Introduction

Get started with Bootstrap, the world's most popular framework for building responsive, mobile-first sites, with BootstrapCDN and a template starter page.

Quick start

Looking to quickly add Bootstrap to your project? Use BootstrapCDN, provided for free by the folks at MaxCDN. Using a package manager or need to download the source files?

CSS

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

```
<link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">
```

JS

Many of our components require the use of JavaScript to function. Specifically, they require jQuery, Popper.js, and our own JavaScript plugins. Place the following `<script>`s 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 JavaScript plugins.

We use jQuery's slim build, but the full version is also supported.

```
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
```

Curious which components explicitly require jQuery, our JS, and Popper.js? Click the show components link below. If you're at all unsure about the general page structure, keep reading for an example page template.

Show components requiring JavaScript

Starter template

Be sure to 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">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity="sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm"
crossorigin="anonymous">

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

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity="sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN"
crossorigin="anonymous"></script>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q"
crossorigin="anonymous"></script>
    <script
src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl"
crossorigin="anonymous"></script>
  </body>
</html>
```

Important globals

Bootstrap 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

Bootstrap 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

Bootstrap 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">
```

You can see an example of this in action in the starter template.

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`.

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.

Community

Stay up to date on the development of Bootstrap and reach out to the community with these helpful resources.

- Follow @getbootstrap on Twitter.
- Read and subscribe to The Official Bootstrap Blog.
- Join the official Slack room.
- Chat with fellow Bootstrappers in IRC. On the `irc.freenode.net` server, in the `##bootstrap` channel.

- Implementation help may be found at Stack Overflow (tagged `bootstrap-4`).
- Developers should use the keyword `bootstrap` on packages which modify or add to the functionality of Bootstrap when distributing through npm or similar delivery mechanisms for maximum discoverability.

Contents

Discover what's included in Bootstrap, including our precompiled and source code flavors. Remember, Bootstrap's JavaScript plugins require jQuery.

Precompiled Bootstrap

Once downloaded, unzip the compressed folder and you'll see something like this:

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap.min.css.map
│   ├── bootstrap-grid.css
│   ├── bootstrap-grid.css.map
│   ├── bootstrap-grid.min.css
│   ├── bootstrap-grid.min.css.map
│   ├── bootstrap-reboot.css
│   ├── bootstrap-reboot.css.map
│   ├── bootstrap-reboot.min.css
│   └── bootstrap-reboot.min.css.map
└── js/
    ├── bootstrap.bundle.js
    ├── bootstrap.bundle.min.js
    ├── bootstrap.js
    └── bootstrap.min.js
```

This is the most basic form of Bootstrap: precompiled files for quick drop-in usage in nearly any web project. We provide compiled CSS and JS (`bootstrap.*`), as well as compiled and minified CSS and JS (`bootstrap.min.*`). CSS source maps(`bootstrap.*.map`) are available for use with certain browsers' developer tools. Bundled JS files (`bootstrap.bundle.js` and minified `bootstrap.bundle.min.js`) include Popper, but not jQuery.

CSS files

Bootstrap includes a handful of options for including some or all of our compiled CSS.

CSS files	Layout	Content	Components	Utilities
`bootstrap.css`	Included	Included	Included	Included

CSS files	Layout	Content	Components	Utilities
`bootstrap.min.css`				
`bootstrap-grid.css` `bootstrap-grid.min.css`	Only grid system	Not included	Not included	Only flex utilities
`bootstrap-reboot.css` `bootstrap-reboot.min.css`	Not included	Only Reboot	Not included	Not included

JS files

Similarly, we have options for including some or all of our compiled JavaScript.

JS files	Popper	jQuery
`bootstrap.bundle.js` `bootstrap.bundle.min.js`	Included	Not included
`bootstrap.js` `bootstrap.min.js`	Not included	Not included

Bootstrap source code

The Bootstrap source code download includes the precompiled CSS and JavaScript assets, along with source Sass, JavaScript, and documentation. More specifically, it includes the following and more:

```
bootstrap/
├── dist/
│   ├── css/
│   └── js/
├── docs/
│   └── examples/
├── js/
└── scss/
```

The `scss/` and `js/` are the source code for our CSS and JavaScript. The `dist/` folder includes everything listed in the precompiled download section above.
The `docs/` folder includes the source code for our documentation, and `examples/` of Bootstrap usage. Beyond that, any other included file provides support for packages, license information, and development.

Browsers and devices

Learn about the browsers and devices, from modern to old, that are supported by Bootstrap, including known quirks and bugs for each.

Supported browsers

Bootstrap supports the **latest, stable releases** of all major browsers and platforms. On Windows, **we support Internet Explorer 10-11 / Microsoft Edge**.

Alternative browsers which use the latest version of WebKit, Blink, or Gecko, whether directly or via the platform's web view API, are not explicitly supported. However, Bootstrap should (in most cases) display and function correctly in these browsers as well. More specific support information is provided below.

Mobile devices

Generally speaking, Bootstrap supports the latest versions of each major platform's default browsers. Note that proxy browsers (such as Opera Mini, Opera Mobile's Turbo mode, UC Browser Mini, Amazon Silk) are not supported.

	Chrome	Firefox	Safari	Android Browser & WebView	Microsoft Edge
Android	Supported	Supported	N/A	Android v5.0+ supported	Supported
iOS	Supported	Supported	Supported	N/A	Supported
Windows 10 Mobile	N/A	N/A	N/A	N/A	Supported

Desktop browsers

Similarly, the latest versions of most desktop browsers are supported.

	Chrome	Firefox	Internet Explorer	Microsoft Edge	Opera	Safari
Mac	Supported	Supported	N/A	N/A	Supported	Supported
Windows	Supported	Supported	Supported, IE10+	Supported	Supported	Not supported

For Firefox, in addition to the latest normal stable release, we also support the latest Extended Support Release (ESR) version of Firefox.

Unofficially, Bootstrap should look and behave well enough in Chromium and Chrome for Linux, Firefox for Linux, and Internet Explorer 9, though they are not officially supported.

For a list of some of the browser bugs that Bootstrap has to grapple with, see our Wall of browser bugs.

Internet Explorer

Internet Explorer 10+ is supported; IE9 and down is not. Please be aware that some CSS3 properties and HTML5 elements are not fully supported in IE10, or require prefixed properties for full functionality. Visit Can I use... for details on browser support of CSS3 and HTML5 features.

If you require IE8-9 support, use Bootstrap 3. It's the most stable version of our code and is still supported by our team for critical bugfixes and documentation changes. However, no new features will be added to it.

Modals and dropdowns on mobile

Overflow and scrolling

Support for `overflow: hidden;` on the `<body>` element is quite limited in iOS and Android. To that end, when you scroll past the top or bottom of a modal in either of those devices' browsers, the `<body>` content will begin to scroll. See Chrome bug #175502 (fixed in Chrome v40) and WebKit bug #153852.

iOS text fields and scrolling

As of iOS 9.2, while a modal is open, if the initial touch of a scroll gesture is within the boundary of a textual `<input>` or a `<textarea>`, the `<body>` content underneath the modal will be scrolled instead of the modal itself. See WebKit bug #153856.

Navbar Dropdowns

The `.dropdown-backdrop` element isn't used on iOS in the nav because of the complexity of z-indexing. Thus, to close dropdowns in navbars, you must directly click the dropdown element (or any other element which will fire a click event in iOS).

Browser zooming

Page zooming inevitably presents rendering artifacts in some components, both in Bootstrap and the rest of the web. Depending on the issue, we may be able to fix it (search first and then open an issue if need be). However, we tend to ignore these as they often have no direct solution other than hacky workarounds.

Sticky `:hover`/`:focus` on iOS

While `:hover` isn't possible on most touch devices, iOS emulates this behavior, resulting in "sticky" hover styles that persist after tapping one element. These hover styles are only removed when users tap another element. This behavior is considered largely undesirable and appears to not be an issue on Android or Windows devices.

Throughout our v4 alpha and beta releases, we included incomplete and commented out code for opting into a media query shim that would disable hover styles in touch device browsers that emulate hovering. This work was never fully completed or enabled, but to avoid complete breakage, we've opted to deprecate this shim and keep the mixins as shortcuts for the pseudo-classes.

Printing

Even in some modern browsers, printing can be quirky.

As of Safari v8.0, use of the fixed-width `.container` class can cause Safari to use an unusually small font size when printing. See issue #14868 and WebKit bug #138192 for more details. One potential workaround is the following CSS:

```
@media print {
  .container {
    width: auto;
  }
}
```

Android stock browser

Out of the box, Android 4.1 (and even some newer releases apparently) ship with the Browser app as the default web browser of choice (as opposed to Chrome). Unfortunately, the Browser app has lots of bugs and inconsistencies with CSS in general.

Select menu

On `<select>` elements, the Android stock browser will not display the side controls if there is a `border-radius` and/or `border` applied. (See this StackOverflow question for details.) Use the snippet of code below to remove the offending CSS and render the `<select>` as an unstyled element on the Android stock browser. The user agent sniffing avoids interference with Chrome, Safari, and Mozilla browsers.

```
<script>
$(function () {
  var nua = navigator.userAgent
  var isAndroid = (nua.indexOf('Mozilla/5.0') > -1 && nua.indexOf('Android ') > -1
&& nua.indexOf('AppleWebKit') > -1 && nua.indexOf('Chrome') === -1)
  if (isAndroid) {
    $('select.form-control').removeClass('form-control').css('width', '100%')
  }
})
</script>
```

Want to see an example? Check out this JS Bin demo.

Validators

In order to provide the best possible experience to old and buggy browsers, Bootstrap uses CSS browser hacks in several places to target special CSS to certain browser versions in order to work around bugs in the browsers themselves. These hacks understandably cause CSS validators to complain that they are invalid. In a couple places, we also use bleeding-edge CSS features that aren't yet fully standardized, but these are used purely for progressive enhancement.

These validation warnings don't matter in practice since the non-hacky portion of our CSS does fully validate and the hacky portions don't interfere with the proper functioning of the non-hacky portion, hence why we deliberately ignore these particular warnings.

Our HTML docs likewise have some trivial and inconsequential HTML validation warnings due to our inclusion of a workaround for a certain Firefox bug.

JavaScript

Bring Bootstrap to life with our optional JavaScript plugins built on jQuery. Learn about each plugin, our data and programmatic API options, and more.

Individual or compiled

Plugins can be included individually (using Bootstrap's individual `*.js` files), or all at once using `bootstrap.js` or the minified `bootstrap.min.js` (don't include both).

Dependencies

Some plugins and CSS components depend on other plugins. If you include plugins individually, make sure to check for these dependencies in the docs. Also note that **all plugins depend on jQuery** (this means jQuery must be included **before** the plugin files). Consult our `package.json` to see which versions of jQuery are supported.

Our dropdowns, popovers and tooltips also depend on Popper.js.

Data attributes

Nearly all Bootstrap plugins can be enabled and configured through HTML alone with data attributes (our preferred way of using JavaScript functionality). Be sure to **only use one set of data attributes on a single element** (e.g., you cannot trigger a tooltip and modal from the same button.)

However, in some situations it may be desirable to disable this functionality. To disable the data attribute API, unbind all events on the document namespaced with `data-api`like so:

```
$(document).off('.data-api')
```

Alternatively, to target a specific plugin, just include the plugin's name as a namespace along with the data-api namespace like this:

```
$(document).off('.alert.data-api')
```

Events

Bootstrap provides custom events for most plugins' unique actions. Generally, these come in an infinitive and past participle form - where the infinitive (ex. `show`) is triggered at the start of an event, and its past participle form (ex. `shown`) is triggered on the completion of an action.

All infinitive events provide `preventDefault()` functionality. This provides the ability to stop the execution of an action before it starts. Returning false from an event handler will also automatically call `preventDefault()`.

```
$('#myModal').on('show.bs.modal', function (e) {
  if (!data) return e.preventDefault() // stops modal from being shown
})
```

Programmatic API

We also believe you should be able to use all Bootstrap plugins purely through the JavaScript API. All public APIs are single, chainable methods, and return the collection acted upon.

```
$('.btn.danger').button('toggle').addClass('fat')
```

All methods should accept an optional options object, a string which targets a particular method, or nothing (which initiates a plugin with default behavior):

```
$('#myModal').modal()                      // initialized with defaults
$('#myModal').modal({ keyboard: false })   // initialized with no keyboard
$('#myModal').modal('show')                // initializes and invokes show
immediately
```

Each plugin also exposes its raw constructor on a `Constructor` property: `$.fn.popover.Constructor`. If you'd like to get a particular plugin instance, retrieve it directly from an element: `$('[rel="popover"]').data('popover')`.

Asynchronous functions and transitions

All programmatic API methods are **asynchronous** and returns to the caller once the transition is started but **before it ends**.

In order to execute an action once the transition is complete, you can listen to the corresponding event.

```
$('#myCollapse').on('shown.bs.collapse', function (e) {
  // Action to execute once the collapsible area is expanded
})
```

In addition a method call on a **transitioning component will be ignored**.

```
$('#myCarousel').on('slid.bs.carousel', function (e) {
  $('#myCarousel').carousel('2') // Will slide to the slide 2 as soon as the
transition to slide 1 is finished
})

$('#myCarousel').carousel('1') // Will start sliding to the slide 1 and returns to
the caller
$('#myCarousel').carousel('2') // !! Will be ignored, as the transition to the
slide 1 is not finished !!
```

Default settings

You can change the default settings for a plugin by modifying the plugin's `Constructor.Default` object:

```
$.fn.modal.Constructor.Default.keyboard = false // changes default for the modal
plugin's `keyboard` option to false
```

No conflict

Sometimes it is necessary to use Bootstrap plugins with other UI frameworks. In these circumstances, namespace collisions can occasionally occur. If this happens, you may call `.noConflict` on the plugin you wish to revert the value of.

```
var bootstrapButton = $.fn.button.noConflict() // return $.fn.button to previously
assigned value
$.fn.bootstrapBtn = bootstrapButton            // give $().bootstrapBtn the
Bootstrap functionality
```

Version numbers

The version of each of Bootstrap's jQuery plugins can be accessed via the `VERSION`property of the plugin's constructor. For example, for the tooltip plugin:

```
$.fn.tooltip.Constructor.VERSION // => "4.0.0"
```

No special fallbacks when JavaScript is disabled

Bootstrap's plugins don't fall back particularly gracefully when JavaScript is disabled. If you care about the user experience in this case, use `<noscript>` to explain the situation (and how to re-enable JavaScript) to your users, and/or add your own custom fallbacks.

Third-party libraries

Bootstrap does not officially support third-party JavaScript libraries like Prototype or jQuery UI. Despite `.noConflict` and namespaced events, there may be compatibility problems that you need to fix on your own.

Util

All Bootstrap's JavaScript files depend on `util.js` and it has to be included alongside the other JavaScript files. If you're using the compiled (or minified) `bootstrap.js`, there is no need to include this—it's already there.

`util.js` includes utility functions and a basic helper for `transitionEnd` events as well as a CSS transition emulator. It's used by the other plugins to check for CSS transition support and to catch hanging transitions.

Theming Bootstrap

Customize Bootstrap 4 with our new built-in Sass variables for global style preferences for easy theming and component changes.

Introduction

In Bootstrap 3, theming was largely driven by variable overrides in LESS, custom CSS, and a separate theme stylesheet that we included in our `dist` files. With some effort, one could completely redesign the look of Bootstrap 3 without touching the core files. Bootstrap 4 provides a familiar, but slightly different approach.

Now, theming is accomplished by Sass variables, Sass maps, and custom CSS. There’s no more dedicated theme stylesheet; instead, you can enable the built-in theme to add gradients, shadows, and more.

Sass

Utilize our source Sass files to take advantage of variables, maps, mixins, and more.

File structure

Whenever possible, avoid modifying Bootstrap’s core files. For Sass, that means creating your own stylesheet that imports Bootstrap so you can modify and extend it. Assuming you’re using a package manager like npm, you’ll have a file structure that looks like this:

```
your-project/
├── scss
│   └── custom.scss
└── node_modules/
    └── bootstrap
        ├── js
        └── scss
```

If you’ve downloaded our source files and aren’t using a package manager, you’ll want to manually setup something similar to that structure, keeping Bootstrap’s source files separate from your own.

```
your-project/
├── scss
│   └── custom.scss
└── bootstrap/
    ├── js
    └── scss
```

Importing

In your `custom.scss`, you’ll import Bootstrap’s source Sass files. You have two options: include all of Bootstrap, or pick the parts you need. We encourage the latter, though be aware there are some requirements and dependencies across our components. You also will need to include some JavaScript for our plugins.

```
// Custom.scss
// Option A: Include all of Bootstrap

@import "node_modules/bootstrap/scss/bootstrap";
```

```
// Custom.scss
// Option B: Include parts of Bootstrap

// Required
@import "node_modules/bootstrap/scss/functions";
@import "node_modules/bootstrap/scss/variables";
@import "node_modules/bootstrap/scss/mixins";

// Optional
@import "node_modules/bootstrap/scss/reboot";
@import "node_modules/bootstrap/scss/type";
@import "node_modules/bootstrap/scss/images";
@import "node_modules/bootstrap/scss/code";
@import "node_modules/bootstrap/scss/grid";
```

With that setup in place, you can begin to modify any of the Sass variables and maps in your `custom.scss`. You can also start to add parts of Bootstrap under the `// Optional`section as needed. We suggest using the full import stack from our `bootstrap.scss` file as your starting point.

Variable defaults

Every Sass variable in Bootstrap 4 includes the `!default` flag allowing you to override the variable's default value in your own Sass without modifying Bootstrap's source code. and paste variables as needed, modify their values, and remove the `!default` flag. If a variable has already been assigned, then it won't be re-assigned by the default values in Bootstrap.

Variable overrides within the same Sass file can come before or after the default variables. However, when overriding across Sass files, your overrides must come before you import Bootstrap's Sass files.

Here's an example that changes the `background-color` and `color` for the `<body>` when importing and compiling Bootstrap via npm:

```
// Your variable overrides
$body-bg: #000;
$body-color: #111;

// Bootstrap and its default variables
@import "node_modules/bootstrap/scss/bootstrap";
```

Repeat as necessary for any variable in Bootstrap, including the global options below.

Maps and loops

Bootstrap 4 includes a handful of Sass maps, key value pairs that make it easier to generate families of related CSS. We use Sass maps for our colors, grid breakpoints, and more. Just like Sass variables, all Sass maps include the `!default` flag and can be overridden and extended.

Some of our Sass maps are merged into empty ones by default. This is done to allow easy expansion of a given Sass map, but comes at the cost of making *removing* items from a map slightly more difficult.

Modify map

To modify an existing color in our `$theme-colors` map, add the following to your custom Sass file:

```
$theme-colors: (
  "primary": #0074d9,
  "danger": #ff4136
);
```

Add to map

To add a new color to `$theme-colors`, add the new key and value:

```
$theme-colors: (
  "custom-color": #900
);
```

Remove from map

To remove colors from `$theme-colors`, or any other map, use `map-remove`:

```
$theme-colors: map-remove($theme-colors, "success", "info", "danger");
```

Required keys

Bootstrap assumes the presence of some specific keys within Sass maps as we used and extend these ourselves. As you customize the included maps, you may encounter errors where a specific Sass map's key is being used.

For example, we use the `primary`, `success`, and `danger` keys from `$theme-colors` for links, buttons, and form states. Replacing the values of these keys should present no issues, but removing them may cause Sass compilation issues. In these instances, you'll need to modify the Sass code that makes use of those values.

Functions

Bootstrap utilizes several Sass functions, but only a subset are applicable to general theming. We've included three functions for getting values from the color maps:

```
@function color($key: "blue") {
  @return map-get($colors, $key);
}

@function theme-color($key: "primary") {
  @return map-get($theme-colors, $key);
}

@function gray($key: "100") {
  @return map-get($grays, $key);
}
```

These allow you to pick one color from a Sass map much like how you'd use a color variable from v3.

```
.custom-element {
  color: gray("100");
  background-color: theme-color("dark");
}
```

We also have another function for getting a particular *level* of color from the `$theme-colors` map. Negative level values will lighten the color, while higher levels will darken.

```
@function theme-color-level($color-name: "primary", $level: 0) {
  $color: theme-color($color-name);
  $color-base: if($level > 0, #000, #fff);
  $level: abs($level);

  @return mix($color-base, $color, $level * $theme-color-interval);
}
```

In practice, you'd call the function and pass in two parameters: the name of the color from `$theme-colors` (e.g., primary or danger) and a numeric level.

```
.custom-element {
  color: theme-color-level(primary, -10);
}
```

Additional functions could be added in the future or your own custom Sass to create level functions for additional Sass maps, or even a generic one if you wanted to be more verbose.

Color contrast

One additional function we include in Bootstrap is the color contrast function, `color-yiq`. It utilizes the YIQ color space to automatically return a light (`#fff`) or dark (`#111`) contrast color based on the specified base color. This function is especially useful for mixins or loops where you're generating multiple classes.

For example, to generate color swatches from our `$theme-colors` map:

```
@each $color, $value in $theme-colors {
  .swatch-#{$color} {
    color: color-yiq($value);
  }
}
```

It can also be used for one-off contrast needs:

```
.custom-element {
  color: color-yiq(#000); // returns `color: #fff`
}
```

You can also specify a base color with our color map functions:

```
.custom-element {
  color: color-yiq(theme-color("dark")); // returns `color: #fff`
}
```

Sass options

Customize Bootstrap 4 with our built-in custom variables file and easily toggle global CSS preferences with new `$enable-*` Sass variables. Override a variable's value and recompile with `npm run test` as needed.

You can find and customize these variables for key global options in our `_variables.scss` file.

Variable	Values	Description
`$spacer`	`1rem` (default), or any value > 0	Specifies the default spacer value to programmatically generate our spacer utilities.
`$enable-rounded`	`true` (default) or `false`	Enables predefined `border-radius` styles on various components.
`$enable-shadows`	`true` or `false`(default)	Enables predefined `box-shadow` styles on various components.
`$enable-gradients`	`true` or `false`(default)	Enables predefined gradients via `background-image`styles on various components.
`$enable-transitions`	`true` (default) or `false`	Enables predefined `transition`s on various components.
`$enable-hover-media-query`	`true` or `false`(default)	**Deprecated**
`$enable-grid-classes`	`true` (default) or `false`	Enables the generation of CSS classes for the grid system (e.g., `.container`, `.row`, `.col-md-1`, etc.).
`$enable-caret`	`true` (default) or `false`	Enables pseudo element caret on `.dropdown-toggle`.
`$enable-print-styles`	`true` (default) or `false`	Enables styles for optimizing printing.

Color

Many of Bootstrap's various components and utilities are built through a series of colors defined in a Sass map. This map can be looped over in Sass to quickly generate a series of rulesets.

All colors

All colors available in Bootstrap 4, are available as Sass variables and a Sass map in our `scss/_variables.scss` file. This will be expanded upon in subsequent minor releases to add additional shades, much like the grayscale palette we already include.

Blue

Indigo

Purple

Pink

Red

Orange

Yellow

Green

Teal

Cyan

Here’s how you can use these in your Sass:

```scss
// With variable
.alpha { color: $purple; }

// From the Sass map with our `color()` function
.beta { color: color("purple"); }
```

Color utility classes are also available for setting `color` and `background-color`.

In the future, we’ll aim to provide Sass maps and variables for shades of each color as we’ve done with the grayscale colors below.

Theme colors

We use a subset of all colors to create a smaller color palette for generating color schemes, also available as Sass variables and a Sass map in our `scss/_variables.scss`file.

Primary

Secondary

Success

Danger

Warning

Info

Light

Dark

Grays

An expansive set of gray variables and a Sass map in `scss/_variables.scss` for consistent shades of gray across your project.

100

200

300

400

500
600
700
800
900

Within `_variables.scss`, you'll find our color variables and Sass map. Here's an example of the `$colors` Sass map:

```
$colors: (
  "blue": $blue,
  "indigo": $indigo,
  "purple": $purple,
  "pink": $pink,
  "red": $red,
  "orange": $orange,
  "yellow": $yellow,
  "green": $green,
  "teal": $teal,
  "cyan": $cyan,
  "white": $white,
  "gray": $gray-600,
  "gray-dark": $gray-800
) !default;
```

Add, remove, or modify values within the map to update how they're used in many other components. Unfortunately at this time, not *every* component utilizes this Sass map. Future updates will strive to improve upon this. Until then, plan on making use of the `${color}` variables and this Sass map.

Components

Many of Bootstrap's components and utilities are built with `@each` loops that iterate over a Sass map. This is especially helpful for generating variants of a component by our `$theme-colors` and creating responsive variants for each breakpoint. As you customize these Sass maps and recompile, you'll automatically see your changes reflected in these loops.

Modifiers

Many of Bootstrap's components are built with a base-modifier class approach. This means the bulk of the styling is contained to a base class (e.g., `.btn`) while style variations are confined to modifier classes (e.g., `.btn-danger`). These modifier classes are built from the `$theme-colors` map to make customizing the number and name of our modifier classes.

Here are two examples of how we loop over the `$theme-colors` map to generate modifiers to the `.alert` component and all our `.bg-*` background utilities.

```
// Generate alert modifier classes
```

```
@each $color, $value in $theme-colors {
  .alert-#{$color} {
    @include alert-variant(theme-color-level($color, -10), theme-color-
level($color, -9), theme-color-level($color, 6));
  }
}

// Generate `.bg-*` color utilities
@each $color, $value in $theme-colors {
  @include bg-variant('.bg-#{$color}', $value);
}
```

Responsive

These Sass loops aren't limited to color maps, either. You can also generate responsive variations of your components or utilities. Take for example our responsive text alignment utilities where we mix an `@each` loop for the `$grid-breakpoints` Sass map with a media query include.

```
@each $breakpoint in map-keys($grid-breakpoints) {
  @include media-breakpoint-up($breakpoint) {
    $infix: breakpoint-infix($breakpoint, $grid-breakpoints);

    .text#{$infix}-left   { text-align: left !important; }
    .text#{$infix}-right  { text-align: right !important; }
    .text#{$infix}-center { text-align: center !important; }
  }
}
```

Should you need to modify your `$grid-breakpoints`, your changes will apply to all the loops iterating over that map.

CSS variables

Bootstrap 4 includes around two dozen [CSS custom properties (variables)] in it's compiled CSS. These provide easy access to commonly used values like our theme colors, breakpoints, and primary font stacks when working in your browser's Inspector, a code sandbox, or general prototyping.

Available variables

Here are the variables we include (note that the `:root` is required). They're located in our `_root.scss` file.

```
:root {
  --blue: #007bff;
  --indigo: #6610f2;
  --purple: #6f42c1;
  --pink: #e83e8c;
  --red: #dc3545;
  --orange: #fd7e14;
  --yellow: #ffc107;
  --green: #28a745;
  --teal: #20c997;
  --cyan: #17a2b8;
  --white: #fff;
  --gray: #6c757d;
  --gray-dark: #343a40;
  --primary: #007bff;
  --secondary: #6c757d;
  --success: #28a745;
  --info: #17a2b8;
  --warning: #ffc107;
  --danger: #dc3545;
  --light: #f8f9fa;
  --dark: #343a40;
  --breakpoint-xs: 0;
  --breakpoint-sm: 576px;
  --breakpoint-md: 768px;
  --breakpoint-lg: 992px;
  --breakpoint-xl: 1200px;
  --font-family-sans-serif: -apple-system, BlinkMacSystemFont, "Segoe UI", Roboto,
"Helvetica Neue", Arial, sans-serif, "Apple Color Emoji", "Segoe UI Emoji", "Segoe
UI Symbol";
  --font-family-monospace: SFMono-Regular, Menlo, Monaco, Consolas, "Liberation
Mono", "Courier New", monospace;
}
```

Examples

CSS variables offer similar flexibility to Sass's variables, but without the need for compilation before being served to the browser. For example, here we're resetting our page's font and link styles with CSS variables.

```
body {
  font: 1rem/1.5 var(--font-family-sans-serif);
}
a {
  color: var(--blue);
}
```

You can also use our breakpoint variables in your media queries:

```
.content-secondary {
  display: none;
}

@media (min-width(var(--breakpoint-sm))) {
  .content-secondary {
    display: block;
  }
}
```

Build tools

Learn how to use Bootstrap's included npm scripts to build our documentation, compile source code, run tests, and more.

Tooling setup

Bootstrap uses NPM scripts for its build system. Our package.json includes convenient methods for working with the framework, including compiling code, running tests, and more.

To use our build system and run our documentation locally, you'll need a of Bootstrap's source files and Node. Follow these steps and you should be ready to rock:

1. Download and install Node.js, which we use to manage our dependencies.
2. Navigate to the root `/bootstrap` directory and run `npm install` to install our local dependencies listed in package.json.
3. Install Ruby, install Bundler with `gem install bundler`, and finally run `bundle install`. This will install all Ruby dependencies, such as Jekyll and plugins.
   - **Windows users:** Read this guide to get Jekyll up and running without problems.

When completed, you'll be able to run the various commands provided from the command line.

Using NPM scripts

Our package.json includes the following commands and tasks:

Task	Description
`npm run dist`	`npm run dist` creates the `/dist` directory with compiled files. **Uses Sass, Autoprefixer, and UglifyJS.**
`npm test`	Same as `npm run dist` plus it runs tests locally
`npm run docs`	Builds and lints CSS and JavaScript for docs. You can then run the documentation locally via `npm run docs-serve`.

Run `npm run` to see all the npm scripts.

Autoprefixer

Bootstrap uses Autoprefixer (included in our build process) to automatically add vendor prefixes to some CSS properties at build time. Doing so saves us time and code by allowing us to write key parts of our CSS a single time while eliminating the need for vendor mixins like those found in v3.

We maintain the list of browsers supported through Autoprefixer in a separate file within our GitHub repository. See /package.json for details.

Local documentation

Running our documentation locally requires the use of Jekyll, a decently flexible static site generator that provides us: basic includes, Markdown-based files, templates, and more. Here's how to get it started:

1. Run through the tooling setup above to install Jekyll (the site builder) and other Ruby dependencies with `bundle install`.

2. From the root `/bootstrap` directory, run `npm run docs-serve` in the command line.
3. Open `http://localhost:9001` in your browser, and voilà.

Learn more about using Jekyll by reading its documentation.

Troubleshooting

Should you encounter problems with installing dependencies, uninstall all previous dependency versions (global and local). Then, rerun `npm install`.

Webpack

Learn how to include Bootstrap in your project using Webpack 3.

Installing Bootstrap

Install bootstrap as a Node.js module using npm.

Importing JavaScript

Import Bootstrap's JavaScript by adding this line to your app's entry point (usually `index.js` or `app.js`):

```
import 'bootstrap';
```

Alternatively, you may **import plugins individually** as needed:

```
import 'bootstrap/js/dist/util';
import 'bootstrap/js/dist/dropdown';
...
```

Bootstrap is dependent on jQuery and Popper, these are defined as `peerDependencies`, this means that you will have to make sure to add both of them to your `package.json`using `npm install --save jquery popper.js`.

Notice that if you chose to **import plugins individually**, you must also install exports-loader

Importing Styles

Importing Precompiled Sass

To enjoy the full potential of Bootstrap and customize it to your needs, use the source files as a part of your project's bundling process.

First, create your own `_custom.scss` and use it to override the built-in custom variables. Then, use your main sass file to import your custom variables, followed by Bootstrap:

```
@import "custom";
@import "~bootstrap/scss/bootstrap";
```

For Bootstrap to compile, make sure you install and use the required loaders: sass-loader, postcss-loader with Autoprefixer. With minimal setup, your webpack config should include this rule or similar:

```
...
{
  test: /\.(scss)$/,
  use: [{
    loader: 'style-loader', // inject CSS to page
  }, {
    loader: 'css-loader', // translates CSS into CommonJS modules
  }, {
    loader: 'postcss-loader', // Run post css actions
    options: {
      plugins: function () { // post css plugins, can be exported to
postcss.config.js
        return [
          require('precss'),
          require('autoprefixer')
        ];
      }
    }
  }, {
    loader: 'sass-loader' // compiles Sass to CSS
  }]
},
...
```

Importing Compiled CSS

Alternatively, you may use Bootstrap's ready-to-use css by simply adding this line to your project's entry point:

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

In this case you may use your existing rule for `css` without any special modifications to webpack config except you don't need `sass-loader` just style-loader and css-loader.

```
...
module: {
  rules: [
    {
      test: /\.css$/,
      use: ['style-loader', 'css-loader']
    }
  ]
}
...
```

Accessibility

A brief overview of Bootstrap's features and limitations for the creation of accessible content.

Bootstrap provides an easy-to-use framework of ready-made styles, layout tools, and interactive components, allowing developers to create websites and applications that are visually appealing, functionally rich, and accessible out of the box.

Overview and Limitations

The overall accessibility of any project built with Bootstrap depends in large part on the author's markup, additional styling, and scripting they've included. However, provided that these have been implemented correctly, it should be perfectly possible to create websites and applications with Bootstrap that
fulfill WCAG 2.0 (A/AA/AAA), Section 508and similar accessibility standards and requirements.

Structural markup

Bootstrap's styling and layout can be applied to a wide range of markup structures. This documentation aims to provide developers with best practice examples to demonstrate the use of Bootstrap itself and illustrate appropriate semantic markup, including ways in which potential accessibility concerns can be addressed.

Interactive components

Bootstrap's interactive components—such as modal dialogs, dropdown menus and custom tooltips—are designed to work for touch, mouse and keyboard users. Through the use of relevant WAI-ARIA roles and attributes, these components should also be understandable and operable using assistive technologies (such as screen readers).

Because Bootstrap's components are purposely designed to be fairly generic, authors may need to include further ARIA roles and attributes, as well as JavaScript behavior, to more accurately convey the precise nature and functionality of their component. This is usually noted in the documentation.

Color contrast

Most colors that currently make up Bootstrap's default palette—used throughout the framework for things such as button variations, alert variations, form validation indicators—lead to *insufficient* color contrast (below the recommended WCAG 2.0 color contrast ratio of 4.5:1) when used against a light background. Authors will need to manually modify/extend these default colors to ensure adequate color contrast ratios.

Visually hidden content

Content which should be visually hidden, but remain accessible to assistive technologies such as screen readers, can be styled using the `.sr-only` class. This can be useful in situations where additional visual information or cues (such as meaning denoted through the use of color) need to also be conveyed to non-visual users.

```html
<p class="text-danger">
  <span class="sr-only">Danger: </span>
  This action is not reversible
</p>
```

For visually hidden interactive controls, such as traditional “skip” links, `.sr-only` can be combined with the `.sr-only-focusable` class. This will ensure that the control becomes visible once focused (for sighted keyboard users).

```html
<a class="sr-only sr-only-focusable" href="#content">Skip to main content</a>
```