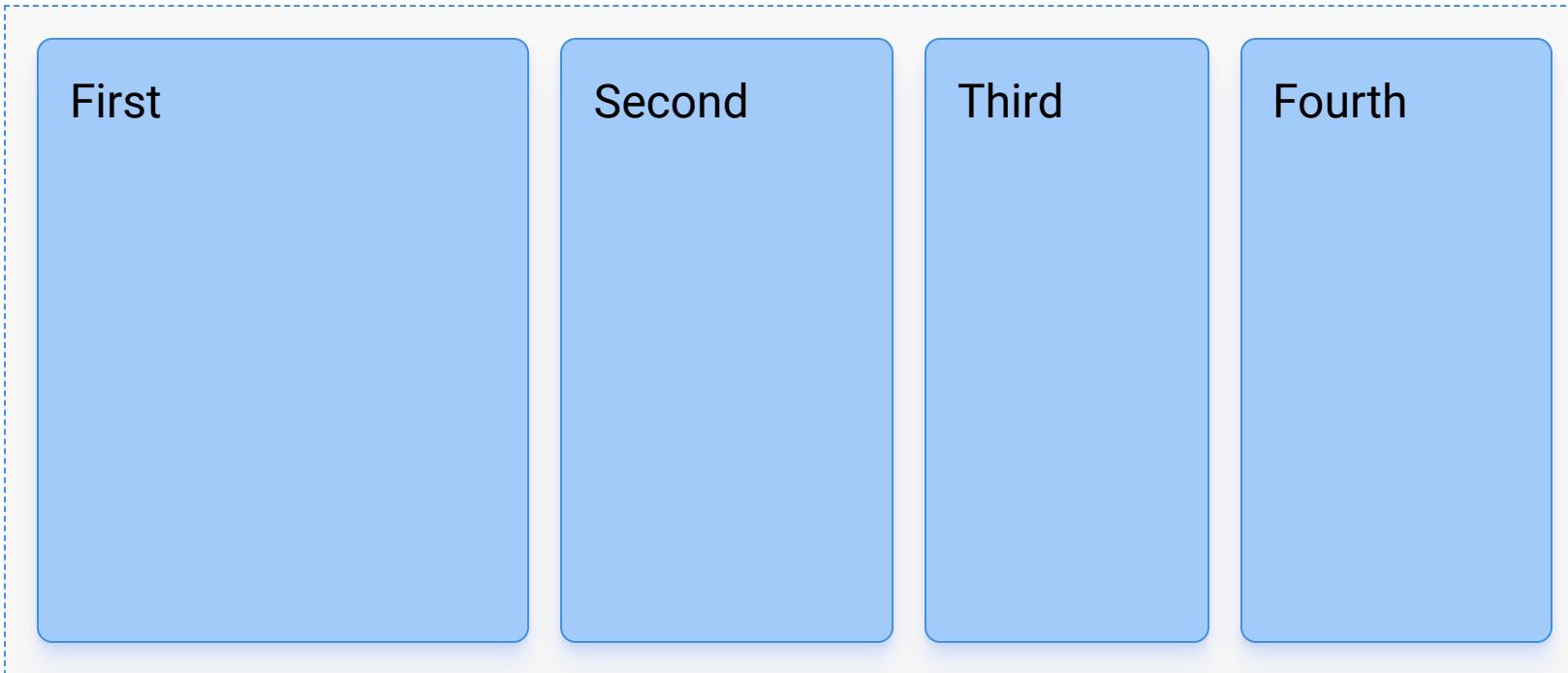
```
01. .container {  
02.     display: flex;  
03. }  
04. .item {  
05.     flex-grow: 0; /* By default */  
06.     flex-grow: 1;  
07. }
```

flex-grow



```
01. .container {  
02.     display: flex;  
03. }  
04. .item {  
05.     flex-grow: 1;  
06. }  
07. .item:nth-child(1) {  
08.     flex-grow: 3;  
09. }
```

flex-grow



flex-shrink

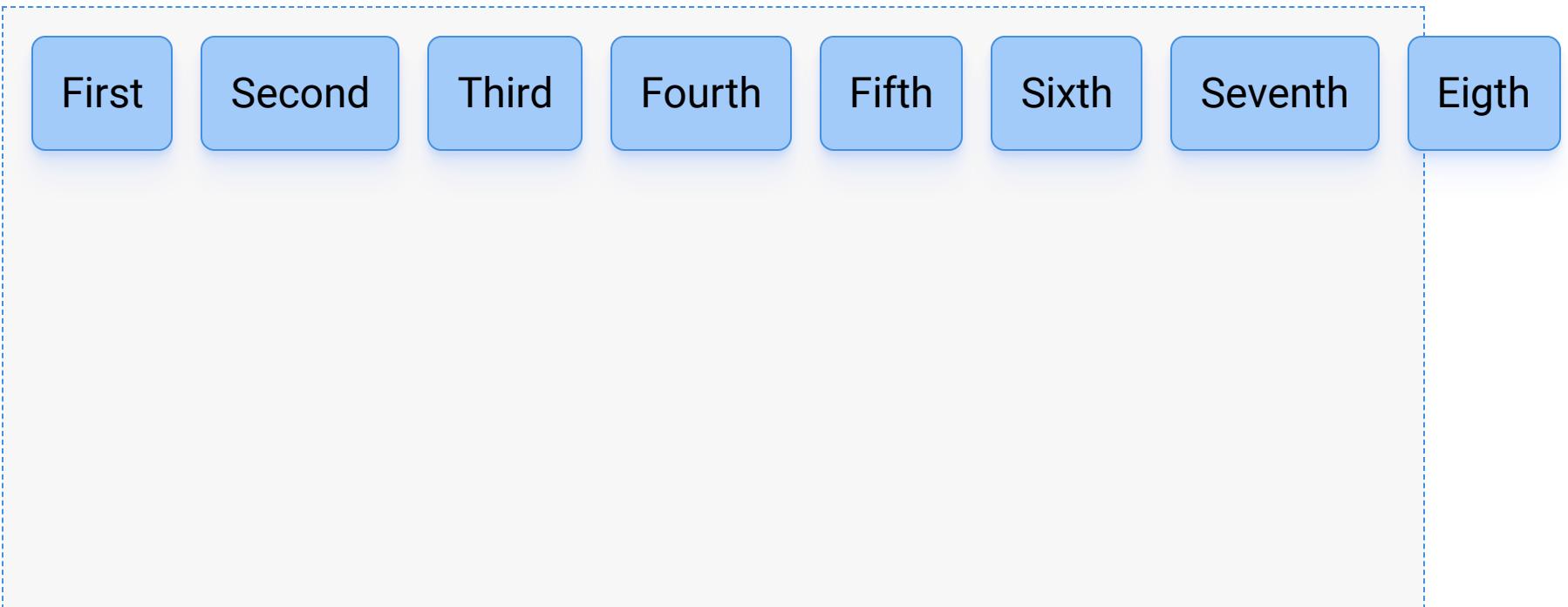
[Example](#)

flex-basis

[Codesandbox](#)

[Difference between flex-basis and width](#)

flex-wrap: nowrap



flex-wrap: wrap

```
01. .container {  
02.   display: flex;  
03.   align-items: flex-start;  
04.   flex-wrap: wrap;  
05. }
```

flex-wrap: wrap



align-content

```
01. .container {  
02.     display: flex;  
03.     align-items: flex-start;  
04.     flex-wrap: wrap;  
05.     align-content: center;  
06. }
```

align-content



Flexbox bugs

 **Not all is well in flexbox land**

[Flexbugs](#)

CSS Grid API

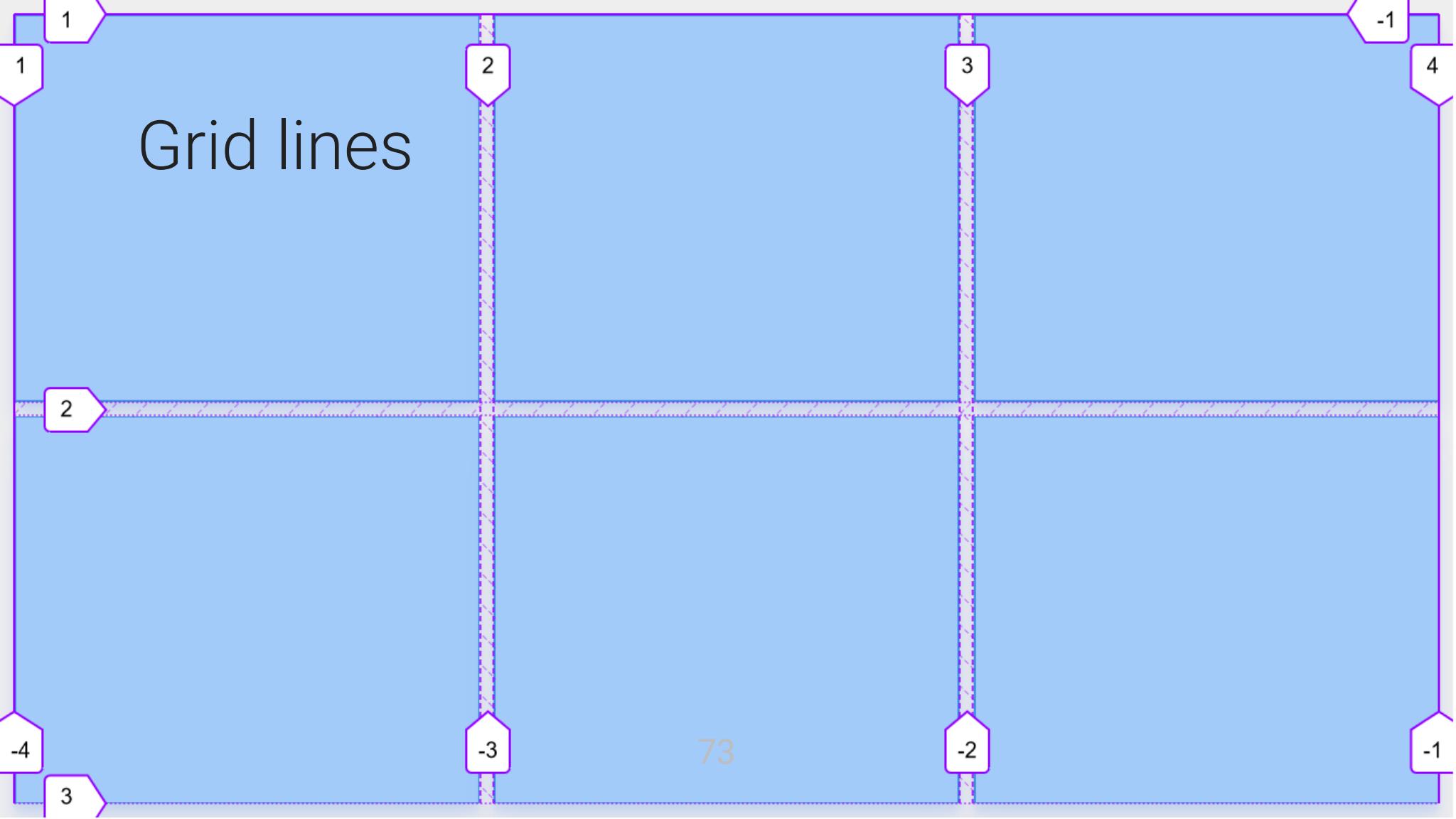
- grid-template-columns
- grid-template-rows
- grid-column
- grid-row
- grid-template-area / grid-area
- gap
- align-items
- justify-content
- align-content
- grid-auto-columns
- grid-auto-rows
- grid-auto-flow

[Complete guide to Grid](#)

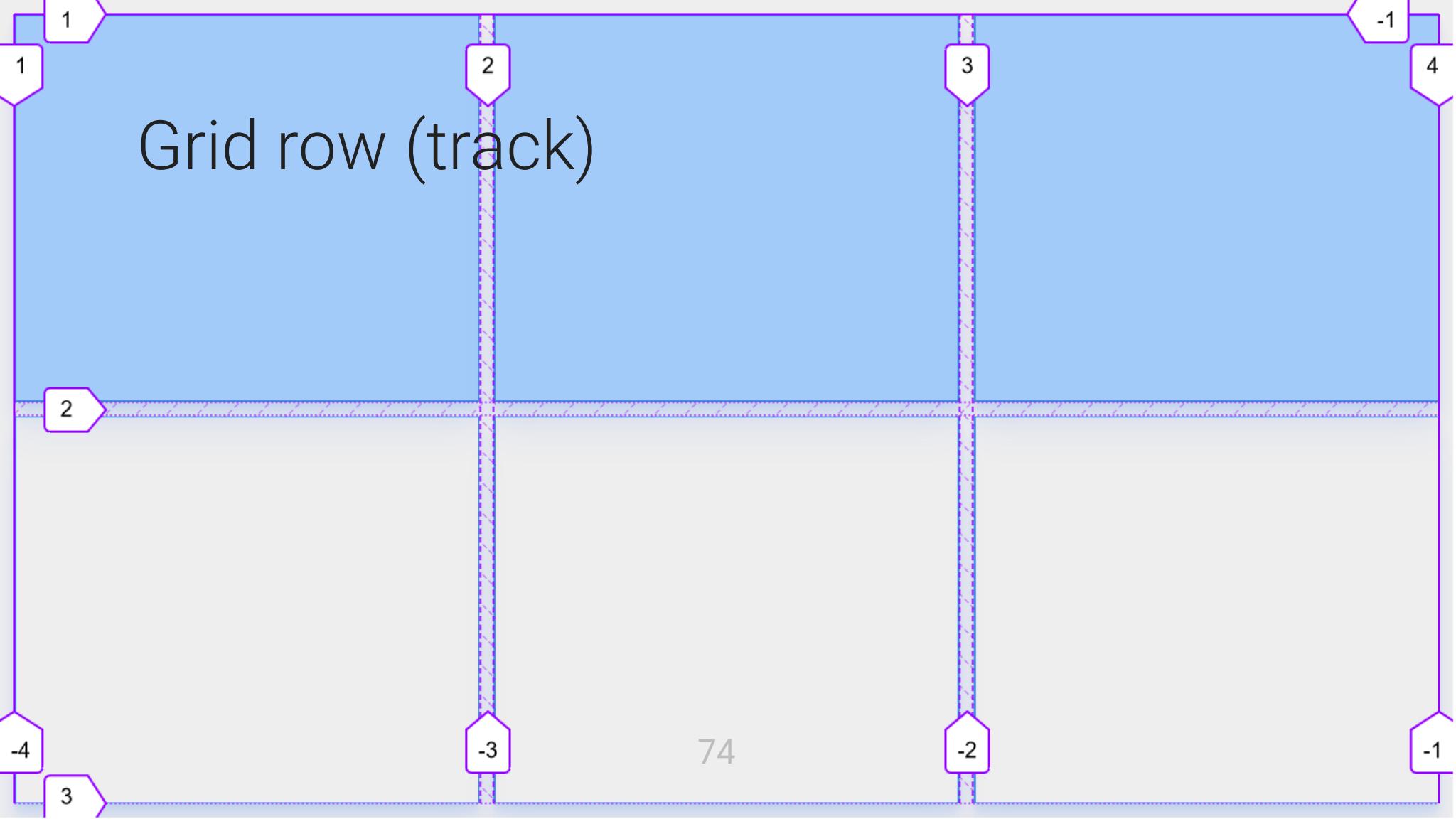
CSS Grid Terminology

- Grid lines
- Grid tracks
- Grid areas
- Grid cell
- Explicit grid
- Implicit grid

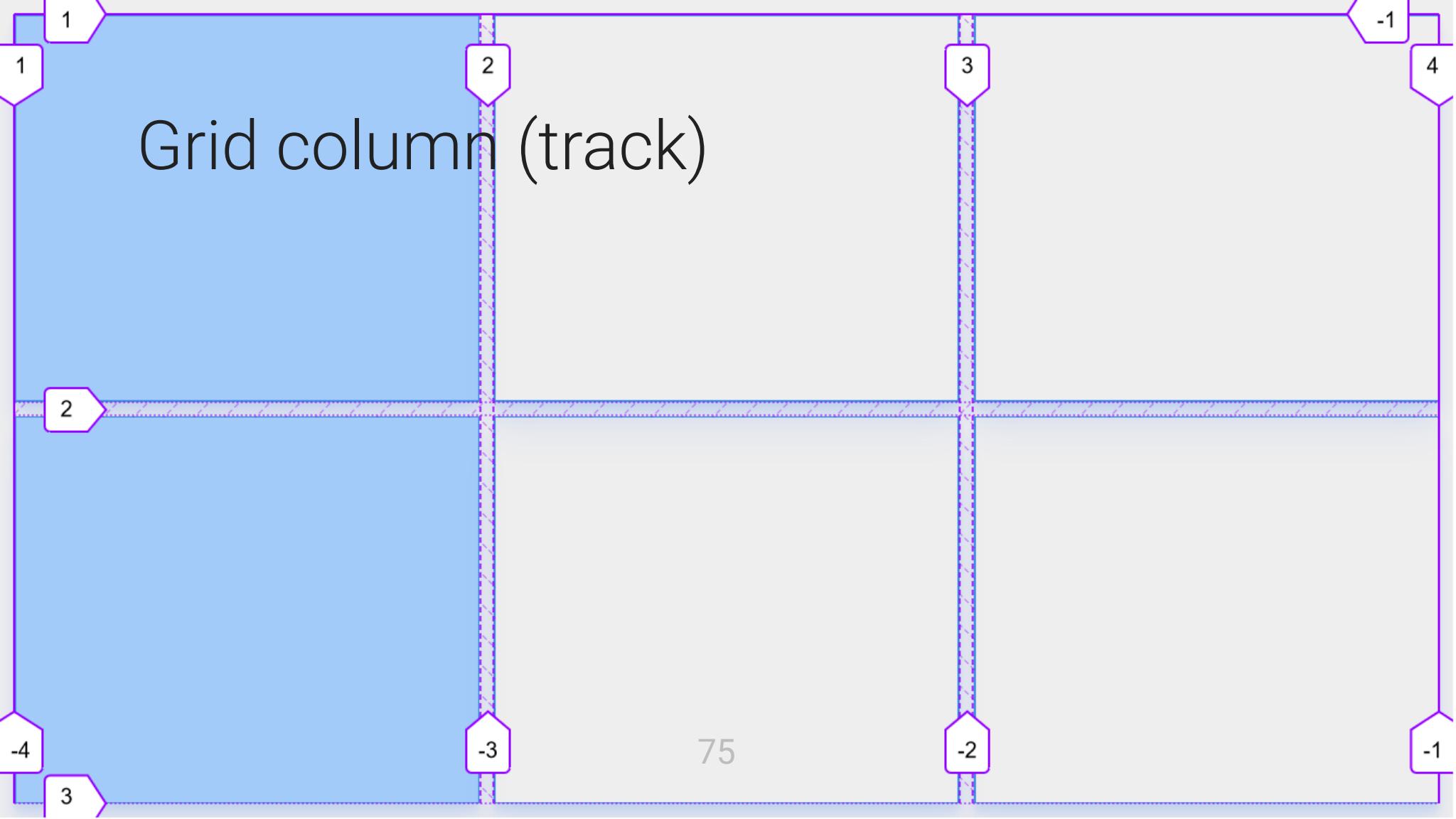
Grid lines



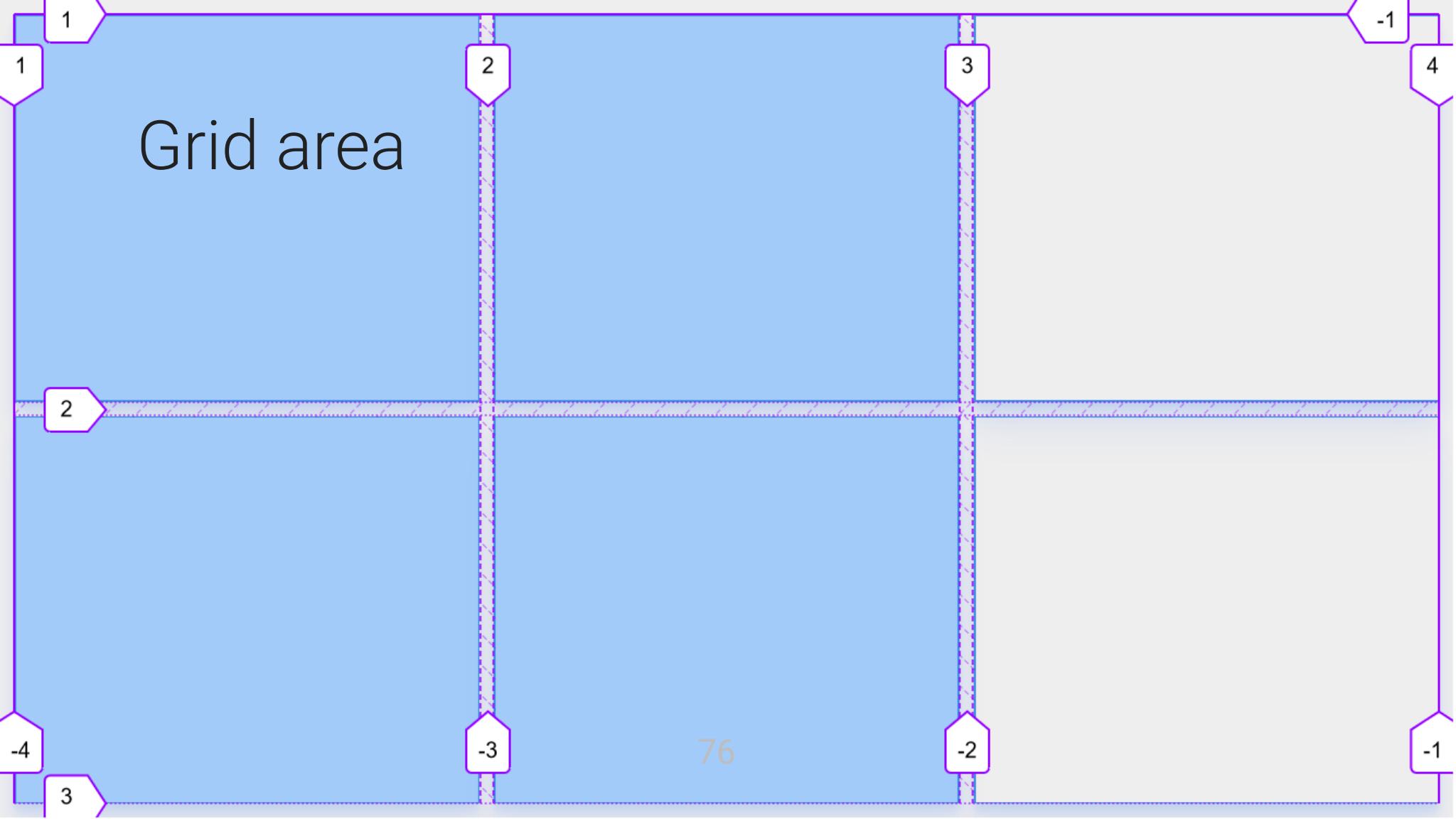
Grid row (track)



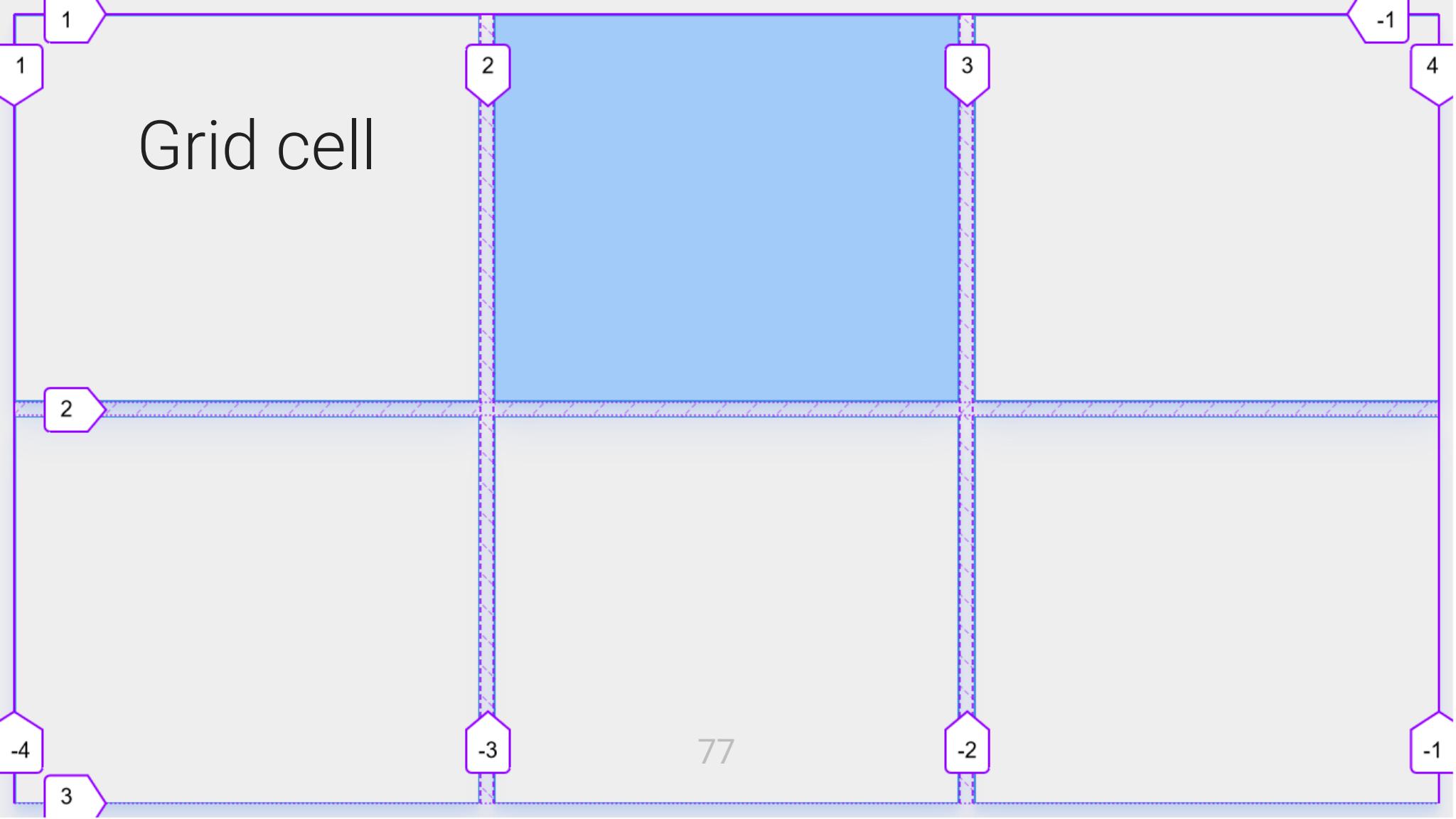
Grid column (track)



Grid area



Grid cell



initial state

```
01. <div class="container">  
02.     <div class="item">First</div>  
03.     <div class="item">Second</div>  
04.     <div class="item">Third</div>  
05. </div>
```

initial state

First

Second

Third

display: grid

```
01. .container {  
02.     display: grid;  
03.     gap: 0.5rem; /* row-gap column-gap */  
04. }
```

display: grid

First

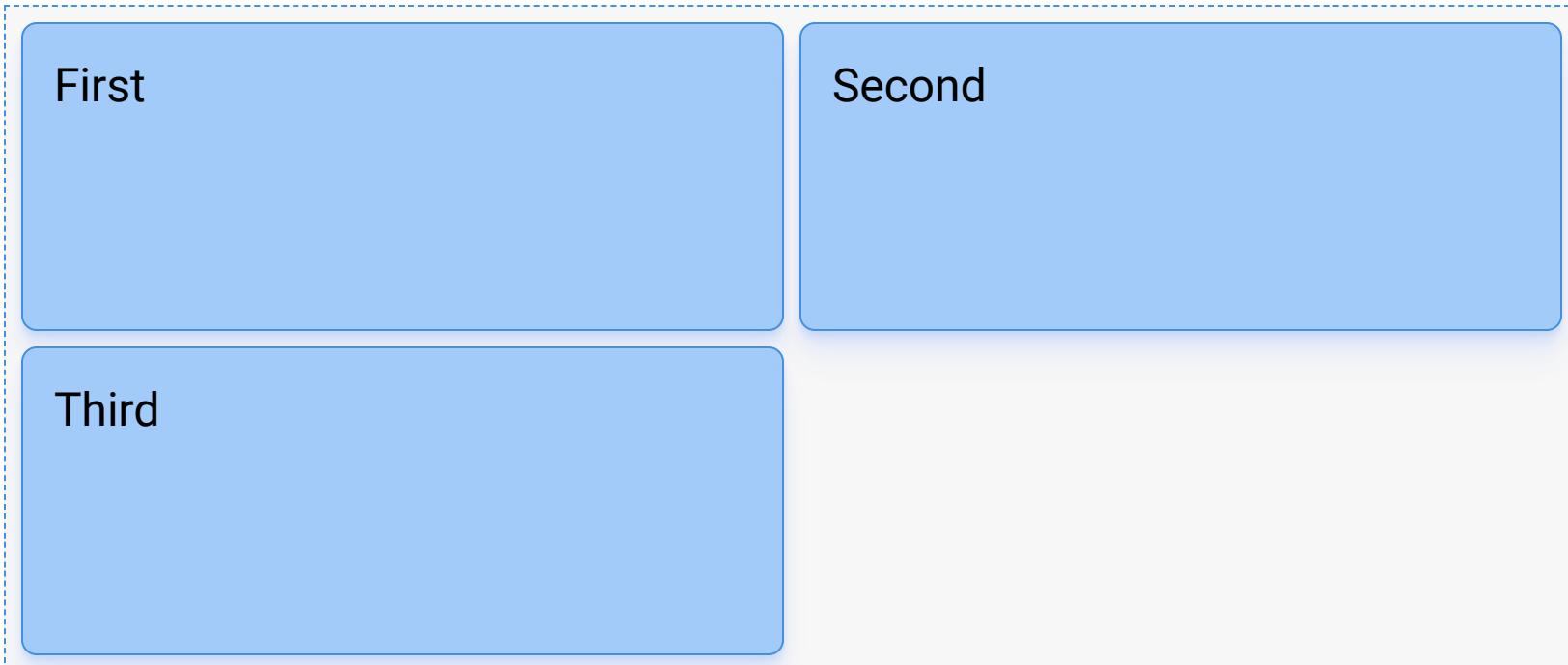
Second

Third

grid-template-columns: 1fr 1fr

```
01. .container {  
02.     display: grid;  
03.     grid-template-columns: 1fr 1fr;  
04. }
```

grid-template-columns: 1fr 1fr



grid-template-rows: 1fr 1.5fr 2fr

```
01. .container {  
02.     display: grid;  
03.     grid-template-rows: 1fr 1.5fr 2fr;  
04. }
```

grid-template-rows: 1fr 1.5fr 2fr

First

Second

Third

```
01. .container {  
02.     display: grid;  
03.     grid-template-rows: 1fr 1.5fr;  
04.     grid-template-columns: 1fr 2fr 1fr;  
05. }
```

First

Second

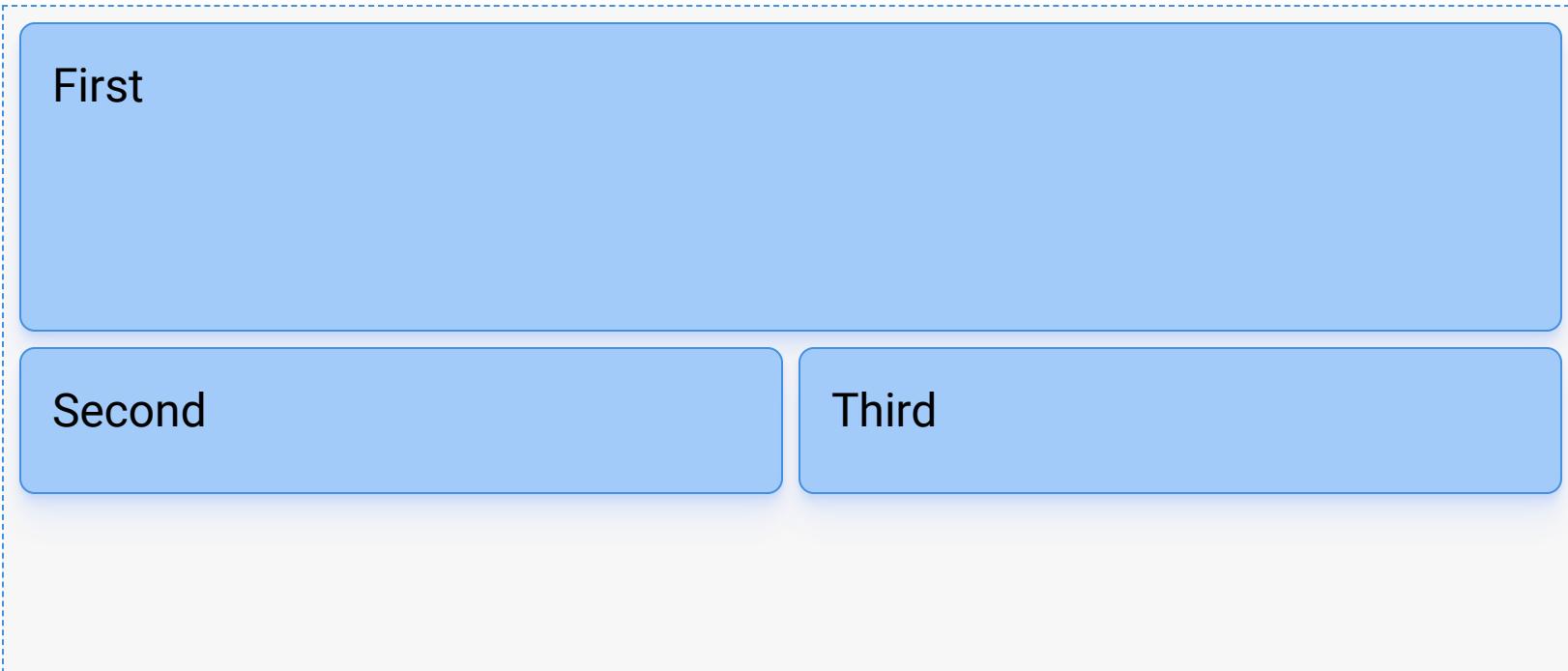
Third

Fourth

grid-rows / grid-columns

```
01. .container {  
02.     display: grid;  
03.     grid-template-rows: repeat(4, 1fr);  
04.     grid-template-columns: 1fr 1fr;  
05. }  
06. .item:nth-child(1) {  
07.     grid-row: 1 / 3; /* grid-row-start / grid-row-end */  
08.     grid-column: span 2; /* grid-column-start / grid-column-end */  
09. }
```

grid-rows / grid-columns

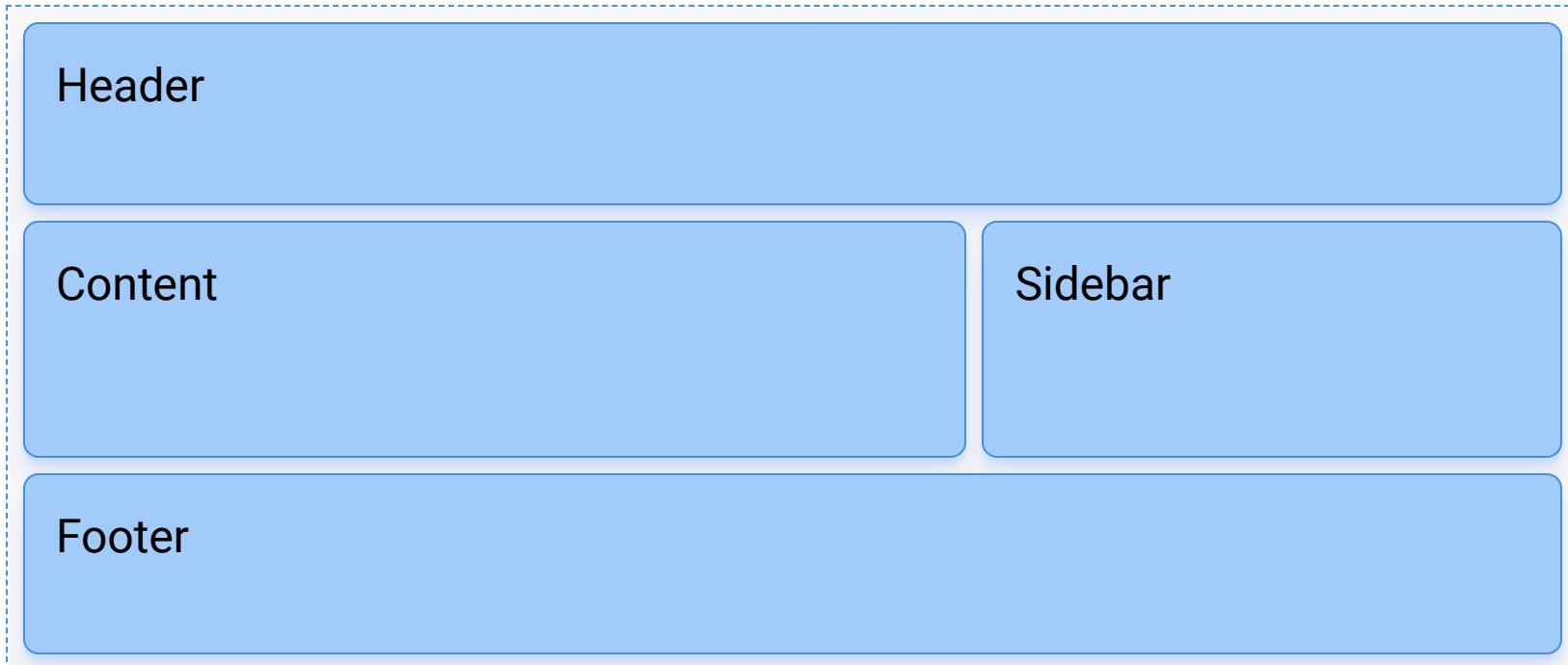


grid-template-area



```
01. .container {          01. .header {  
02.   display: grid;      02.   grid-area: header;  
03.   grid-template-areas:  
04.     "header header header"  
05.     "content content sidebar"  
06.     "content content sidebar"  
07.     "footer footer footer";  
08. }                      03. }  
                           04. .sidebar {  
                           05.   grid-area: sidebar;  
                           06. }  
                           07. .content {  
                           08.   grid-area: content;  
                           09. }
```

grid-template-area



Implicit grid



When grid items are positioned outside of these bounds, the grid container generates implicit grid tracks by adding implicit grid lines to the grid. These lines together with the explicit grid form the implicit grid. The `grid-auto-rows` and `grid-auto-columns` properties size these implicit grid tracks, as well as any explicit grid tracks created by `grid-template-areas` but not explicitly sized by `grid-template-rows` or `grid-template-columns`

[CSS WG](#)

Implicit grid

[Codesandbox 1](#)

[Codesandbox 2](#)

[The Explicit and Implicit grid explained](#)



[Practice] Grid

👉 **Implement Calendar component with CSS Grid**

[Codesandbox](#)

CSS Grid vs CSS Flexbox

Flexbox layout



CSS Grid layout



Useful links

Games

1. [Flexbox Froggy](#)
2. [Grid Garden](#)
3. [Guess-css.app](#)

Articles

1. [Grid by Example by Rachel Andrew](#)
2. [Understanding CSS Grid Container](#)
3. [Masonry with CSS by Tobias Ahlin](#)
4. [Blog by Ahmad Shadeed](#)
5. [Grid bugs](#)

CSS Modules



```
body { ... }
```

```
body div.content div.container { ... }
```

```
body div.content div.container div.articles { ... }
```

```
body div.content div.container div.articles > div.post { ... }
```

```
body div.content div.container div.articles > div.post div.title
```

```
body div.content div.container div.articles > div.post div.title
```

```
body div.content div.container div.articles > div.post div.content
```



Existing solutions

👉 Naming conventions & frameworks

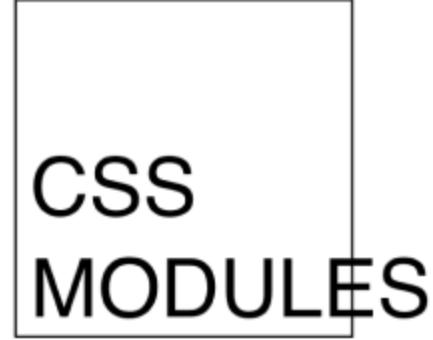
- BEM
- Atomic CSS
- Block_Element-Modifier
- .bg-gray-100.py-6.flex

👉 W3C CSS Scoping Module ([Shadow DOM](#))

👉 CSS-in-JS

👉 ... and many more

CSS Modules & CSS-in-JS



Emotion

CSS Modules

... are local by default 

```
01. .Block { background-color: plum; } // declare css  
02. import styles from "./styles.module.css" // import a css module  
03. function Block() { // apply the class name  
04.   return <div class={styles.Block}>Component</div>  
05. }  
06. <div class="Block-9gbnd5">Component</div> // result
```

102

CSS Modules

- 👍 Solution to global scoping
- 👍 "Native" CSS
- 👍 Server-side rendering

- 👎 Building step is required (and some tweaking may be needed)

Styled Components

```
01. import styled from "styled-components"  
02. let Block = styled.div`  
03.   background-color: # plum;  
04. `.  
05. function Component() {  
06.   return <Block>Component</Block>  
07. }  
08. <div class="Block-9gbnd5">Component</div> // result
```

... all might of JS 💪

```
01. import styled, { css } from "styled-components"  
02. let Button = styled.button`  
03.   background: ■ black;  
04.   color: □ white;  
05.   ${props => props.primary && css`  
06.     background: □ white;  
07.     color: ■ black;  
08.   `}
```

Styled Components

- 👍 Solution to global scoping
- 👍 Variables, functions, all might of JS
- 👍 Easy to make Critical Path CSS
- 👍 Server-side rendering
- 👍 Compilation step is optional
- 🤔 If you skip the compilation step, the css will be generated right at runtime.

PostCSS

PostCSS is a tool for transforming CSS



- 👉 Polyfills
- 👉 Many CSS tools are built on top of it
- 👉 Code Minification
- 👉 Has a huge ecosystem of plugins
- 👉 Linting
- 👉 Mixins
- 👉 Code isolation (scoping)

[A searchable catalog of PostCSS plugins](#)

Tools built with PostCSS



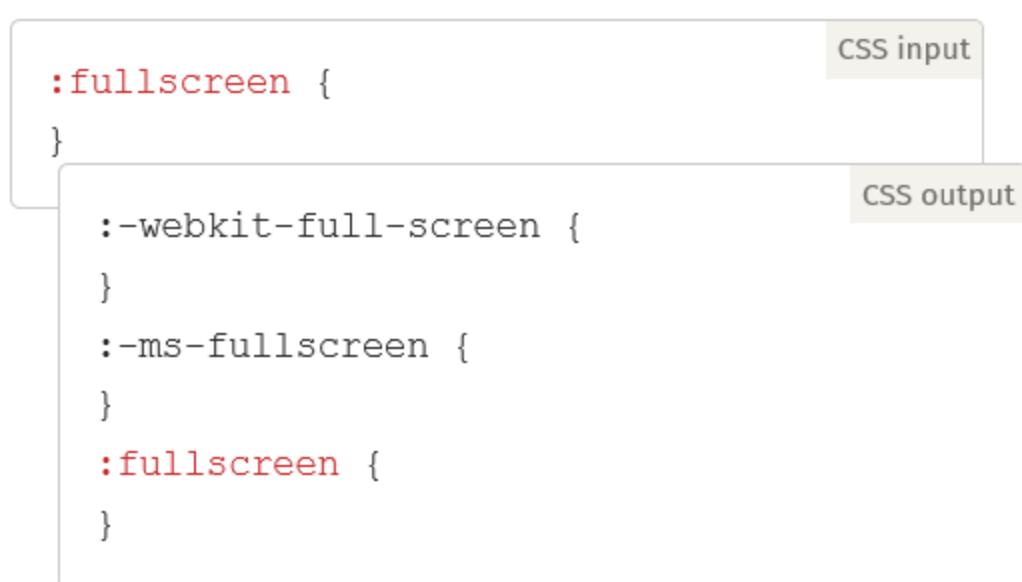
 tailwindcss



... and many more!

Autoprefixer

Adds vendor prefixes
to CSS rules using
values from [Can I Use](#)



Stylelint

A mighty, modern linter that helps you avoid errors and enforce conventions in your styles.

```
stylelint-test|master ✚ ➔ npm run stylelint
> stylelint-test@1.0.0 stylelint /Users/bj0/stylelint-test
> stylelint '**/*.scss'

failing-test-cases.scss
25:10  ▲ Unexpected named color "green"
32:3   ▲ Unexpected at-rule "debug"
73:6   ▲ Unexpected empty block
78:10  ▲ Expected "#ff22ee" to be "#f2e"
83:10  ▲ Expected "#F00" to be "#f00"
257:18 ▲ Expected single space before "{"
261:20 ▲ Expected single space before "{"
267:19 ▲ Expected single space before ","
268:19 ▲ Expected single space before ","
269:19 ▲ Expected single space before "{"
314:1   ▲ Unexpected vendor-prefixed at-rule
         "@webkit-keyframes"
```

Taking CSS Linting to the Next Level with Stylelint

TailwindCSS



```
<figure class="bg-gray-100 rounded-xl p-8">
  
  <div class="pt-6 space-y-4">
    <blockquote>
      <p class="text-lg">
        "Tailwind CSS is the only framework that I've seen scale
        on large teams. It's easy to customize, adapts to any design,
        and the build size is tiny."
      </p>
    </blockquote>
    <figcaption>
      <div>
        Sarah Dayan
      </div>
      <div>
        Staff Engineer, Algolia
      </div>
    </figcaption>
  </div>
</figure>
```

PostCSS Preset Env

Lets you convert modern CSS into something most browsers can understand, determining the polyfills you need based on your targeted browsers or runtime environments, using cssdb

- Enables polyfilling
- Extends CSS syntax

Extra stuff

Extra stuff

Can I use ...?

The End