# Week 3

## Web Design 2

**11 Oct 2023**

# Recap

## Interactive JavaScript

- DOM Tree
- Selecting DOM elements with JS
- Manipulating DOM
- Updating styles
    - Adding/removing/toggling classes
    - Updating content and HTML with `innerText` and `innerHTML`
- Callback functions
- JavaScript Events & Event Listeners

# Designing Interfaces

# Intro to UI Design

- User Interface Design is the process of creating interfaces for software, or other computerised devices.
- GUI is the most common form of interfaces we interact with regularly
- Some other forms include
    - Voice-controlled interfaces (example: Siri/Alexa)
    - Gesture based interfaces (example: Virtual Reality)

# UI Design is only a subset of UX Design

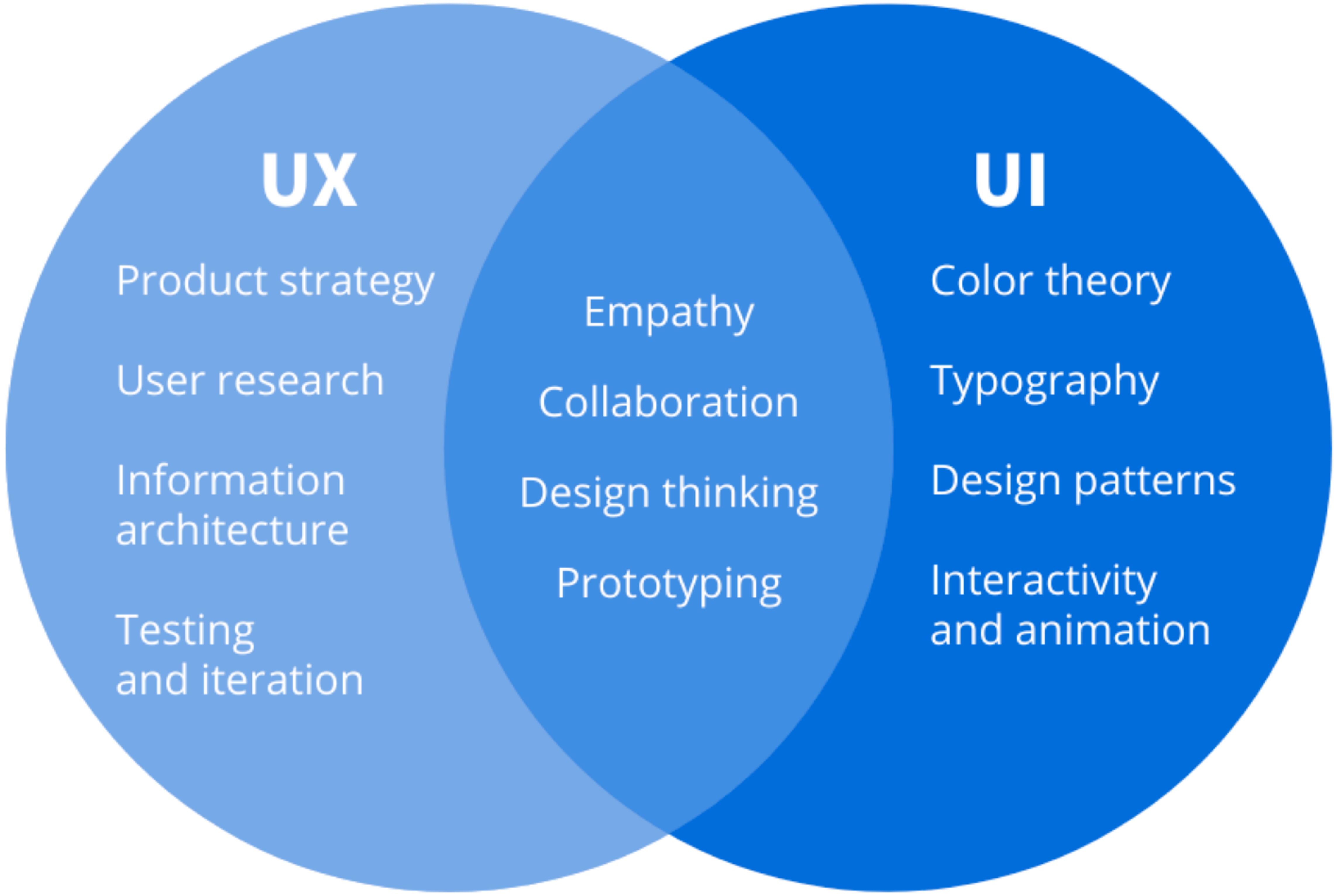


Image from https://www.coursera.org/articles/ui-vs-ux-design

# Designing Interfaces vs Landing Pages

## Interfaces

- Complex in structure
- Require a clear navigation
- Userflows and interaction

## Landing Pages

- Typically have a focused purpose
- Simpler structure
- Involve a Call-to-Action

# GUI for the Web

**Key things to note:**

- Device diversity
- Navigation
- Typography
- Color
- Accessibility
- Performance

# Device Diversity

- Designing for different screens, sizes, resolutions and devices
- Websites today can be accessed from mobile devices, tablets, laptops, 4k monitors and TVs
- Accounting for different input methods (e.g. touch, mouse, keyboard)

# Navigation

- Ensuring the navigation across the website is intuitive and easy to understand
- Clear and concise labelling for items
- Consistency through navigation
- Also require a clear exit (e.g. logging out)

# Typography

- Choose appropriate font sizes and styles for readability
- Considering line height, letter spacing, and font-weight
- Consistency font-families throughout the website or application

# Colour

- Choose colours that are appropriate for the brand and provide contrast for readability
- Avoid using colour as the sole means of conveying information
- Ensure that colour choices are accessible to users with colour blindness or other visual impairments

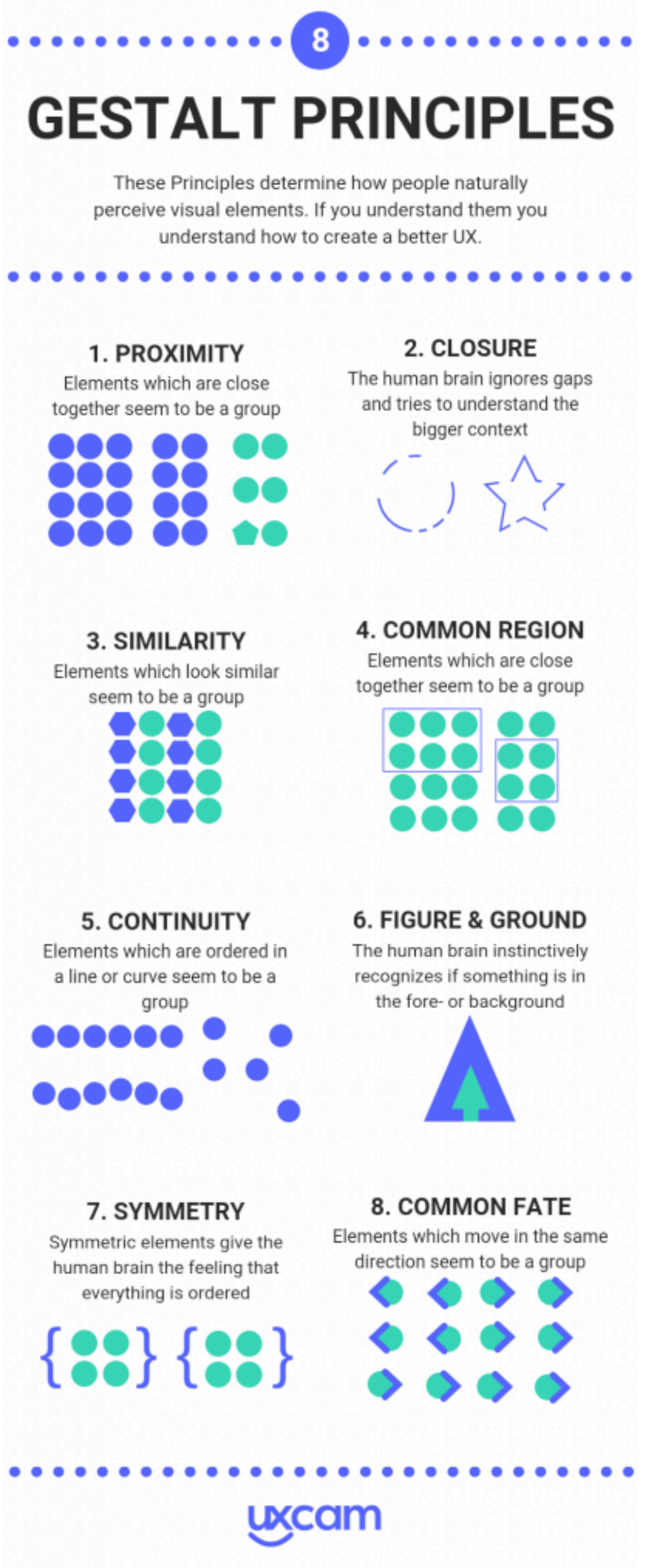
# Accessibility

- Designing for users with disabilities, including those who use assistive technologies
- Ensuring that the interface is keyboard-accessible
- Accessibility in colour contrasts and providing alternative text for images

# Performance

- Optimizing the size and loading times of images and other assets
- Accounting for different internet speeds
- Optimising CSS and JS code for performance

# Revisiting Gestalt Principles of Visual Design



8

## GESTALT PRINCIPLES

These Principles determine how people naturally perceive visual elements. If you understand them you understand how to create a better UX.

**1. PROXIMITY**
Elements which are close together seem to be a group

**2. CLOSURE**
The human brain ignores gaps and tries to understand the bigger context

**3. SIMILARITY**
Elements which look similar seem to be a group

**4. COMMON REGION**
Elements which are close together seem to be a group

**5. CONTINUITY**
Elements which are ordered in a line or curve seem to be a group

**6. FIGURE & GROUND**
The human brain instinctively recognizes if something is in the fore- or background

**7. SYMMETRY**
Symmetric elements give the human brain the feeling that everything is ordered

**8. COMMON FATE**
Elements which move in the same direction seem to be a group

uxcam

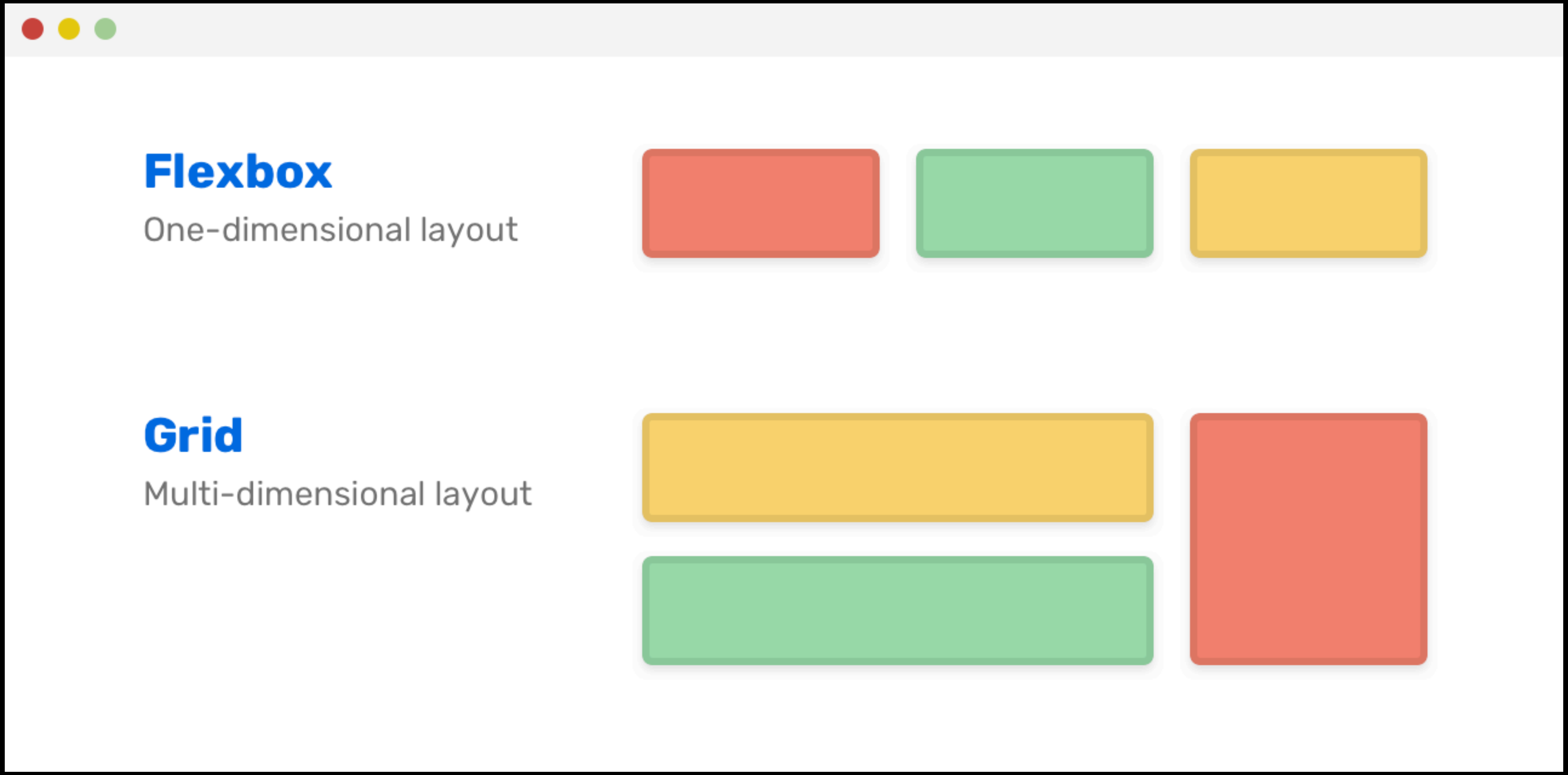
Image from https://uxcam.com/blog/8-gestalt-principles-design-infographic/

# Advanced CSS

# Advanced CSS

- CSS Layout
- CSS Transitions
- CSS Animations
- CSS Preprocessors
- CSS Libraries

# CSS Layouts



**Flexbox**

One-dimensional layout

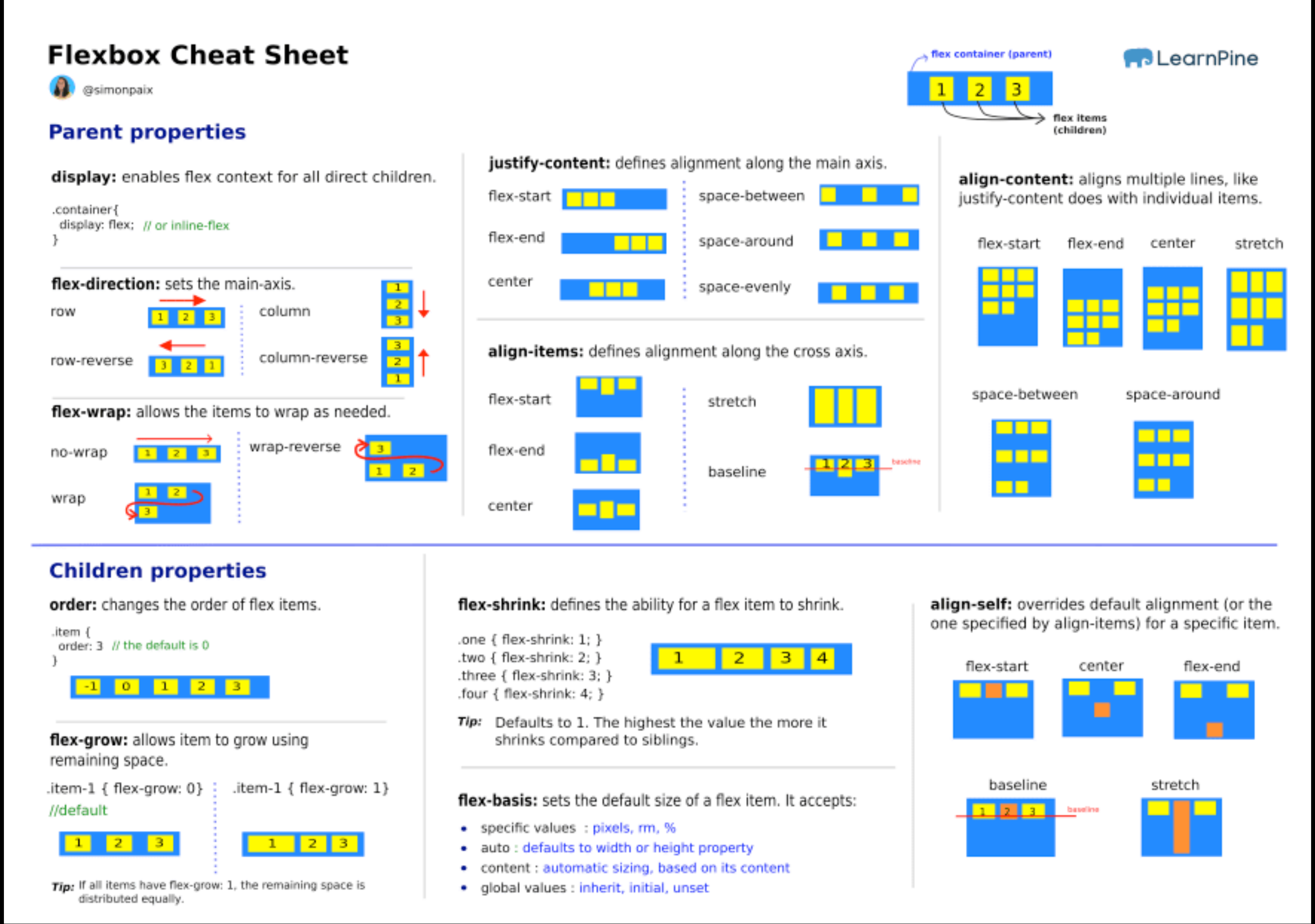
**Grid**

Multi-dimensional layout

Image from https://css-tricks.com/grid-for-layout-flexbox-for-components/

# Flex

https://dev.to/simonpaix/css-flexbox-cheat-sheet-1g09



## Flexbox Cheat Sheet

@simonpaix

LearnPine

### Parent properties

**display:** enables flex context for all direct children.

```
.container{
  display: flex;  // or inline-flex
}
```

**flex-direction:** sets the main-axis.

row

row-reverse

column

column-reverse

**flex-wrap:** allows the items to wrap as needed.

no-wrap

wrap

wrap-reverse

**justify-content:** defines alignment along the main axis.

flex-start

flex-end

center

space-between

space-around

space-evenly

**align-items:** defines alignment along the cross axis.

flex-start

flex-end

center

stretch

baseline

**align-content:** aligns multiple lines, like justify-content does with individual items.

flex-start

flex-end

center

stretch

space-between

space-around

### Children properties

**order:** changes the order of flex items.

```
.item {
  order: 3  // the default is 0
}
```

**flex-grow:** allows item to grow using remaining space.

.item-1 { flex-grow: 0}
//default

.item-1 { flex-grow: 1}

*Tip:* If all items have flex-grow: 1, the remaining space is distributed equally.

**flex-shrink:** defines the ability for a flex item to shrink.

```
.one { flex-shrink: 1; }
.two { flex-shrink: 2; }
.three { flex-shrink: 3; }
.four { flex-shrink: 4; }
```

*Tip:* Defaults to 1. The highest the value the more it shrinks compared to siblings.

**flex-basis:** sets the default size of a flex item. It accepts:

- specific values : pixels, rm, %
- auto : defaults to width or height property
- content : automatic sizing, based on its content
- global values : inherit, initial, unset

**align-self:** overrides default alignment (or the one specified by align-items) for a specific item.

flex-start

center

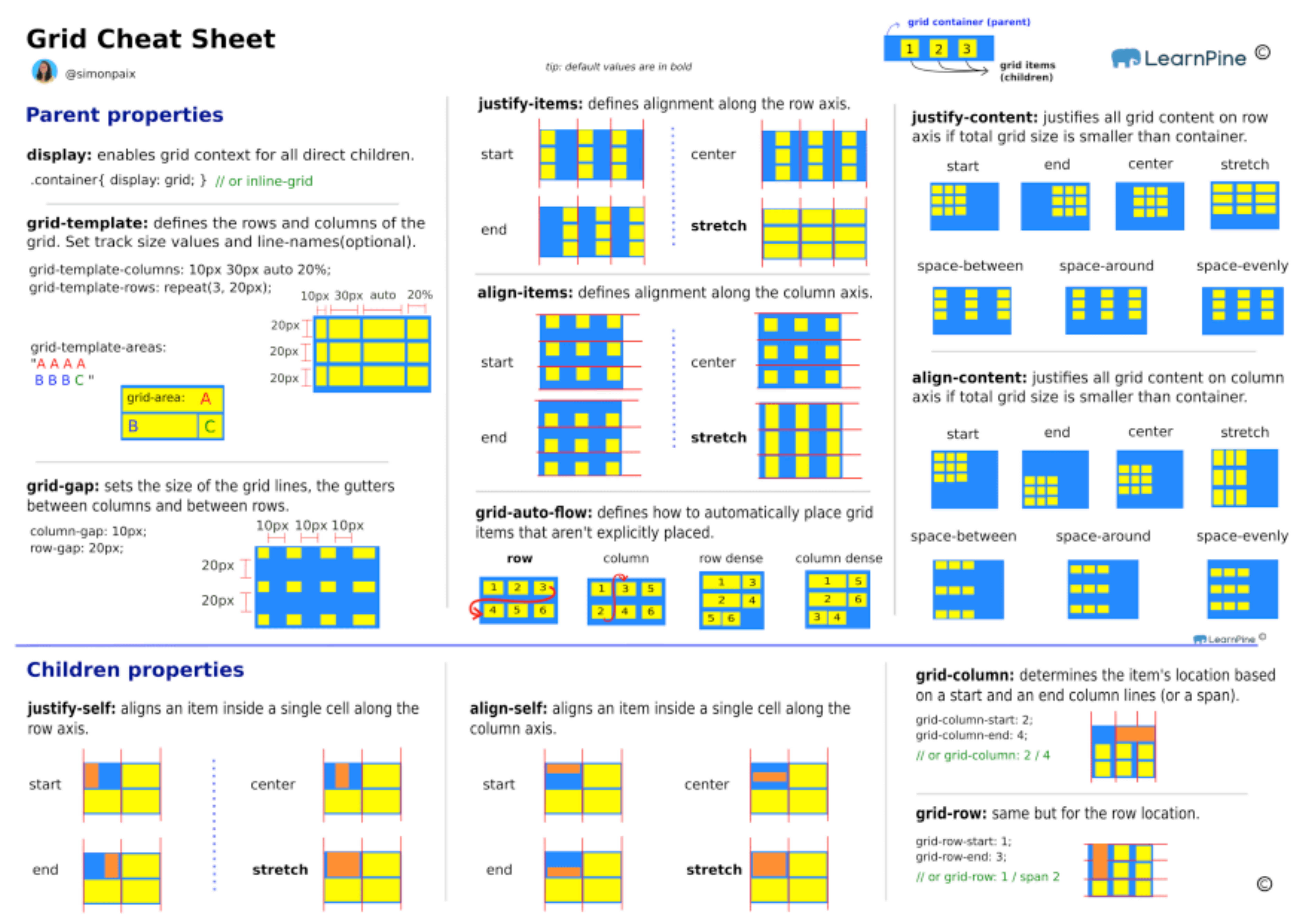
flex-end

baseline

stretch

# Grid

https://dev.to/simonpaix/css-grid-cheat-sheet-2ll8



## Grid Cheat Sheet

@simonpaix

*tip: default values are in bold*

grid container (parent)

1 2 3

grid items (children)

LearnPine ©

### Parent properties

**display:** enables grid context for all direct children.

```
.container{ display: grid; }  // or inline-grid
```

**grid-template:** defines the rows and columns of the grid. Set track size values and line-names(optional).

```
grid-template-columns: 10px 30px auto 20%;
grid-template-rows: repeat(3, 20px);
```

10px 30px auto 20%

20px 20px 20px

```
grid-template-areas:
"A A A A
 B B B C "
```

grid-area: A

B C

**grid-gap:** sets the size of the grid lines, the gutters between columns and between rows.

```
column-gap: 10px;
row-gap: 20px;
```

10px 10px 10px

20px 20px

**justify-items:** defines alignment along the row axis.

start center end **stretch**

**align-items:** defines alignment along the column axis.

start center end **stretch**

**grid-auto-flow:** defines how to automatically place grid items that aren't explicitly placed.

**row** column row dense column dense

**justify-content:** justifies all grid content on row axis if total grid size is smaller than container.

start end center stretch

space-between space-around space-evenly

**align-content:** justifies all grid content on column axis if total grid size is smaller than container.

start end center stretch

space-between space-around space-evenly

LearnPine ©

### Children properties

**justify-self:** aligns an item inside a single cell along the row axis.

start center end **stretch**

**align-self:** aligns an item inside a single cell along the column axis.

start center end **stretch**

**grid-column:** determines the item's location based on a start and an end column lines (or a span).

```
grid-column-start: 2;
grid-column-end: 4;
// or grid-column: 2 / 4
```

**grid-row:** same but for the row location.

```
grid-row-start: 1;
grid-row-end: 3;
// or grid-row: 1 / span 2
```

©

# When to use Flex and Grids

Flex is best suited for

- Arranging items in a row or column
- Responsive list items

Grid is best suited for

- Having more control over the layouts
- Creating asymmetrical 2D layouts

# CSS Transitions

- Transitions are triggered by changes to CSS properties, such as hover effects, changes in class, or JavaScript events
- Transitions are defined using the transition property, which specifies the CSS property to animate, the duration of the animation, and an easing function to control the pace of the animation
- The transition property can be applied to individual elements or to entire groups of elements using CSS selectors
- Transition can help enhance the user experience and provide a more engaging and visually appealing design.

# CSS Animations

- Just like transitions, animations are triggered by changes to CSS properties, such as hover effects, changes in class, or JavaScript events
- Animations are defined using the @keyframes rule, which specifies the styles for each stage of the animation
- Animations can also provide a visual meaning to the UX (e.g. elements sliding away on delete actions)

# Motion as Meaning

- CSS Transitions and Animations can drastically improve the visual experience of a website when used appropriately.
- They can provide visual feedback, and meaning to interactions on the website
- They can also be performance heavy sometimes. It is a good idea to use transitions and animations carefully

# CSS Preprocessors

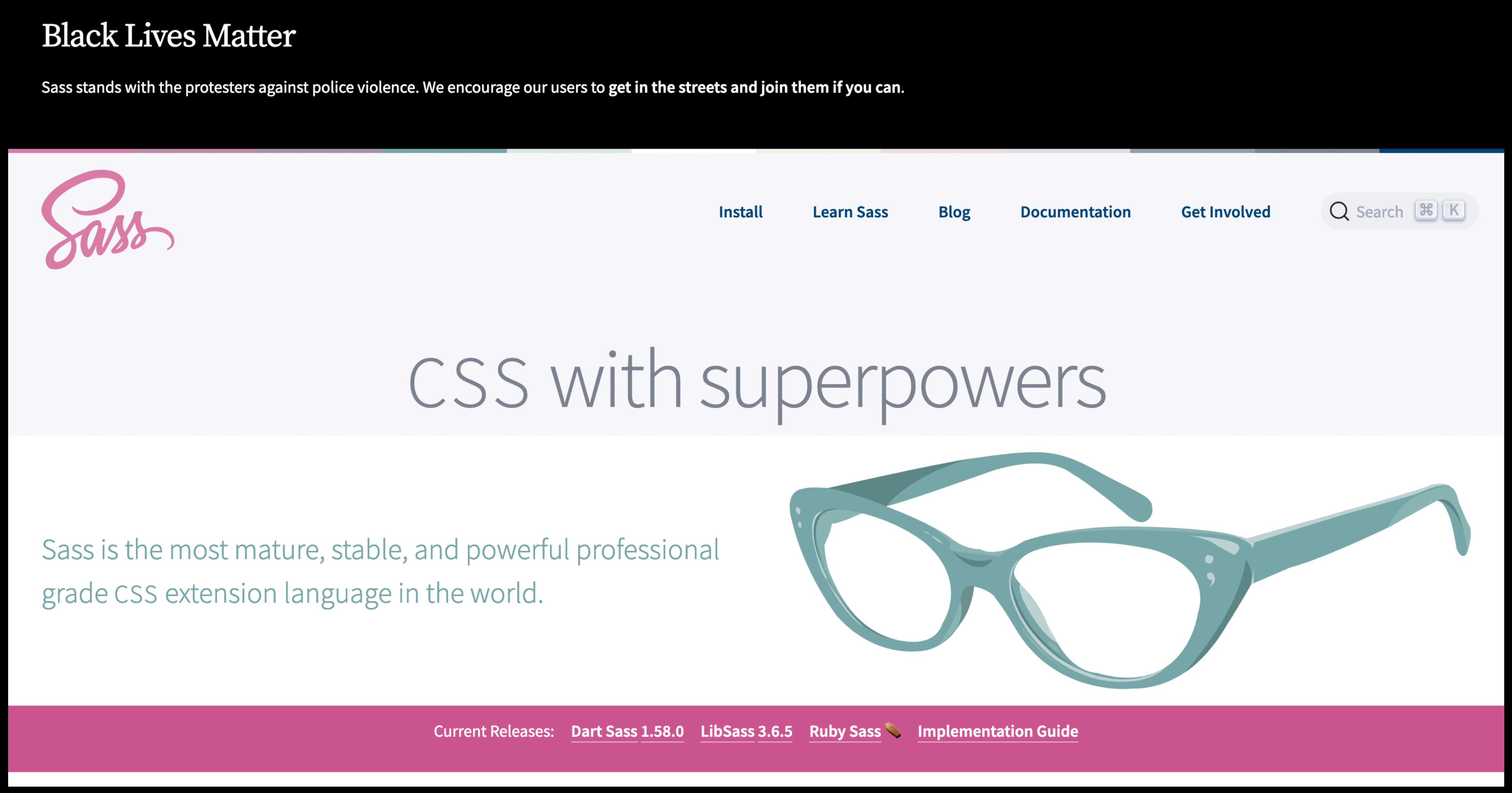
## CSS with superpowers ⚡ ⚡

# What are CSS Preprocessors?

- Preprocessors are built on CSS standards with additional support
- Preprocessors cannot directly work with browsers
- They need to be compiled to generate a CSS file before it can be rendered

# SASS
# Syntactically Awesome Stylesheet

- CSS is one of the popular CSS Preprocessors
- Reduces repetition
- Supports variables, nested rules, imports, functions…



Black Lives Matter

Sass stands with the protesters against police violence. We encourage our users to **get in the streets and join them if you can.**

CSS with superpowers

Sass is the most mature, stable, and powerful professional grade CSS extension language in the world.

Current Releases: Dart Sass 1.58.0 LibSass 3.6.5 Ruby Sass Implementation Guide

https://sass-lang.com/

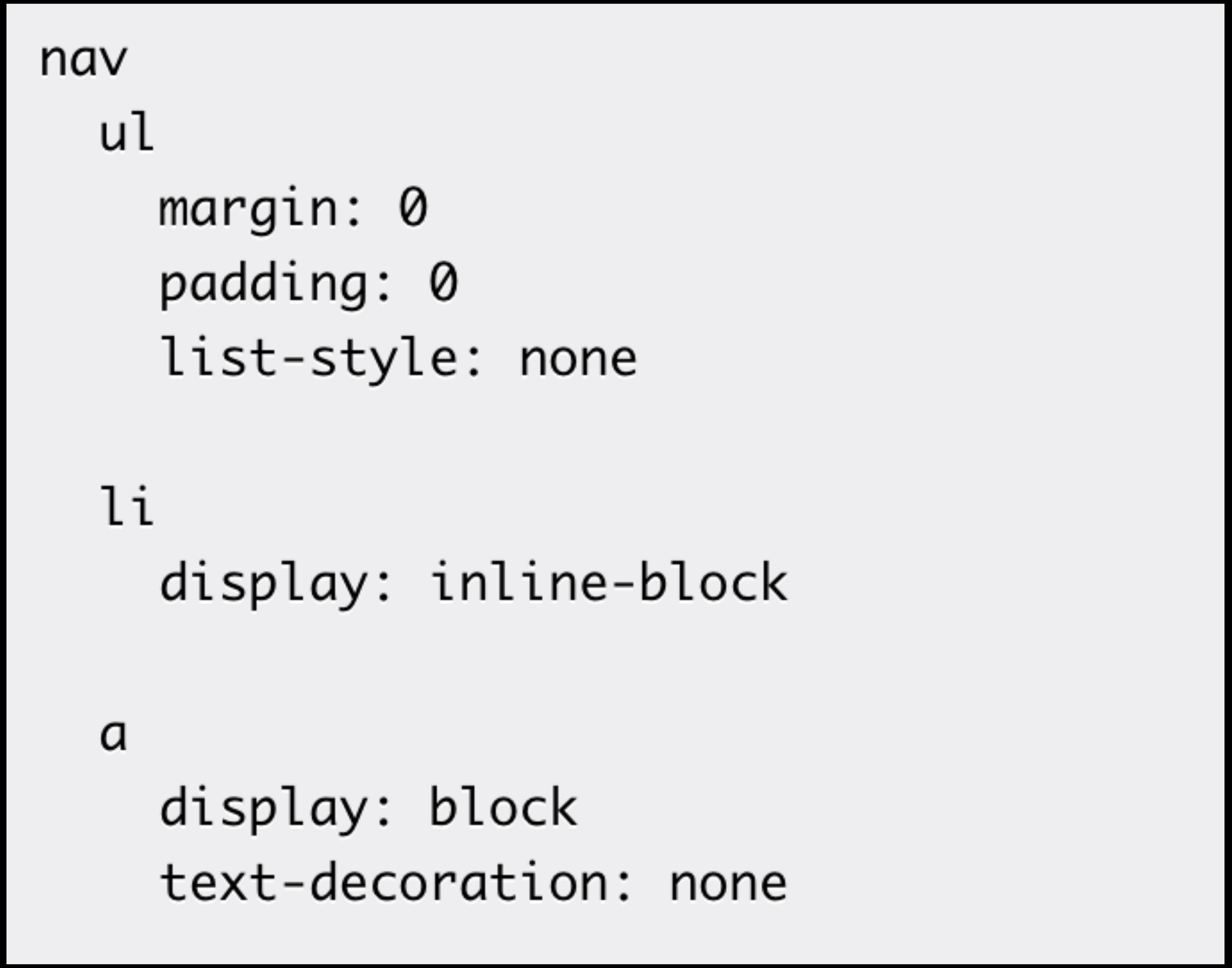
# SASS
# Syntactically Awesome Stylesheet

- CSS is one of the popular CSS Preprocessors
- Reduces repetition
- Supports variables, nested rules, imports, functions…

# SASS Syntax

1. Nesting

Elements inside a parent element can be grouped and styled together

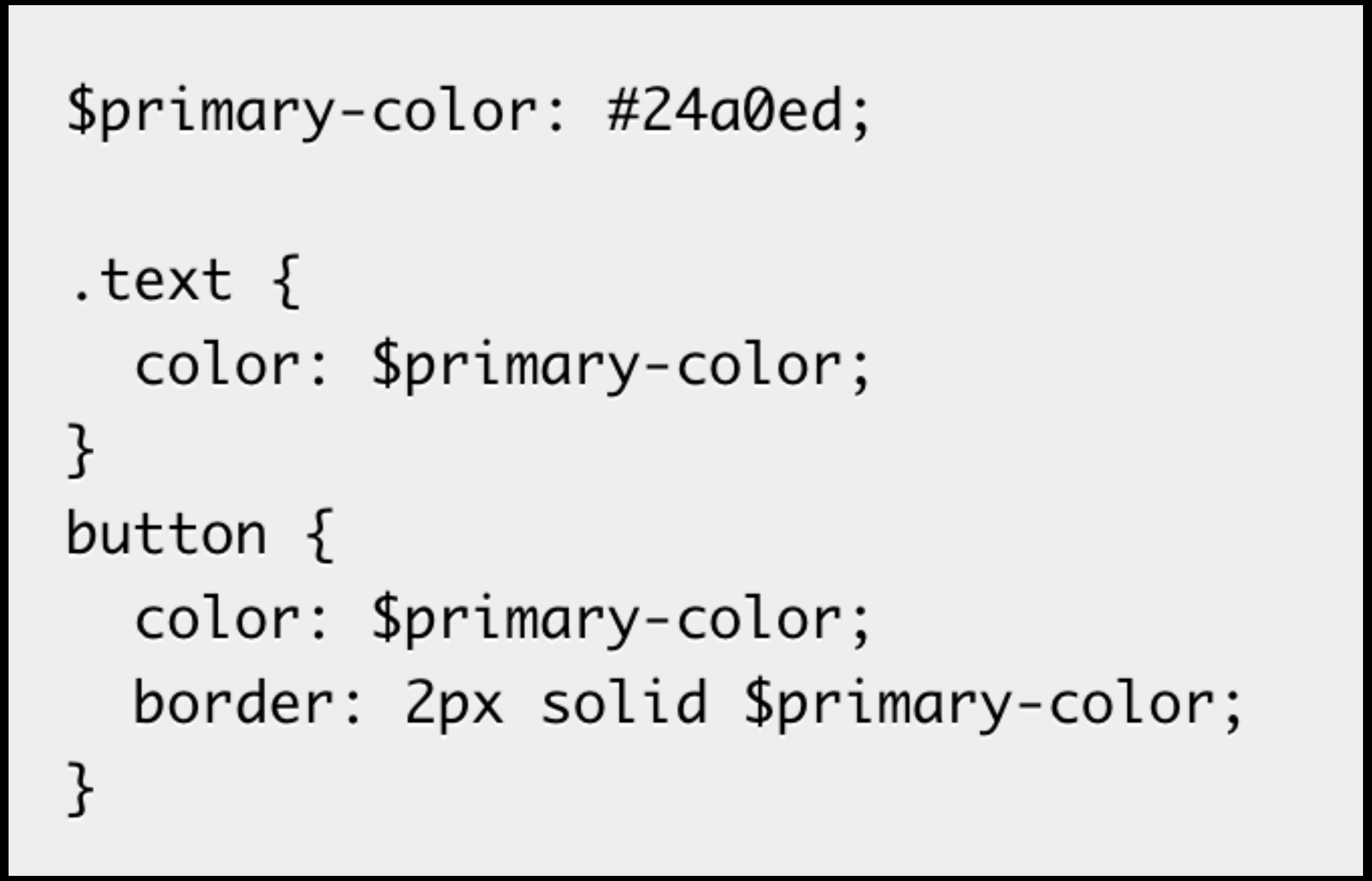


```
nav
  ul
    margin: 0
    padding: 0
    list-style: none

  li
    display: inline-block

  a
    display: block
    text-decoration: none
```

# SASS Syntax

2. Variables

Easy to maintain consistency and developing style systems using variables

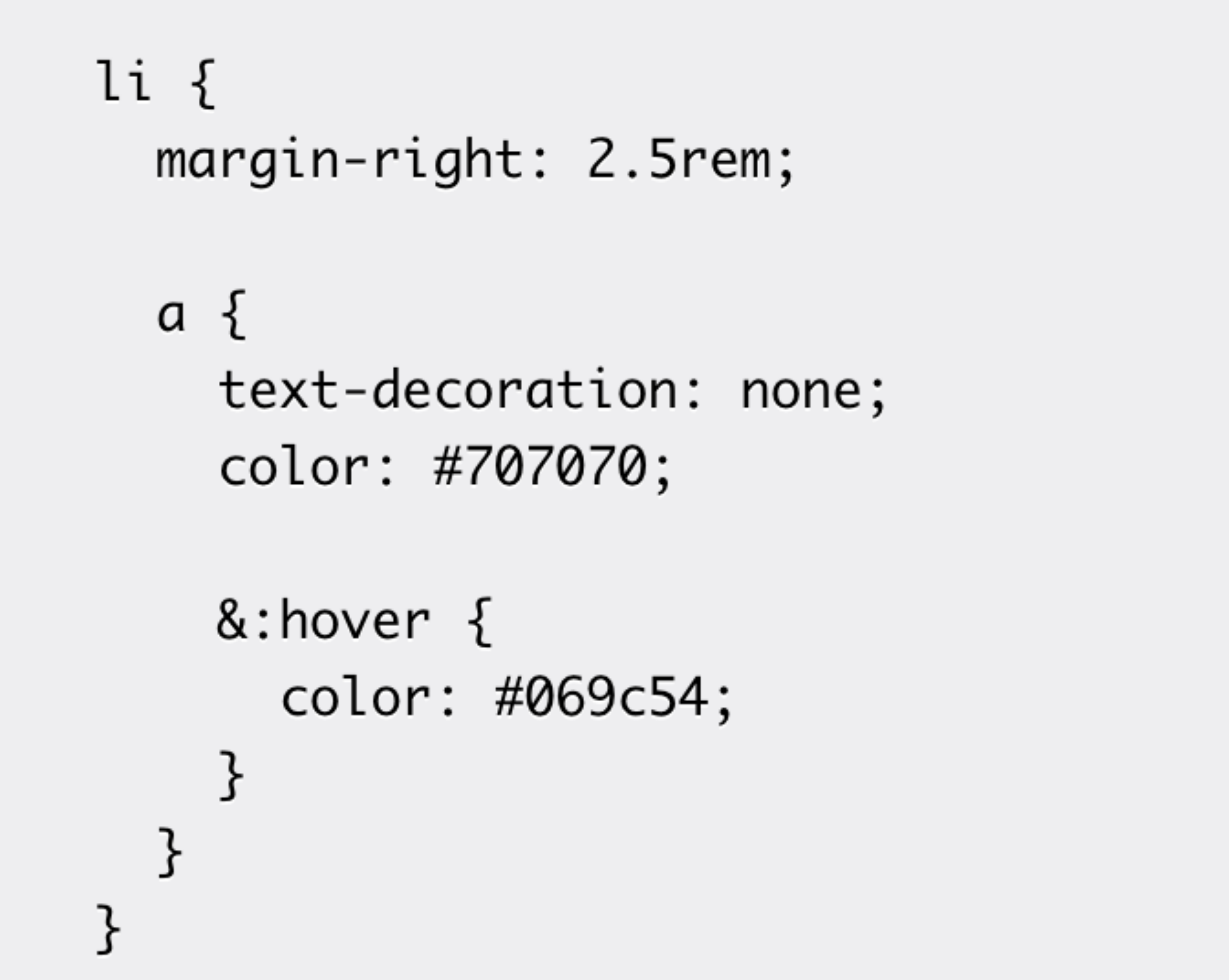


```
$primary-color: #24a0ed;

.text {
  color: $primary-color;
}
button {
  color: $primary-color;
  border: 2px solid $primary-color;
}
```

# SASS Syntax

3. Targeting pseudo-classes

Pseudo-classes like :hover or :active of the parent elements can be selected using &



```
li {
  margin-right: 2.5rem;

  a {
    text-decoration: none;
    color: #707070;

    &:hover {
      color: #069c54;
    }
  }
}
```

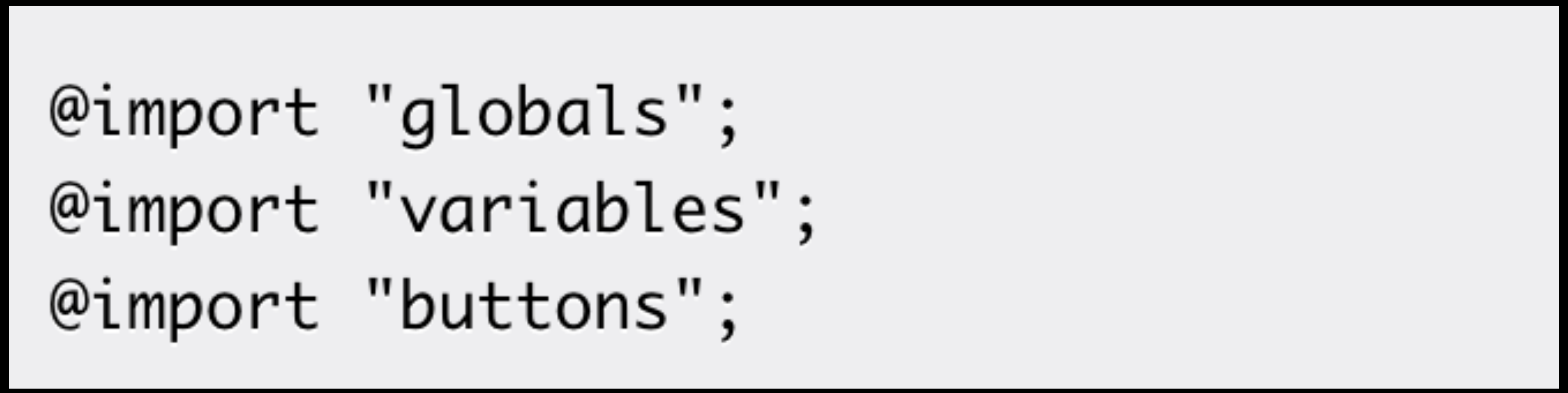
# SASS Syntax

4. @import

Partials are supported in Sass, which makes splitting and organising code easier

You can separate your stylings by pages/features/components into separate files and import them into the main file

Note: file names for partials must be prefixed with an underscore. e.g. _globals.sass; _variables.sass

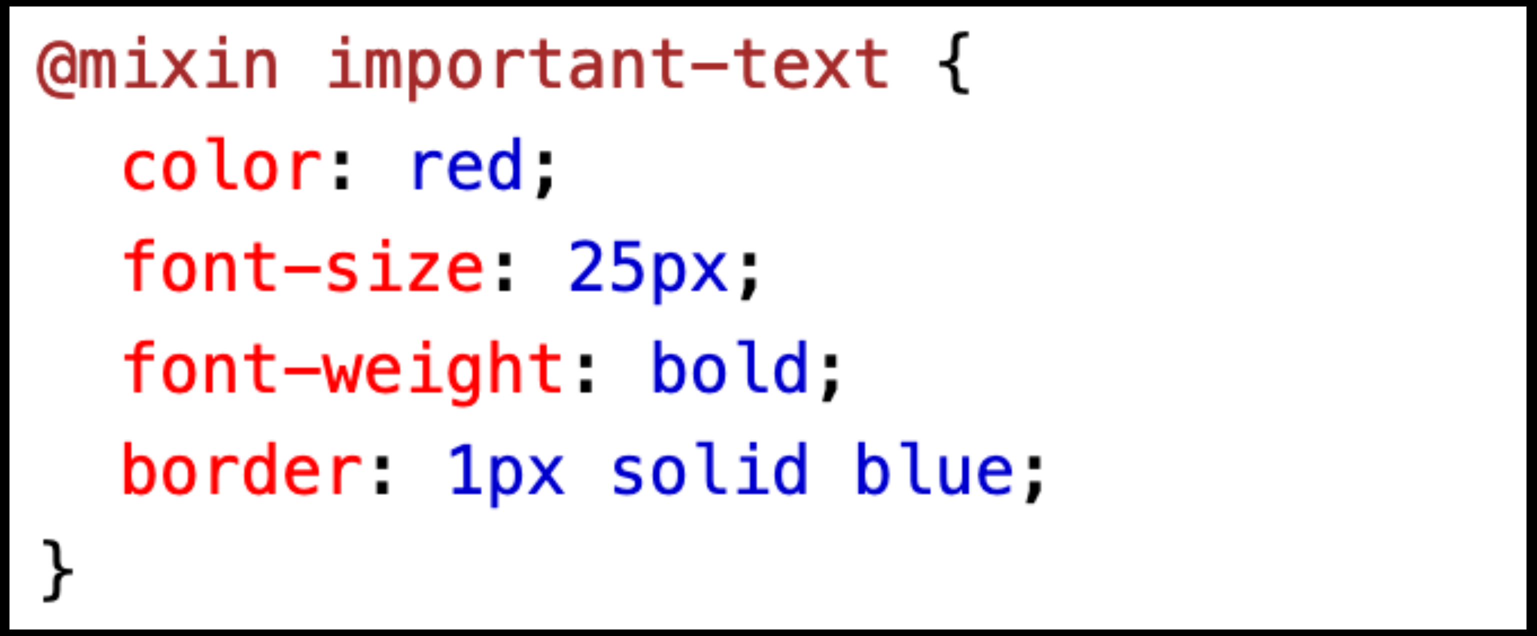


```
@import "globals";
@import "variables";
@import "buttons";
```

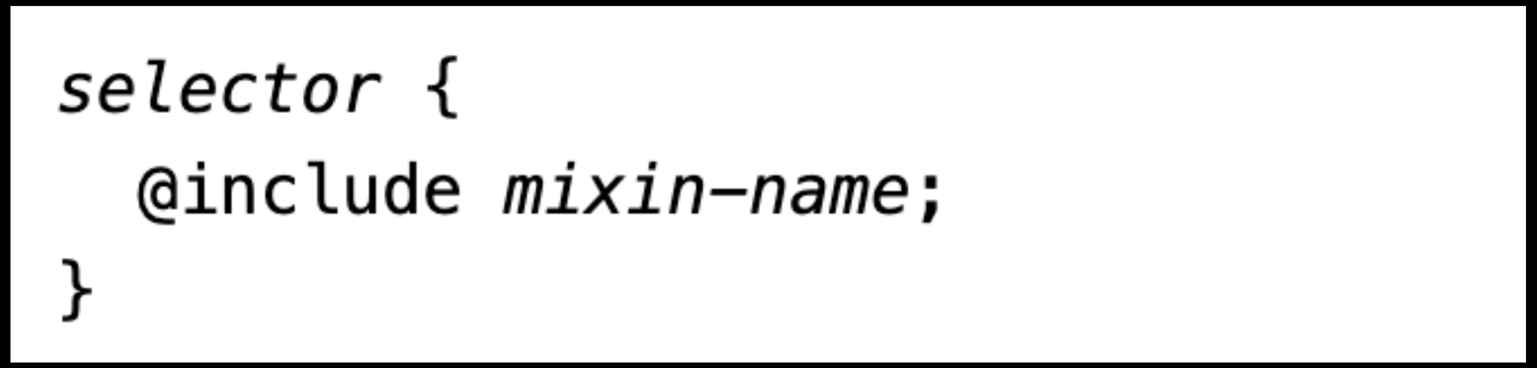
# SASS Syntax

5. @mixins

Reusable styling blocks with mixins can be used with a simple @include



```
@mixin important-text {
  color: red;
  font-size: 25px;
  font-weight: bold;
  border: 1px solid blue;
}
```

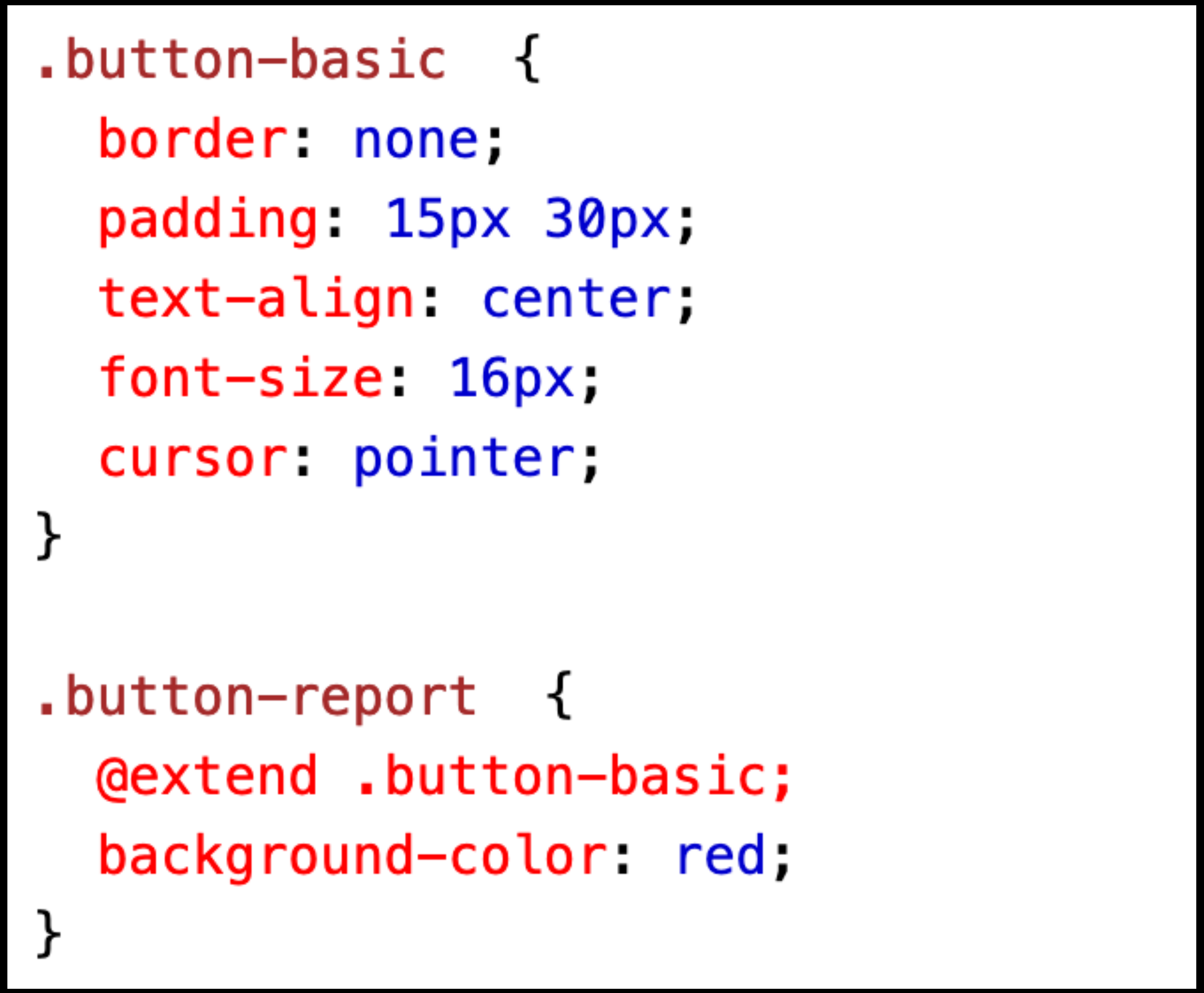


```
selector {
  @include mixin-name;
}
```

# SASS Syntax

6. @extend

@extend directive supports extending styling from one style block to another



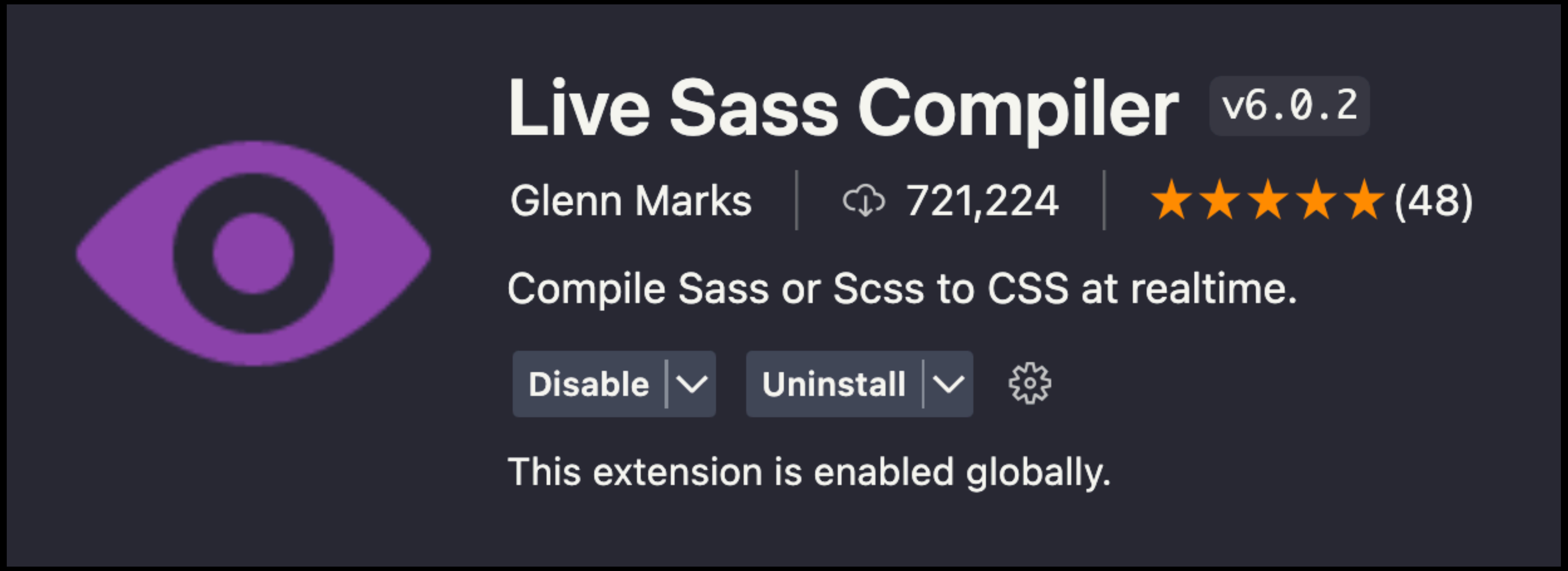
```
.button-basic  {
  border: none;
  padding: 15px 30px;
  text-align: center;
  font-size: 16px;
  cursor: pointer;
}

.button-report  {
  @extend .button-basic;
  background-color: red;
}
```

# Using SASS with HTML

Sass must be compiled before it can run on the browser

VSCode

- Live Sass Compiler



Live Sass Compiler v6.0.2

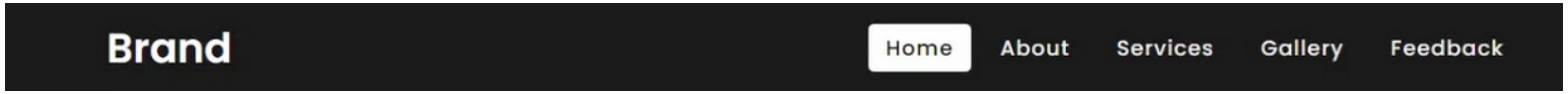
Glenn Marks | 721,224 | ★★★★★ (48)

Compile Sass or Scss to CSS at realtime.

This extension is enabled globally.

# Lab Exercise

- Using Sass
  - 1. Create a navigation menu using Flex (reference below)



  - 2. Create an asymmetrical media gallery using Grid (reference below)