SUDOCORE v5.0.0

Version Title: ***JThai***

Release Date: January 1, 2020

TABLE OF CONTENTS

To find content throughout the document use ctrl + f for Windows or cmd + f for MacOS.

Button Group
Card
Carousel
Collapse
Dropdowns
Forms
Input Group
Jumbotron
List Group
Media Object
Modal
Nav
Navbar
Pagination
Popovers
Progress
Scrollspy
Spinners
Toasts
Tooltips

Section 5. Utilities

Borders
Clearfix
Close Icon
Colors
Display
Embed

JavaScript
Theming
Build Tools
Webpack
Accessibility

01 Getting Started

In this section, we will go over how to implement Sudocore onto your website/app.

USING THE SUDOCORE CDN

Get started with Sudocore, the world's newest framework for building responsive, mobile-first sites, with SudocoreCDN and a template starter page.

Quick Start

Looking to quickly add Sudocore to your project? Use SudocoreCDN, provided for free by our sponsors in SoCal.

CSS

Copy-paste the stylesheet <link> into your <head> before all other stylesheets to load our CSS.

```
<link rel="stylesheet" href="https://sudocore.dev/sudocore/5.0.0/sudocore.min.css">
```

JS

Our dynamic components require the use of JS to function. Specifically, they require jQuery, Popper.js, and our own JS plugins. Place the following <script> tags near the end of your pages, right before the closing </body> tag, to enable them. jQuery must come first, then Popper.js, and then our JS plugins.

```
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoR
SJoZ+n" crossorigin="anonymous"></script>

<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"
integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMf
ooAo" crossorigin="anonymous"></script>

<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"
integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8i
fwB6" crossorigin="anonymous"></script>
```

Starter Template

Be sure you have your pages set up with the latest design and development standards. That means using an HTML5 doctype and including a viewport meta tag for proper responsive behaviors. Put it all together and your pages should look like this:

```
<!doctype html>
```

```
<html lang="en">
    <head>
        <!-- Required meta tags -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1,
        shrink-to-fit=no">

        <!-- SudocoreCSS -->
        <link rel="stylesheet"
        href="https://sudocore.dev/sudocore/5.0.0/sudocore.min.css">

        <title>Hello, world!</title>
    </head>
    <body>
        <h1>Hello, world!</h1>

        <!-- Optional JavaScript -->
        <!-- jQuery first, then Popper.js, then SudocoreJS -->
        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
        integrity="sha384-J6qa4849blE2+poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFA
        V0wwj1yYfoRSJoZ+n" crossorigin="anonymous"></script>
        <script
        src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.j
        s"
        integrity="sha384-Q6E9RHvbIyZFJoft+2mJbHaEWldlvI9IOYy5n3zV9zzTtmI3U
        ksdQRVvoxMfooAo" crossorigin="anonymous"></script>
        <script
        src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.
        js"
        integrity="sha384-wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uo
        d8GCExl3Og8ifwB6" crossorigin="anonymous"></script>
    </body>
</html>
```

Important Globals

Sudocore employs a handful of important global styles and settings that you'll need to be aware of when using it, all of which are almost exclusively geared towards the normalization of cross browser styles. Let's dive in.

HTML5 doctype

Sudcore requires the use of the HTML5 doctype. Without it, you'll see some funky incomplete styling, but including it shouldn't cause any considerable hiccups.

```
<!doctype html>
 <html lang="en">
            ...
</html>
```

Responsive Meta Tag

Sudocore is developed mobile first, a strategy in which we optimize code for mobile devices first and then scale up components as necessary using CSS media queries. To ensure proper rendering and touch zooming for all devices, add the responsive viewport meta tag to your <head>.

```
<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
```

Box-Sizing

For more straightforward sizing in CSS, we switch the global box-sizing value from content-box to border-box. This ensures

padding does not affect the final computed width of an element, but it can cause problems with some third party software like Google Maps and Google Custom Search Engine.

On the rare occasion you need to override it, use something like the following:

```
.selector-for-some-widget {
        box-sizing: content-box;
}
```

With the above snippet, nested elements—including generated content via ::before and ::after—will all inherit the specified box-sizing for that .selector-for-some-widget. Learn more about box model and sizing at CSS Tricks.

Reboot

For improved cross-browser rendering, we use Reboot to correct inconsistencies across browsers and devices while providing slightly more opinionated resets to common HTML elements.

02 The Fundamentals

In this section, we will go over how to manipulate your components to conform to every screen size.

UNDERSTANDING SCREEN SIZES

Sudocore breaks down screen sizes into three basic devices: laptops/desktops, tablets, and smartphones.

Understanding Breakpoints

In order to set different screen widths, Sudocore was designed with 6 breakpoints. Breakpoints are pixel dimensions where the code breaks from one device to the next.

The Six Breakpoints:

- ❖ xs: extra small
- ❖ sm: small
- ❖ md: medium
- ❖ lg: large
- ❖ xl: extra large
- ❖ & default

For the most part, the only three that you should concern yourself with are lg, md, and default.

- ❖ lg refers to most desktop and laptop screen dimensions.
- ❖ md refers to tablets.

- ❖ default will be good enough to define your small screens (smartphones)

An Example:

```
<div class="col-lg-4 col-md-6 col-12"> </div>
```

This means that the div will take the space of 4 columns on a large screen, 6 columns on a medium, and 12 columns on all other screen sizes (small).

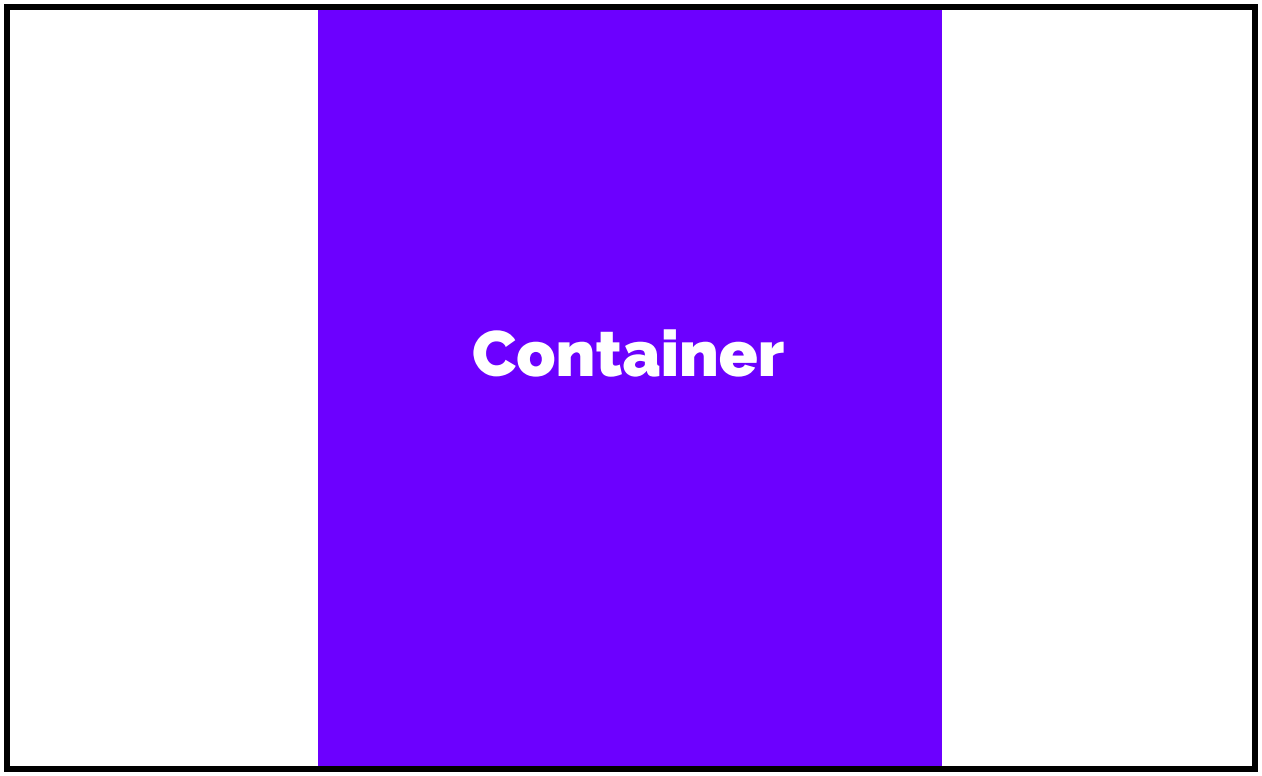
CONTAINERS

Containers are a great way of keeping content in the center of the device. Do not mistake this with centering text or components. Containers constrain the page width preventing it from spanning from edge to edge of the screen.

Purpose

Containers are actually more responsive because they keep elements constrained. Imagine you were looking through a 4k monitor. If elements weren't constrained then they would be thousands of pixels apart.

You typically find this on pages like blogs or news articles.



You can also explicitly choose which screen sizes you would like to implement a container.

These classes can be called:

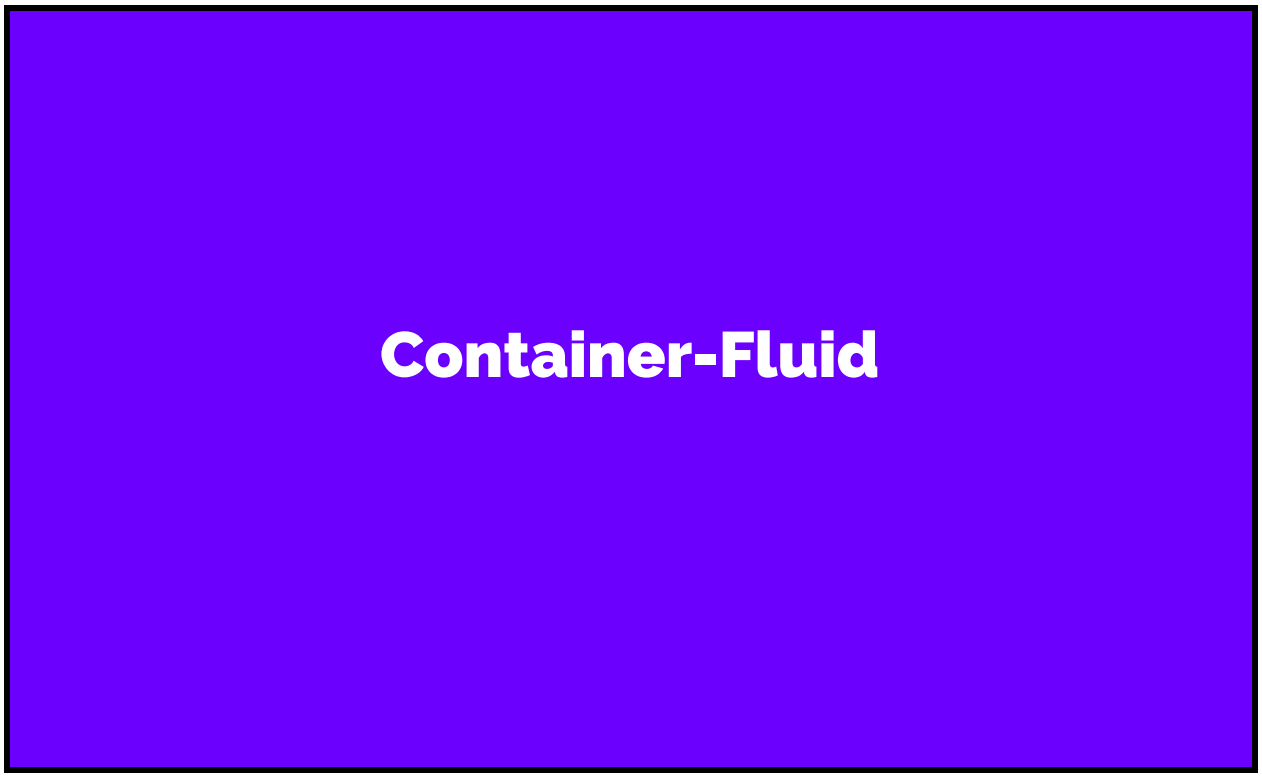
- ❖ container-sm
- ❖ container-md
- ❖ container-lg
- ❖ Container-xl

CONTAINERS-FLUIDS

Container-fluids are a great way of making components span the entire width of the screen. Container-fluids should be used purposefully and deliberately. The last thing you want would be elements thousands of px apart.

Purpose

Container-fluids are great for image headers, banners, or navbars where the element doesn't lose readability as the screen gets larger.



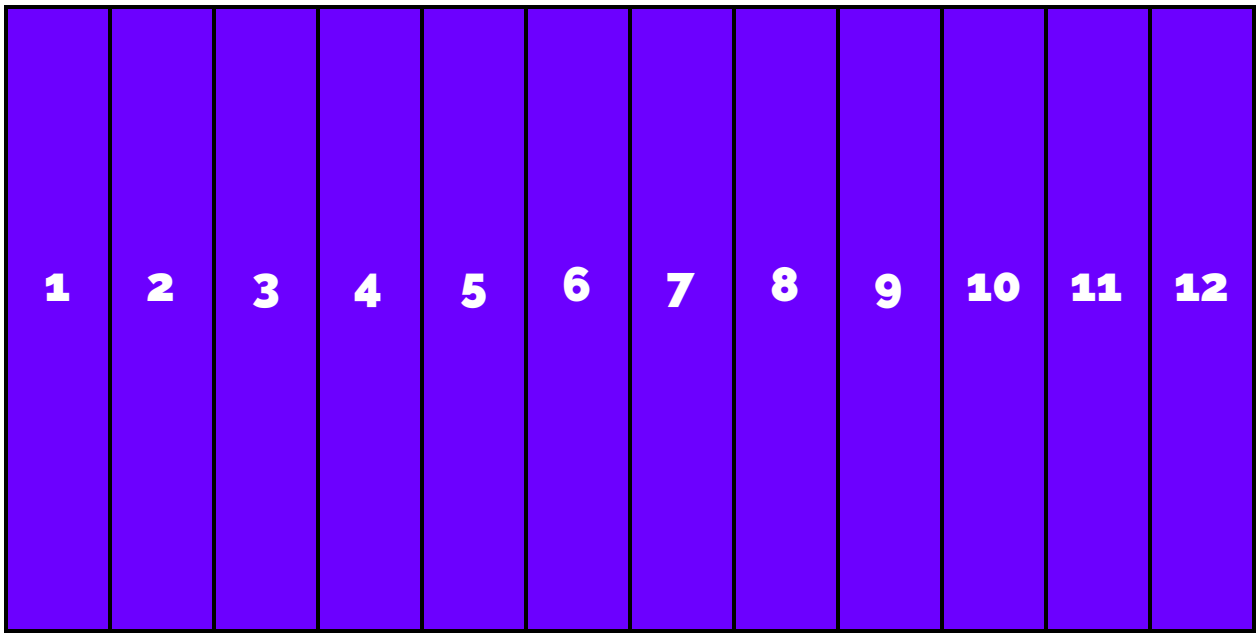
THE GRID SYSTEM

Use our powerful mobile-first flexbox grid to build layouts of all shapes and sizes thanks to a twelve column system, five default responsive tiers, Sass variables and mixins, and dozens of predefined classes.

How it works

Sudocore's grid system uses a series of containers, rows, and columns to layout and align content. It's built with flexbox and is fully responsive. Below is an example and an in-depth look at how the grid comes together.

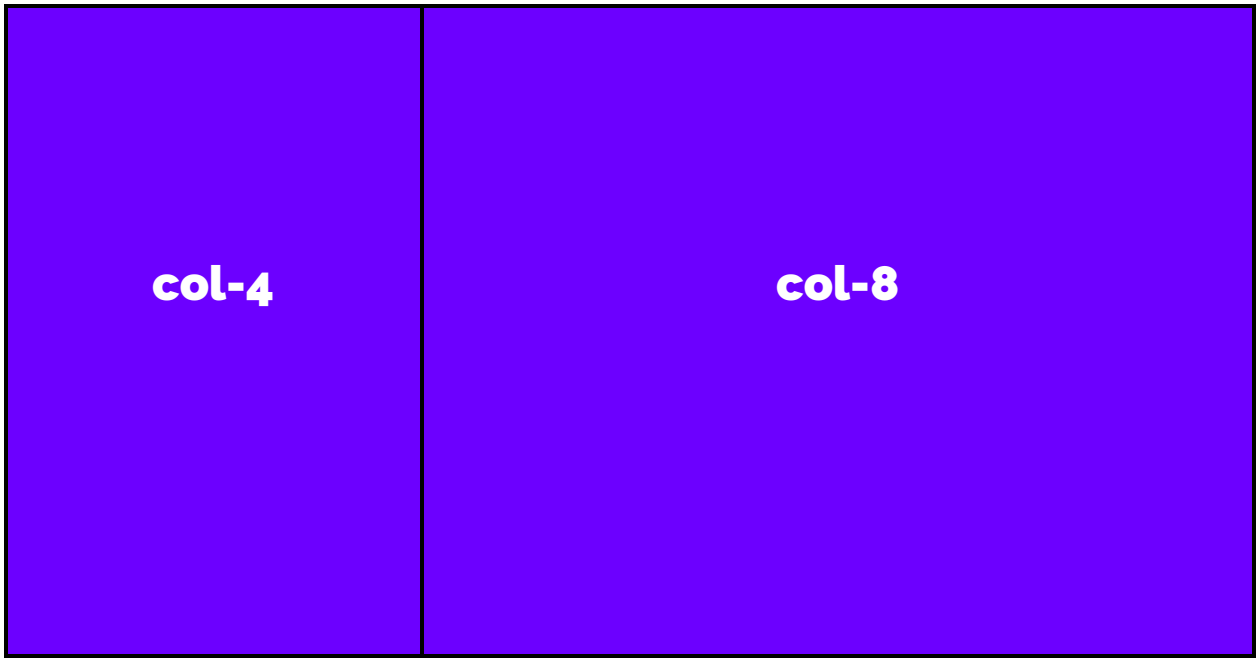
Our grid has been divided into 12 equal columns. This is true regardless of whether you are using a container or a container-fluid.



It is up to you to choose how these 12 columns are divided.

Columns with equal widths

Columns divided unevenly



SPACING

Sudocore includes a wide range of shorthand responsive margin and padding utility classes to modify an element's appearance.

How it works

Assign responsive-friendly margin or padding values to an element or a subset of its sides with shorthand classes. Includes support for individual properties, all properties, and vertical and horizontal properties. Classes are built from a default Sass map ranging from .25rem to 3rem.

Notation

Spacing utilities that apply to all breakpoints, from xs to xl, have no breakpoint abbreviation in them. This is because those classes are applied from min-width: 0 and up, and thus are not bound by a media query. The remaining breakpoints, however, do include a breakpoint abbreviation.

The classes are named using the format {property}{sides}-{size} for xs and {property}{sides}-{breakpoint}-{size} for sm, md, lg, and xl.

Where property is one of:

- ❖ m - for classes that set margin
- ❖ p - for classes that set padding

Where sides is one of:

- ❖ t - for classes that set margin-top or padding-top
- ❖ b - for classes that set margin-bottom or padding-bottom
- ❖ l - for classes that set margin-left or padding-left
- ❖ r - for classes that set margin-right or padding-right
- ❖ x - for classes that set both *-left and *-right
- ❖ y - for classes that set both *-top and *-bottom
- ❖ blank - for classes that set a margin or padding on all 4 sides of the element

Where size is one of:

- ❖ 0 - for classes that eliminate the margin or padding by setting it to 0
- ❖ 1 - (by default) for classes that set the margin or padding to $spacer * .25
- ❖ 2 - (by default) for classes that set the margin or padding to $spacer * .5
- ❖ 3 - (by default) for classes that set the margin or padding to $spacer

- ❖ 4 - (by default) for classes that set the margin or padding to $spacer * 1.5
- ❖ 5 - (by default) for classes that set the margin or padding to $spacer * 3
- ❖ auto - for classes that set the margin to auto

Horizontal Centering

Additionally, Sudocore also includes an .mx-auto class for horizontally centering fixed-width block level content—that is, content that has display: block and a width set—by setting the horizontal margins to auto.

03 Content

In this section, we will go over how basic HTML5 features react to Sudocore's Framework.

REBOOT

Reboot, a collection of element-specific CSS changes in a single file, kickstart Sudocore to provide an elegant, consistent, and simple baseline to build upon.

Approach

Reboot builds upon Normalize, providing many HTML elements with somewhat opinionated styles using only element selectors. Additional styling is done only with classes. For example, we reboot some <table> styles for a simpler baseline and later provide .table, .table-bordered, and more.

Here are our guidelines and reasons for choosing what to override in Reboot:

- ❖ Update some browser default values to use rems instead of ems for scalable component spacing.
- ❖ Avoid margin-top. Vertical margins can collapse, yielding unexpected results. More importantly though, a single direction of margin is a simpler mental model.
- ❖ For easier scaling across device sizes, block elements should use rems for margins.
- ❖ Keep declarations of font-related properties to a minimum, using inherit whenever possible.

Page Defaults

The <html> and <body> elements are updated to provide better page-wide defaults. More specifically:

- ❖ The box-sizing is globally set on every element—including *::before and *::after, to border-box. This ensures that the

declared width of element is never exceeded due to padding or border.

- ❖ No base font-size is declared on the <html>, but 16px is assumed (the browser default). font-size: 1rem is applied on the <body> for easy responsive type-scaling via media queries while respecting user preferences and ensuring a more accessible approach.
- ❖ The <body> also sets a global font-family, line-height, and text-align. This is inherited later by some form elements to prevent font inconsistencies.
- ❖ For safety, the <body> has a declared background-color, defaulting to #fff.

Native Font Stack

The default web fonts (Helvetica Neue, Helvetica, and Arial) have been dropped in Sudocore and replaced with a “native font stack” for optimum text rendering on every device and OS. Read more about native font stacks in this Smashing Magazine article.

This font-family is applied to the <body> and automatically inherited globally throughout Sudocore. To switch the global font-family, update $font-family-base and recompile Sudocore.

Heading and Paragraphs

All heading elements—e.g., <h1>—and <p> are reset to have their margin-top removed. Headings have margin-bottom: .5rem added and paragraphs margin-bottom: 1rem for easy spacing.

Lists

All lists—<ul>, <ol>, and <dl>—have their margin-top removed and a margin-bottom: 1rem. Nested lists have no margin-bottom.

For simpler styling, clear hierarchy, and better spacing, description lists have updated margins. <dd>s reset margin-left to 0 and add margin-bottom: .5rem. <dt>s are bolded.

Preformatted Text

The <pre> element is reset to remove its margin-top and use rem units for its margin-bottom.

Tables

Tables are slightly adjusted to style <caption>s, collapse borders, and ensure consistent text-align throughout.

Additional changes for borders, padding, and more come with the .table class.

Forms

Various form elements have been rebooted for simpler base styles. Here are some of the most notable changes:

- <fieldset>s have no borders, padding, or margin so they can be easily used as wrappers for individual inputs or groups of inputs.
- <legend>s, like fieldsets, have also been restyled to be displayed as a heading of sorts.
- <label>s are set to display: inline-block to allow margin to be applied.
- <input>s, <select>s, <textarea>s, and <button>s are mostly addressed by Normalize, but Reboot removes their margin and sets line-height: inherit, too.
- <textarea>s are modified to only be resizable vertically as horizontal resizing often “breaks” page layout.
- <button>s and <input> button elements have cursor: pointer when :not(:disabled).

Address

The <address> element is updated to reset the browser default font-style from italic to normal. line-height is also now inherited, and margin-bottom: 1rem has been added.

<address>s are for presenting contact information for the nearest ancestor (or an entire body of work). Preserve formatting by ending lines with
.

Blockquote

The default margin on blockquotes is 1em 40px, so we reset that to 0 0 1rem for something more consistent with other elements.

Inline Elements

The <abbr> element receives basic styling to make it stand out amongst paragraph text.

Summary

The default cursor on summary is text, so we reset that to pointer to convey that the element can be interacted with by clicking on it.

TYPOGRAPHY

Documentation and examples for Sudocore typography, including global settings, headings, body text, lists, and more.

Global Settings

Sudocore sets basic global display, typography, and link styles. When more control is needed, check out the textual utility classes.

- ❖ Use a native font stack that selects the best font-family for each OS and device.
- ❖ For a more inclusive and accessible type scale, we assume the browser default root font-size (typically 16px) so visitors can customize their browser defaults as needed.
- ❖ Use the $font-family-base, $font-size-base, and $line-height-base attributes as our typographic base applied to the <body>.
- ❖ Set the global link color via $link-color and apply link underlines only on :hover.
- ❖ Use $body-bg to set a background-color on the <body> (#fff by default).

These styles can be found within _reboot.scss, and the global variables are defined in _variables.scss. Make sure to set $font-size-base in rem.

Headings

All HTML headings, <h1> through <h6>, are available.

.h1 through .h6 classes are also available, for when you want to match the font styling of a heading but cannot use the associated HTML element.

Customizing Headings

Use the included utility classes to recreate the small secondary heading text from Sudocore 4.

Display Headings

Traditional heading elements are designed to work best in the meat of your page content. When you need a heading to stand out, consider using a display heading—a larger, slightly more opinionated heading style. Keep in mind these headings are not responsive by default, but it's possible to enable responsive font sizes.

Lead

Make a paragraph stand out by adding .lead.

Inline Text Elements

Styling for common inline HTML5 elements.

```
<p>You can use the mark tag to <mark>highlight</mark> text.</p>
<p><del>This line of text is meant to be treated as deleted text.</del></p>
<p><s>This line of text is meant to be treated as no longer accurate.</s></p>
```

```
<p><ins>This line of text is meant to be treated as an addition to the document.</ins></p>
<p><u>This line of text will render as underlined</u></p> <p><small>This line of text is meant to be treated as fine print.</small></p>
<p><strong>This line rendered as bold text.</strong></p> <p><em>This line rendered as italicized text.</em></p>
```

.mark and .small classes are also available to apply the same styles as <mark> and <small> while avoiding any unwanted semantic implications that the tags would bring.

While not shown above, feel free to use <b> and <i> in HTML5. <b> is meant to highlight words or phrases without conveying additional importance while <i> is mostly for voice, technical terms, etc.

Text Utilities

Change text alignment, transform, style, weight, and color with our text utilities and color utilities.

Abbreviations

Stylized implementation of HTML's <abbr> element for abbreviations and acronyms to show the expanded version on hover. Abbreviations have a default underline and gain a help cursor to provide additional context on hover and to users of assistive technologies.

Add .initialism to an abbreviation for a slightly smaller font-size.

Blockquotes

For quoting blocks of content from another source within your document. Wrap <blockquote class="blockquote"> around any HTML as the quote.

Naming a source

Add a <footer class="blockquote-footer"> for identifying the source. Wrap the name of the source work in <cite>.

Alignment

Use text utilities as needed to change the alignment of your blockquote.

Unstyled Lists

Remove the default list-style and left margin on list items (immediate children only). This only applies to immediate children list items, meaning you will need to add the class for any nested lists as well.

Inline Lists

Remove a list's bullets and apply some light margin with a combination of two classes, .list-inline and .list-inline-item.

Description List Alignment

Align terms and descriptions horizontally by using our grid system's predefined classes (or semantic mixins). For longer terms, you can optionally add a .text-truncate class to truncate the text with an ellipsis.

Responsive Font Sizes

Sudocore v4.3.0 ships with the option to enable responsive font sizes, allowing text to scale more naturally across device and viewport sizes. RFS can be enabled by changing the $enable-responsive-font-sizes Sass variable to true and recompiling Bootstrap.

To support RFS, we use a Sass mixin to replace our normal font-size properties. Responsive font sizes will be compiled into calc() functions with a mix of rem and viewport units to enable the responsive scaling behavior. More about RFS and its configuration can be found on its GitHub repository.

CODE

Documentation and examples for displaying inline and multiline blocks of code with Sudocore.

Inline Code

Wrap inline snippets of code with <code>. Be sure to escape HTML angle brackets.

Code Blocks

Use <pre>s for multiple lines of code. Once again, be sure to escape any angle brackets in the code for proper rendering. You may optionally add the .pre-scrollable class, which will set a max-height of 340px and provide a y-axis scrollbar.

Variables

For indicating variables use the <var> tag.

User Input

Use the <kbd> to indicate input that is typically entered via keyboard.

Sample Output

For indicating sample output from a program use the <samp> tag.

IMAGES

Documentation and examples for opting images into responsive behavior (so they never become larger than their parent elements) and add lightweight styles to them—all via classes.

Responsive Images

Images in Sudocore are made responsive with .img-fluid. max-width: 100%; and height: auto; are applied to the image so that it scales with the parent element.

Image Thumbnails

In addition to our border-radius utilities, you can use .img-thumbnail to give an image a rounded 1px border appearance.

Aligning Images

Align images with the helper float classes or text alignment classes. block-level images can be centered using the .mx-auto margin utility class.

Picture

If you are using the <picture> element to specify multiple <source> elements for a specific <img>, make sure to add the .img-* classes to the <img> and not to the <picture> tag.

TABLES

Documentation and examples for opt-in styling of tables (given their prevalent use in JavaScript plugins) with Sudocore.

Basics

Due to the widespread use of tables across third-party widgets like calendars and date pickers, we've designed our tables to be opt-in. Just add the base class .table to any <table>, then extend with custom styles or our various included modifier classes.

Using the most basic table markup, here's how .table-based tables look in Sudocore. All table styles are inherited in Sudocore 4, meaning any nested tables will be styled in the same manner as the parent.

You can also invert the colors—with light text on dark backgrounds—with .table-dark.

Table Head Options

Similar to tables and dark tables, use the modifier classes .thead-light or .thead-dark to make <thead>s appear light or dark gray.

Striped Rows

Use .table-striped to add zebra-striping to any table row within the <tbody>.

Bordered Table

Add .table-bordered for borders on all sides of the table and cells.

Borderless Table

Add .table-borderless for a table without borders.

Hoverable Rows

Add .table-hover to enable a hover state on table rows within a <tbody>.

Small Table

Add .table-sm to make tables more compact by cutting cell padding in half.

Contextual Classes

Use contextual classes to color table rows or individual cells.

Regular table background variants are not available with the dark table, however, you may use text or background utilities to achieve similar styles.

Create responsive tables by wrapping any .table with .table-responsive{-sm|-md|-lg|-xl}, making the table scroll horizontally at each max-width breakpoint of up to (but not including) 576px, 768px, 992px, and 1120px, respectively.

Captions

A <caption> functions like a heading for a table. It helps users with screen readers to find a table and understand what it's about and decide if they want to read it.

Responsive Tables

Responsive tables allow tables to be scrolled horizontally with ease. Make any table responsive across all viewports by wrapping a .table with .table-responsive. Or, pick a maximum breakpoint with which to have a responsive table up to by using .table-responsive{-sm|-md|-lg|-xl}.

Always Responsive

Across every breakpoint, use .table-responsive for horizontally scrolling tables.

Breakpoint Specific

Use .table-responsive{-sm|-md|-lg|-xl} as needed to create responsive tables up to a particular breakpoint. From that breakpoint and up, the table will behave normally and not scroll horizontally.

FIGURES

Documentation and examples for displaying related images and text with the figure component in Sudocore.

Basics

Anytime you need to display a piece of content—like an image with an optional caption, consider using a <figure>.

Use the included .figure , .figure-img and .figure-caption classes to provide some baseline styles for the HTML5 <figure> and <figcaption> elements. Images in figures have no explicit size, so be sure to add the .img-fluid class to your <img> to make it responsive.

Aligning the figure's caption is easy with our text utilities.

04 Components

In this section, we will go over all the components Sudocore has to offer.

ALERTS

Provide contextual feedback messages for typical user actions with the handful of available and flexible alert messages.

Examples

Alerts are available for any length of text, as well as an optional dismiss button. For proper styling, use one of the eight required contextual classes (e.g., .alert-success). For inline dismissal, use the alerts jQuery plugin.

```
alert alert-{{color}}
```

Link Color

Use the .alert-link utility class to quickly provide matching colored links within any alert.

Additional Content

Alerts can also contain additional HTML elements like headings, paragraphs and dividers.

Dismissing Content

Using the alert JavaScript plugin, it's possible to dismiss any alert inline. Here's how:

- ❖ Be sure you've loaded the alert plugin, or the compiled Bootstrap JavaScript.
- ❖ If you're building our JavaScript from source, it requires util.js. The compiled version includes this.
- ❖ Add a dismiss button and the .alert-dismissible class, which adds extra padding to the right of the alert and positions the .close button.
- ❖ On the dismiss button, add the data-dismiss="alert" attribute, which triggers the JavaScript functionality. Be sure to use the <button> element with it for proper behavior across all devices.
- ❖ To animate alerts when dismissing them, be sure to add the .fade and .show classes.

Triggers

Using the alert JavaScript plugin, it's possible to dismiss any alert inline.

Enable dismissal of an alert via JavaScript:

```
$('.alert').alert()
```

Or with data attributes on a button within the alert, as demonstrated above:

```
<button type="button" class="close" data-dismiss="alert" aria-label="Close">
    <span aria-hidden="true">&times;</span>
</button>
```

Note that closing an alert will remove it from the DOM.

Methods

Method	Description
$().alert()	Makes an alert listen for click events on descendant elements which have the data-dismiss="alert" attribute. (Not necessary when using the data-api's auto-initialization.)
$().alert('close')	Closes an alert by removing it from the DOM. If the .fade and .show classes are present on the element, the alert will fade out before it is removed.
$().alert('dispose')	Destroys an element's alert.

Events

Bootstrap's alert plugin exposes a few events for hooking into alert functionality.

Event	Description
close.bs.alert	This event fires immediately when the close instance

	method is called.
closed.bs.alert	This event is fired when the alert has been closed (will wait for CSS transitions to complete).

BADGE

Documentation and examples for badges, our small count and labeling component.

Examples

Badges scale to match the size of the immediate parent element by using relative font sizing and em units.

Badges can be used as part of links or buttons to provide a counter.

Note that depending on how they are used, badges may be confusing for users of screen readers and similar assistive technologies. While the styling of badges provides a visual cue as to their purpose, these users will simply be presented with the content of the badge. Depending on the specific situation, these badges may seem like random additional words or numbers at the end of a sentence, link, or button.

Unless the context is clear (as with the “Notifications” example, where it is understood that the “4” is the number of notifications), consider including additional context with a visually hidden piece of additional text.

Contextual Variations

Add any of the below mentioned modifier classes to change the appearance of a badge.

Pill Badges

Use the .badge-pill modifier class to make badges more rounded (with a larger border-radius and additional horizontal padding). Useful if you miss the badges from v3.

Links

Using the contextual .badge-* classes on an <a> element quickly provide actionable badges with hover and focus states.

BREADCRUMB

Indicate the current page’s location within a navigational hierarchy that automatically adds separators via CSS.

Changing the Separator

Separators are automatically added in CSS through ::before and content. They can be changed by changing $breadcrumb-divider. The quote function is needed to generate the quotes around a string, so if you want > as separator, you can use this:

It's also possible to use a base64 embedded SVG icon:

The separator can be removed by setting $breadcrumb-divider to none:

Accessibility

Since breadcrumbs provide a navigation, it's a good idea to add a meaningful label such as aria-label="breadcrumb" to describe the type of navigation provided in the <nav> element, as well as applying an aria-current="page" to the last item of the set to indicate that it represents the current page.

For more information, see the WAI-ARIA Authoring Practices for the breadcrumb pattern.

BUTTONS

Use Sudocore's custom button styles for actions in forms, dialogs, and more with support for multiple sizes, states, and more.

Example

Sudocore includes several predefined button styles, each serving its own semantic purpose, with a few extras thrown in for more control.

Disable Text Wrapping

If you don't want the button text to wrap, you can add the .text-nowrap class to the button. In Sass, you can set $btn-white-space: nowrap to disable text wrapping for each button.

Button Tags

The .btn classes are designed to be used with the <button> element. However, you can also use these classes on <a> or <input> elements (though some browsers may apply a slightly different rendering).

When using button classes on <a> elements that are used to trigger in-page functionality (like collapsing content), rather than linking to new pages or sections within the current page,

these links should be given a role="button" to appropriately convey their purpose to assistive technologies such as screen readers.

Outline Buttons

In need of a button, but not the hefty background colors they bring? Replace the default modifier classes with the .btn-outline-* ones to remove all background images and colors on any button.

Sizes

Fancy larger or smaller buttons? Add .btn-lg or .btn-sm for additional sizes.

Create block level buttons—those that span the full width of a parent—by adding .btn-block.

Active State

Buttons will appear pressed (with a darker background, darker border, and inset shadow) when active. There's no need to add a class to <button>s as they use a pseudo-class. However, you can still force the same active appearance with .active (and include the aria-pressed="true" attribute) should you need to replicate the state programmatically.

Disable State

Make buttons look inactive by adding the disabled boolean attribute to any <button> element.

Disabled buttons using the <a> element behave a bit different:

- ❖ <a>s don't support the disabled attribute, so you must add the .disabled class to make it visually appear disabled.
- ❖ Some future-friendly styles are included to disable all pointer-events on anchor buttons. In browsers which support that property, you won't see the disabled cursor at all.
- ❖ Disabled buttons should include the aria-disabled="true" attribute to indicate the state of the element to assistive technologies.

Button Plugin

Do more with buttons. Control button states or creates groups of buttons for more components like toolbars.

Toggle States

Add data-toggle="button" to toggle a button's active state. If you're pre-toggling a button, you must manually add the .active class and aria-pressed="true" to the <button>.

Checkbox and Radio Buttons

Bootstrap's .button styles can be applied to other elements, such as <label>s, to provide checkbox or radio style button toggling. Add data-toggle="buttons" to a .btn-group containing those modified buttons to enable their toggling behavior via JavaScript and add .btn-group-toggle to style the <input>s within your buttons. **Note that you can create single input-powered buttons or groups of them.**

The checked state for these buttons is **only updated via** click **event** on the button. If you use another method to update the input—e.g., with <input type="reset"> or by manually applying the input's checked property—you'll need to toggle .active on the <label> manually.

Note that pre-checked buttons require you to manually add the .active class to the input's <label>.

Methods

Methods	Description
$().button('toggle')	Toggles push the state. Gives the button the appearance that it has been activated.
$().button(dispose)	Destroys an element's button.

BUTTON GROUP

Group a series of buttons together on a single line with the button group, and super-power them with JavaScript.

Basic Example

Wrap a series of buttons with .btn in .btn-group. Add on optional JavaScript radio and checkbox style behavior with our buttons plugin.

Button Toolbar

Combine sets of button groups into button toolbars for more complex components. Use utility classes as needed to space out groups, buttons, and more.

Feel free to mix input groups with button groups in your toolbars. Similar to the example above, you'll likely need some utilities though to space things properly.

Sizing

Instead of applying button sizing classes to every button in a group, just add .btn-group-* to each .btn-group, including each one when nesting multiple groups.

Nesting

Place a .btn-group within another .btn-group when you want dropdown menus mixed with a series of buttons.

Vertical Variation

Make a set of buttons appear vertically stacked rather than horizontally. **Split button dropdowns are not supported here.**

CARD

Sudocore's cards provide a flexible and extensible content container with multiple variants and options.

About

A card is a flexible and extensible content container. It includes options for headers and footers, a wide variety of content, contextual background colors, and powerful display options. If you're familiar with Sudocore 3, cards replace our old panels, wells, and thumbnails. Similar functionality to those components is available as modifier classes for cards.

Example

Cards are built with as little markup and styles as possible, but still manage to deliver a ton of control and customization. Built with flexbox, they offer easy alignment and mix well with other Bootstrap components. They have no margin by default, so use spacing utilities as needed.

Below is an example of a basic card with mixed content and a fixed width. Cards have no fixed width to start, so they'll naturally fill the full width of its parent element. This is easily customized with our various sizing options.

Content Types

Cards support a wide variety of content, including images, text, list groups, links, and more. Below are examples of what's supported.

Body

The building block of a card is the .card-body. Use it whenever you need a padded section within a card.

Titles, Text, and Links

Card titles are used by adding .card-title to a <h*> tag. In the same way, links are added and placed next to each other by adding .card-link to an <a> tag.

Subtitles are used by adding a .card-subtitle to a <h*> tag. If the .card-title and the .card-subtitle items are placed in a .card-body item, the card title and subtitle are aligned nicely.

Images

.card-img-top places an image to the top of the card. With .card-text, text can be added to the card. Text within .card-text can also be styled with the standard HTML tags.

List Groups

Create lists of content in a card with a flush list group.

Kitchen Sink

Mix and match multiple content types to create the card you need, or throw everything in there. Shown below are image styles, blocks, text styles, and a list group—all wrapped in a fixed-width card.

Header and Footer

Add an optional header and/or footer within a card.

Card headers can be styled by adding .card-header to <h*> elements.

Sizing

Cards assume no specific width to start, so they'll be 100% wide unless otherwise stated. You can change this as needed with custom CSS, grid classes, grid Sass mixins, or utilities.

Using Grid Markup

Using the grid, wrap cards in columns and rows as needed.

Using Utilities

Use our handful of available sizing utilities to quickly set a card's width.

Using Custom CSS

Use custom CSS in your stylesheets or as inline styles to set a width.

Text Alignment

You can quickly change the text alignment of any card—in its entirety or specific parts—with our text align classes.

Navigation

Add some navigation to a card's header (or block) with Bootstrap's nav components.

Images

Cards include a few options for working with images. Choose from appending “image caps” at either end of a card, overlaying images with card content, or simply embedding the image in a card.

Images Caps

Similar to headers and footers, cards can include top and bottom “image caps”—images at the top or bottom of a card.

Images Overlays

Turn an image into a card background and overlay your card’s text. Depending on the image, you may or may not need additional styles or utilities.

Horizontal

Using a combination of grid and utility classes, cards can be made horizontal in a mobile-friendly and responsive way. In the example below, we remove the grid gutters with .no-gutters and use .col-md-* classes to make the card horizontal at the md breakpoint. Further adjustments may be needed depending on your card content.

Card Styles

Cards include various options for customizing their backgrounds, borders, and color.

Background and Color

Use text and background utilities to change the appearance of a card.

Border

Use border utilities to change just the border-color of a card. Note that you can put .text-{color} classes on the parent .card or a subset of the card's contents as shown below.

Mixin Utilities

You can also change the borders on the card header and footer as needed, and even remove their background-color with .bg-transparent.

Card Layout

In addition to styling the content within cards, Bootstrap includes a few options for laying out a series of cards. For the time being, these layout options are not yet responsive.

Card Groups

Use card groups to render cards as a single, attached element with equal width and height columns. Card groups use display: flex; to achieve their uniform sizing.

When using card groups with footers, their content will automatically line up.

Card Decks

Need a set of equal width and height cards that aren't attached to one another? Use card decks.

Just like with card groups, card footers in decks will automatically line up.

Grid Cards

Use the Bootstrap grid system and its .row-cols classes to control how many grid columns (wrapped around your cards) you show per row. For example, here's .row-cols-1 laying out the cards on one column, and .row-cols-md-2 splitting four cards to equal width across multiple rows, from the medium breakpoint up.

Change it to .row-cols-3 and you'll see the fourth card wrap.

When you need equal height, add .h-100 to the cards. If you want equal heights by default, you can set $card-height: 100% in Sass.

Card Columns

Cards can be organized into Masonry-like columns with just CSS by wrapping them in .card-columns. Cards are built with CSS column properties instead of flexbox for easier alignment. Cards are ordered from top to bottom and left to right.

Heads up! Your mileage with card columns may vary. To prevent cards breaking across columns, we must set them to display: inline-block as column-break-inside: avoid isn't a bulletproof solution yet.

Card columns can also be extended and customized with some additional code. Shown below is an extension of the .card-columns class using the same CSS we use—CSS columns— to generate a set of responsive tiers for changing the number of columns.

CAROUSEL

A slideshow component for cycling through elements—images or slides of text—like a carousel.

How it works

The carousel is a slideshow for cycling through a series of content, built with CSS 3D transforms and a bit of JavaScript. It works with a series of images, text, or custom markup. It also includes support for previous/next controls and indicators.

In browsers where the Page Visibility API is supported, the carousel will avoid sliding when the webpage is not visible to the user (such as when the browser tab is inactive, the browser window is minimized, etc.).

Please be aware that nested carousels are not supported, and carousels are generally not compliant with accessibility standards.

Lastly, if you're building our JavaScript from source, it requires util.js.

Example

Carousels don't automatically normalize slide dimensions. As such, you may need to use additional utilities or custom styles to appropriately size content. While carousels support

previous/next controls and indicators, they're not explicitly required. Add and customize as you see fit.

The .active class needs to be added to one of the slides otherwise the carousel will not be visible. Also be sure to set a unique id on the .carousel for optional controls, especially if you're using multiple carousels on a single page. Control and indicator elements must have a data-target attribute (or href for links) that matches the id of the .carousel element.

Slides Only

Here's a carousel with slides only. Note the presence of the .d-block and .w-100 on carousel images to prevent browser default image alignment.

```
<div id="carouselExampleSlidesOnly" class="carousel slide"
data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
        <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
</div>
```

With Controls

Adding in the previous and next controls:

```
<div id="carouselExampleControls" class="carousel slide"
data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleControls"
role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleControls"
role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

With Indicators

You can also add the indicators to the carousel, alongside the controls, too.

```
<div id="carouselExampleIndicators" class="carousel slide"
data-ride="carousel">
  <ol class="carousel-indicators">
```

```
    <li data-target="#carouselExampleIndicators" data-slide-to="0"
class="active"></li>
    <li data-target="#carouselExampleIndicators" data-slide-to="1"></li>
    <li data-target="#carouselExampleIndicators" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleIndicators"
role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleIndicators"
role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

With Captions

Add captions to your slides easily with the .carousel-caption element within any .carousel-item. They can be easily hidden on smaller viewports, as shown below, with optional display utilities. We hide them initially with .d-none and bring them back on medium-sized devices with .d-md-block.

```
<div id="carouselExampleCaptions" class="carousel slide"
data-ride="carousel">
  <ol class="carousel-indicators">
```

```
    <li data-target="#carouselExampleCaptions" data-slide-to="0"
class="active"></li>
    <li data-target="#carouselExampleCaptions" data-slide-to="1"></li>
    <li data-target="#carouselExampleCaptions" data-slide-to="2"></li>
  </ol>
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
      <div class="carousel-caption d-none d-md-block">
        <h5>First slide label</h5>
        <p>Nulla vitae elit libero, a pharetra augue mollis
interdum.</p>
      </div>
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
      <div class="carousel-caption d-none d-md-block">
        <h5>Second slide label</h5>
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</p>
      </div>
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
      <div class="carousel-caption d-none d-md-block">
        <h5>Third slide label</h5>
        <p>Praesent commodo cursus magna, vel scelerisque nisl
consectetur.</p>
      </div>
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleCaptions"
role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleCaptions"
role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

Crossfade

Add .carousel-fade to your carousel to animate slides with a fade transition instead of a slide.

```
<div id="carouselExampleFade" class="carousel slide carousel-fade"
data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleFade"
role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleFade"
role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

Individual .carousel-item interval

Add data-interval="" to a .carousel-item to change the amount of time to delay between automatically cycling to the next item.

```
<div id="carouselExampleInterval" class="carousel slide"
data-ride="carousel">
  <div class="carousel-inner">
    <div class="carousel-item active" data-interval="10000">
      <img src="..." class="d-block w-100" alt="...">
    </div>
```

```
    <div class="carousel-item" data-interval="2000">
      <img src="..." class="d-block w-100" alt="...">
    </div>
    <div class="carousel-item">
      <img src="..." class="d-block w-100" alt="...">
    </div>
  </div>
  <a class="carousel-control-prev" href="#carouselExampleInterval"
role="button" data-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="carousel-control-next" href="#carouselExampleInterval"
role="button" data-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>
</div>
```

Usage Via Data Attributes

Use data attributes to easily control the position of the carousel. data-slide accepts the keywords prev or next, which alters the slide position relative to its current position. Alternatively, use data-slide-to to pass a raw slide index to the carousel data-slide-to="2", which shifts the slide position to a particular index beginning with 0.

The data-ride="carousel" attribute is used to mark a carousel as animating starting at page load. If you don't use data-ride="carousel" to initialize your carousel, you have to initialize it yourself. It cannot be used in combination with (redundant and unnecessary) explicit JavaScript initialization of the same carousel.

Usage Via JavaScript

Call carousel manually with:

```
$('.carousel').carousel()
```

Options

Options can be passed via data attributes or JavaScript. For data attributes, append the option name to data-, as in data-interval="".

Name	Type	Default	Description
Interval	number	5000	
Keyboard	boolean	true	
Pause	string | boolean	“hover”	