sitepoint

#1

# Bootstrap

## A SitePoint Anthology

August 2016

# Table of Contents

# Bootstrap: A SitePoint Anthology

- Copyright © 2016 SitePoint Pty. Ltd.
- Editor: Simon Mackie
- Designer: Alex Walker

## Notice of Rights

## Notice of Liability

## Trademark Notice

- Published by SitePoint Pty. Ltd.
- 48 Cambridge Street Collingwood
- VIC Australia 3066
- Web: www.sitepoint.com
- Email: books@sitepoint.com

# The Rise of the Puss in Bootstrap

Once upon a time web developers had to build websites and web apps from scratch. They would take the mock-ups from the designers, painstakingly counting the pixels and crafting the CSS with lots of care and patience. Countless hours would be spent in an (often futile) attempt to obtain a pixel-perfect reproduction of the design concepts across the range of supported browsers.

Things started to change in August 2011, when a new hero rose to challenge the web. Born from the passion and dedication of Mark Otto and Jacob Thornton, the newborn Bootstrap took the front-end development kingdom by storm. The second version came fast, after only half a year and it was built along the principles of responsive web design. It boasted a robust twelve column grid, lots of pre-styled components and it allowed almost everyone to put together a basic webpage, just as easy as stacking together LEGO pieces. And just on its second birthday came the announcement of the evolution to version 3, featuring a mobile-first approach and ditching the previous Web 2.0 gradients in favor of the flat design. Two more years later, in August 2015, the first alpha version of the fourth iteration became available to the wide community. The growth trend seems to have come to a halt though, as at the time this words were written, no further public advancement happened.

I dare say that Bootstrap had almost the same effect on the World Wide Web as Ford's assembly line had for automotive industry. Every Joe and Jane with a minimum of knowledge in the field of front-end technologies could now put together a basic website that worked well across a large variety of devices and browsers. Sure, on one hand, most of them are similar to thousand others, to the point where many people started complaining that "Bootstrap is killing web design". On the other hand, there are true masterpieces of design and craftsmanship, that you can admire on dedicated websites such as the official Bootstrap Expo or BuiltWithBootstrap.

So here we are, almost 5 years from the birth of Bootstrap. In this time it grew from an internal framework at Twitter to the most used UI/front-end framework on the web, with more than 7 million websites using it (according to BuiltWith.com). Despite the apparent break in the development cycle, version 3.X.X of Bootstrap remains a solid option for building up your websites and web apps.

This is where this book comes in. Beginners will find a wealth of knowledge to help them start on this new and fascinating road. Advanced developers could also catch small tricks and tips, designed to make their life easier. One thing is sure though: Bootstrap is here to stay and we haven't seen the last of this great framework.

# Responsive Web Design Tips from Bootstrap CSS

By Syed Fazle Rahman

With the release of version 3, Bootstrap has gone mobile first, building on its already responsive base. What kinds of things does Bootstrap include in its CSS to help with this? Let's examine a few things and gain some insight that might help us in our own custom projects.

## Defining Proper Media Queries

Bootstrap has clearly defined breakpoints for different kinds of devices, specified by using CSS media queries. The following are the breakpoint categories used for the different types of devices:

1. **Extra Small Devices** (e.g. cell phones) are the default, creating the "mobile first" concept in Bootstrap. This covers devices smaller than 768px wide.
2. **Small Devices** (e.g. tablets) are targeted with `@media (min-width: 768px) and (max-width: 991px) { ... }`.
3. **Medium Sized Devices** (e.g. Desktops) have a screen size smaller than 1200px and greater than 991px, using `@media (min-width: 992px) and (max-width: 1199px) { ... }`.
4. **Larger Devices** (e.g. wide-screen monitors) are greater than 1200px, targeted using `@media (min-width: 1200px) { ... }`.

> ***Note**: Mobile devices that we use today come with 2 different screen orientations: **Portrait** and **Landscape**. So the above is mostly true for landscape view only. For example, if you are using a Samsung Galaxy Note III phone, the landscape view falls in the “Small Devices” category whereas the portrait view would fall under “Extra Small Devices”.*

This kind of categorization is common in responsive frameworks and it’s something you can certainly benefit from understanding better.

## The Grid System Demystified

If you are familiar with Bootstrap’s grid system, you might know that there is a specific HTML structure required to properly set up its grids. Let’s demystify it.

Let’s first have a look at Bootstrap’s HTML for a two-column setup:

```html
<div class="container">
  <div class="row">
    <div class="col-xs-6">
      <p>Column 1</p>
    </div>
    <div class="col-xs-6">
      <p>Column 2</p>
    </div>
  </div>
</div>
```

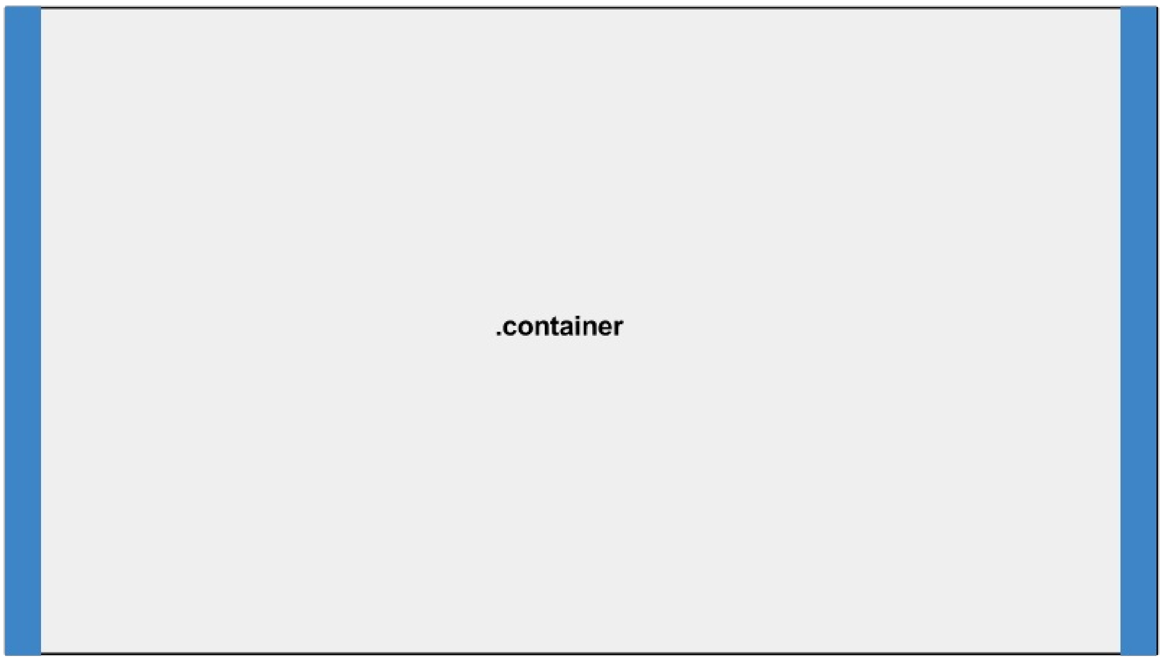
As shown above, Bootstrap’s grid system starts with a container element. Containers define how much space a grid system should use. They can be of two types: `.container` has different widths for different types of devices whereas `.container-fluid` expands to fit the width of the device.

With the help of media queries, Bootstrap gives different widths to the `.container` depending on the size of the device:

```
-   **Extra small devices (<768px)**: `width: auto` (or no width)
-   **Small Devices (≥768px)**: `width: 750px`
-   **Medium Devices (≥992px)**: `width: 970px`
-   **Larger Devices (≥1200px)**: `width: 1170px`
```

Here are some more CSS declarations that are applied to the `.container` class.

```css
.container {
  padding-right: 15px;
  padding-left: 15px;
  margin-right: auto;
  margin-left: auto;
}
```

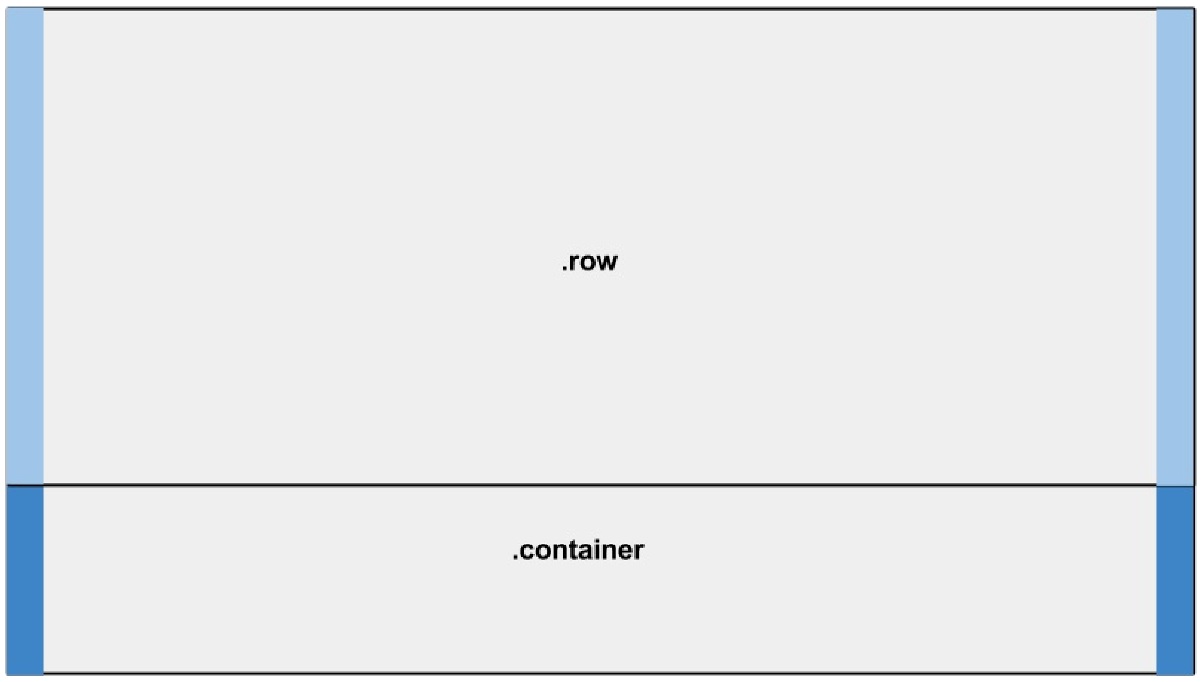


As seen in the above image, the `.container` prevents the content inside the element from touching the browser edge using 15px of padding on each side. It also ensures the container is centered using `auto` for left and right margins.

Rows are another important element in Bootstrap's Grid System. Before creating columns, you can define a row using the class `.row`. Here's a snippet from Bootstrap's CSS for the `.row` class:

```css
.row {
  margin-right: -15px;
  margin-left: -15px;
}
```

As shown above, our row has negative left and right margins of -15px to allow the row to touch the edge of its container element. This acts as a wrapper to hold columns, which can add up to 12 in number.

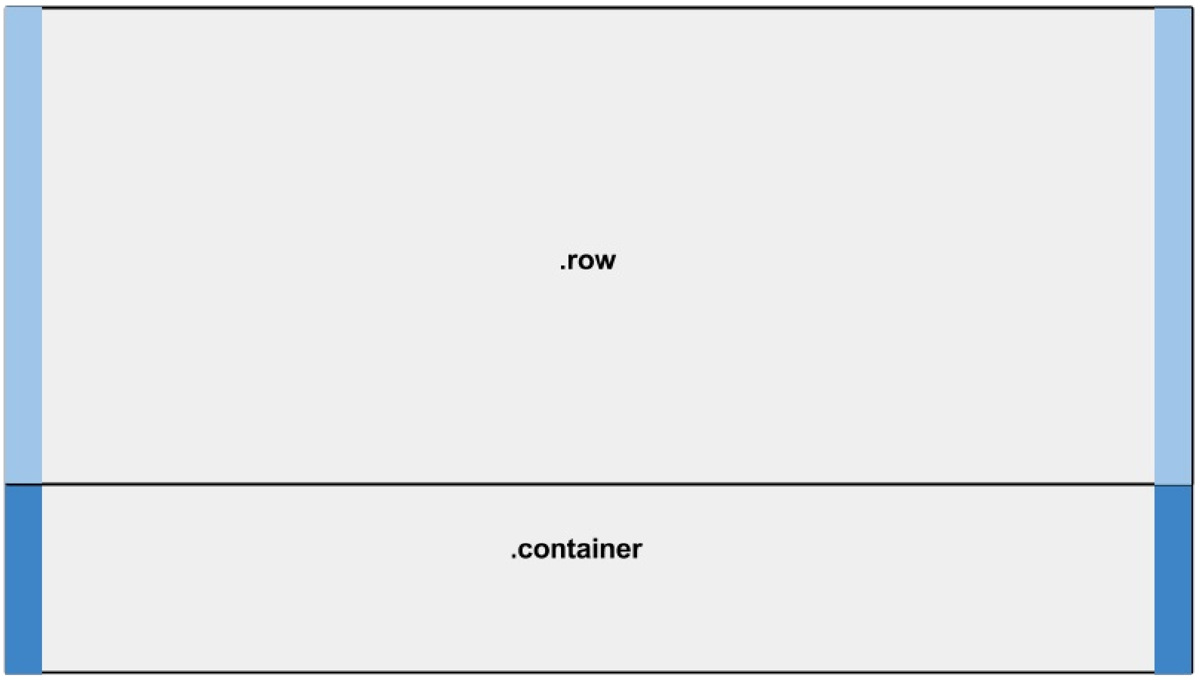


You may have noticed that the margins on the row seem to be counteracting the 15px of padding applied to the container. If we analyze the columns, we can see why this is needed.

Here’s a snippet from Bootstrap’s CSS for the `.col-xs-6` class.

```
.col-xs-6 {
  padding-right: 15px;
  padding-left: 15px;
}
```

As shown, left and right padding of 15px is applied to the columns, resulting in something like the image below:



Because of the negative margins on the row, the columns are touching the edges of the row and the edges of the container. But the padding causes the contents that go inside these columns to remain 15px away from the edges of the container.

Containers are used for multiple purposes, not just for the grid system, so the 15px padding helps to avoid the content touching the edges of the browser (when using `.container-fluid`). Rows have the negative margins so that they are not pushed by the padding of the container.

If you are considering designing your own framework, you might want to consider using this padding/margin technique.

## Defining Proper Column Widths

Bootstrap uses percentages (%) as the unit to define the widths of columns, helping with responsiveness. As stated above, there are 4 different categories of device-based breakpoints. Each category has its own set classes for columns of different sizes.

1. **Extra small devices** use `.col-xs-*` .
2. **Small devices** use `.col-sm-*` .
3. **Medium devices** use `.col-md-*` .
4. **Large devices** use `.col-lg-*` .

The asterisk character `*` gets replaced by a number. For example, `.col-xs-6` creates a column 6 times the size of a `.col-xs-1` column; `.col-sm-4` creates a column four times the size of `.col-sm-1` , and so on.

By default, all the columns have no width set, which defaults to `width: auto`. However, within the media queries, Bootstrap gives width values to each column class.

Here's a snippet from Bootstrap's CSS for the column classes with asterisks replacing the sizes for brevity (xs, sm, md, etc):

```
.col-*-12 { width: 100%; }
.col-*-11 { width: 91.66666667%; }
.col-*-10 { width: 83.33333333%; }
.col-*-9  { width: 75%; }
.col-*-8  { width: 66.66666667%; }
.col-*-7  { width: 58.33333333%; }
.col-*-6  { width: 50%; }
.col-*-5  { width: 41.66666667%; }
.col-*-4  { width: 33.33333333%; }
.col-*-3  { width: 25%; }
.col-*-2  { width: 16.66666667%; }
.col-*-1  { width: 8.33333333%; }
```

Let's analyze the above code. A class `.col-lg-6` will have a width of 50% in large devices but when viewed in medium, small, and extra-small devices, the default `width: auto` is used. This ensures that the columns are converted to a stacked layout (rather than side by side) in smaller devices.

## Responsive Tables

Tables, used for displaying tabular data, are also responsive in Bootstrap. To use Bootstrap's table styles, we use the class `.table`, which has the following CSS:

```
.table {
  width: 100%;
  max-width: 100%;
  margin-bottom: 20px;
}
```

Bootstrap forces tables to fit the width of the parent element by applying a width of 100%. But this has an issue. A multi-column table will get squeezed on smaller devices and may not be readable.

Bootstrap has another class to remedy this: `.table-responsive`. Here's the CSS:

```
.table-responsive {
  width: 100%;
  overflow-x: auto;
  overflow-y: hidden;
  -webkit-overflow-scrolling: touch;
  -ms-overflow-style: -ms-autohiding-scrollbar;
  border: 1px solid #ddd;
}
```

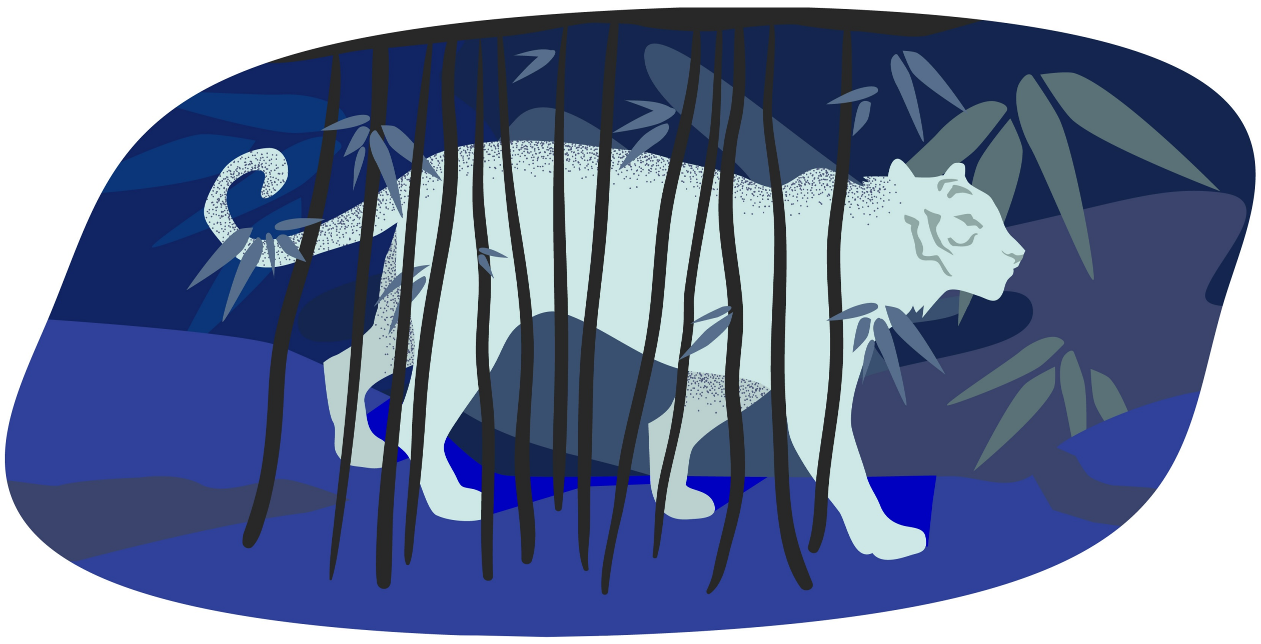
These styles cause the table to become scrollable on the horizontal axis on smaller devices.

## Responsive Images

Working with larger images may be a problem for smaller devices. Bootstrap uses a class of `.img-responsive` to make any image responsive:

```
.img-responsive {
  display: block;
  max-width: 100%;
  height: auto;
}
```

This combination of `max-width: 100%` and `height: auto` will ensure the images scale down proportionally in smaller devices, while staying within the parent element's constraints on larger devices.



# Understanding the Bootstrap Grid System

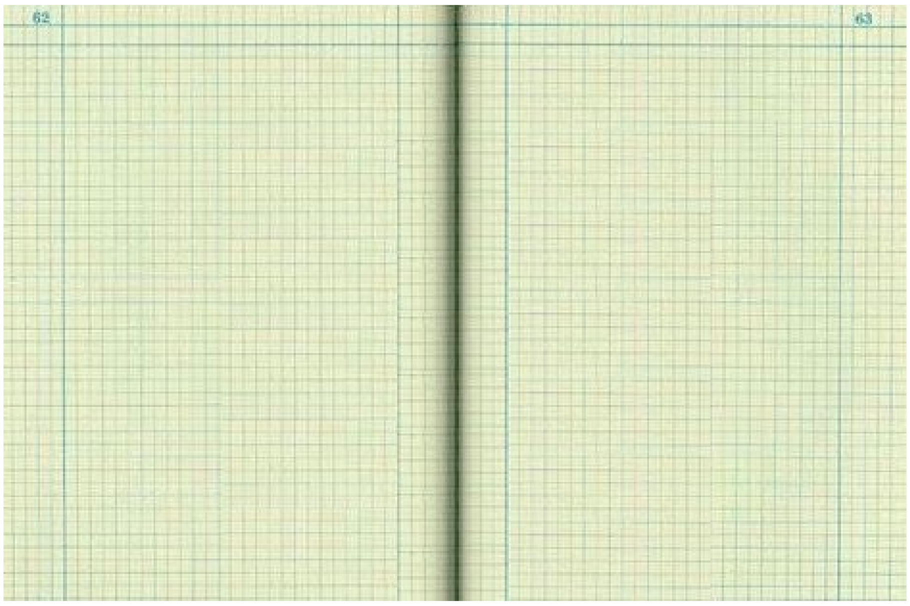
By Syed Fazle Rahman

Bootstrap is undoubtedly one of the most popular front-end frameworks. With more than 73k stars and 27k forks, Bootstrap is also one of the most popular GitHub repositories. In my last article, Responsive Web Design Tips from Bootstrap’s CSS, I explained how Bootstrap functions as a responsive framework. In this article, we will discuss a related topic: **The Grid System**, one of the most important concepts in Bootstrap.

## What is the Bootstrap Grid System?

Like any grid system, the Bootstrap grid is a library of HTML/CSS components that allow you to structure a website and place a website’s content in desired locations easily.

Think of graph paper, where every page has a set of vertical and horizontal lines. When these lines intersect, we get squares or rectangular spaces.



*By Sfoerster (Own work) CC-BY-SA-3.0, via Wikimedia Commons*

Well, this is also true for Bootstrap's Grid System. It allows you to create rows and columns and then place content in the "intersected" areas.

Now the question is, how many rows and columns you can create using Bootstrap's Grid System? Bootstrap allows you to create up to **12 columns** and **unlimited rows** — hence the name **12-Grid System**. So, let's see how we can utilize this grid system to create various types of layouts.

## Getting started with Bootstrap's Grid System

To get started, naturally, you'll need to have the necessary assets in your page to get Bootstrap working. If you're new to Bootstrap, you can refer to our previous article Getting started with Bootstrap or my book Jump Start Bootstrap, to dig deeper.

Bootstrap's Grid System is made up of 3 things:

1. A container
2. Rows
3. Columns

Let's explore each of the above in detail.

## Creating a Container

Bootstrap's grid system needs a container to hold rows and columns. A container is a simple `<div>` element with a class of `.container`. The container is used to provide a proper width for the layout, acting as a wrapper for the content. Here is the code for this task:

```
<div class="container">
  Some content
</div>
```

Here the container element wraps the content and sets left and right margins. It also has different fixed widths in different sized devices. Have a look at the following table:

| Device Width | Container Width |
|---|---|
| 1200px or higher | 1170px |
| 992px to 1199px | 970px |
| 768px to 991px | 750px |
| Less than 768px | auto |

You can choose a fluid container if you are not fond of a fixed layout. To do this, you use the class `.container-fluid`. A fluid container has no fixed width; its width will always be the width of the device.

Just note that both fixed and fluid containers have padding of 15px on the left and right sides.

## Creating a Row

A row acts like a wrapper around the columns. The row nullifies the padding set by the container element by using a negative margin value of -15px on both the left and right sides.

A row spans from the left edge to the right edge of the container element. It is created by adding the class `.row` to a block level element inside the container.

Have a look at the following CodePen:

```
<div class="container">
  <div class="row">
    Some content
  </div>
</div>
```

In this demo, you can see the text touching the left edge of the container element. This is because the container's padding has been removed by the row due to the negative margins on the row.

Finally, there's no limit on the number of rows you can create.

## Creating Columns

Bootstrap uses different column class prefixes for different sized devices. These prefixes are shown in the table below:

| Class Prefix | Device Size |
|---|---|
| .col-xs- | < 768px |
| .col-sm- | 768px to 991px |
| .col-md- | 992px to 1199px |
| .col-lg- | ≥ 1200px |

So, let's create our first Bootstrap column:

```
<div class="container">
  <div class="row">
    <div class="col-xs-12">
      Some content
    </div>
  </div>
</div>
```

In the above demo, I used the class `.col-xs-12` to create a single column that spans across 12 virtual Bootstrap columns. Hence, this column's width will be the width of the row.

In the above demo, you will also see the 15px padding reappear to push the element away from the container. This is because every column in Bootstrap has a padding of 15px.

You must be wondering why I used the class prefix that belonged to extra smaller devices, which is `.col-xs-`. In Bootstrap, if a column is defined for a particular type of device then it is guaranteed to behave similarly in larger devices as well. Therefore, **a column defined for extra smaller devices will work in all types of devices**.

Let's now create a 2-column layout for smaller devices and check out its behaviour in larger devices and extra-small devices. We will use the class prefix `.col-sm-` here. To create 2 columns of equal widths, we should assign 6 virtual Bootstrap columns to each one of them. This way, we maintain the limit of 12 virtual Bootstrap columns for a single row.

```
<div class="container">
  <div class="row">
    <div class="col-sm-6">
      <h1>Bootstrap Grid Demo</h1>
    </div>
    <div class="col-sm-6 other">
      <h1>2 Columns</h1>
    </div>
  </div>
</div>
```

## Nesting with the Grid System

Nesting is one of the ways to create complex designs using Bootstrap's grid system. It is also the one section where many first-timers have trouble.

We understand that to use Bootstrap's grid system, we need 3 things: A container, rows, and columns. So to nest a grid system within a column we will need the same three things. But the only difference is that **the container is already defined**. In this case, the columns will behave as the containers for the nested grid system.

Here's the logic: The containers provide 15px of padding, which is nullified by the row. Then we define columns that again have 15px of padding on the left and right side. So, to nest a grid system within a column, we simply need rows and columns. No `.container` or `.container-fluid` elements are necessary for a nested grid system.

Here's an example of a nested grid system:

Bootstrap grid demo with nested columns

## What About More than 12 Columns?

This is one of the root causes for disordered Bootstrap layouts. A wrong calculation in deciding the number of virtual Bootstrap columns can lead to an improper layout.

In such a case, a virtual row will be created and unfitted columns will shift to the next row. For example, if you have defined 2 columns with the classes `.col-md-8` and `.col-md-5`, the second column will shift to a new row because it requires 5 virtual Bootstrap columns whereas only 4 are left.

## Helper Classes

Bootstrap provides various helper classes that can be useful in certain situations in dealing with grids. These classes are:

- `.clearfix`: Normally used to clear floats, adding this class to any column will make it shift to a new row automatically, to help you correct problems that occur with uneven column heights.
- **Offsetting columns**: You don't have to occupy all 12 of the virtual columns. You can use offset classes like `.col-xs-offset-*` or `.col-md-offset-*` to leave a particular number of virtual Bootstrap columns to the left of any column (kind of like invisible place holders).
- **Reordering**: Use classes like `.col-md-push-*` and `.col-md-pull-*` to shift a column to the right or left, respectively.

# Understanding Bootstrap Modals

By Syed Fazle Rahman

Everyone who spends a bit of time and effort researching the Bootstrap framework can understand how easy it is to use for novice designers. It ships with some of the best ready-to-use JavaScript and jQuery components and plugins. Here we will be talking about one of the most useful jQuery Bootstrap plugins - **The Modal**.

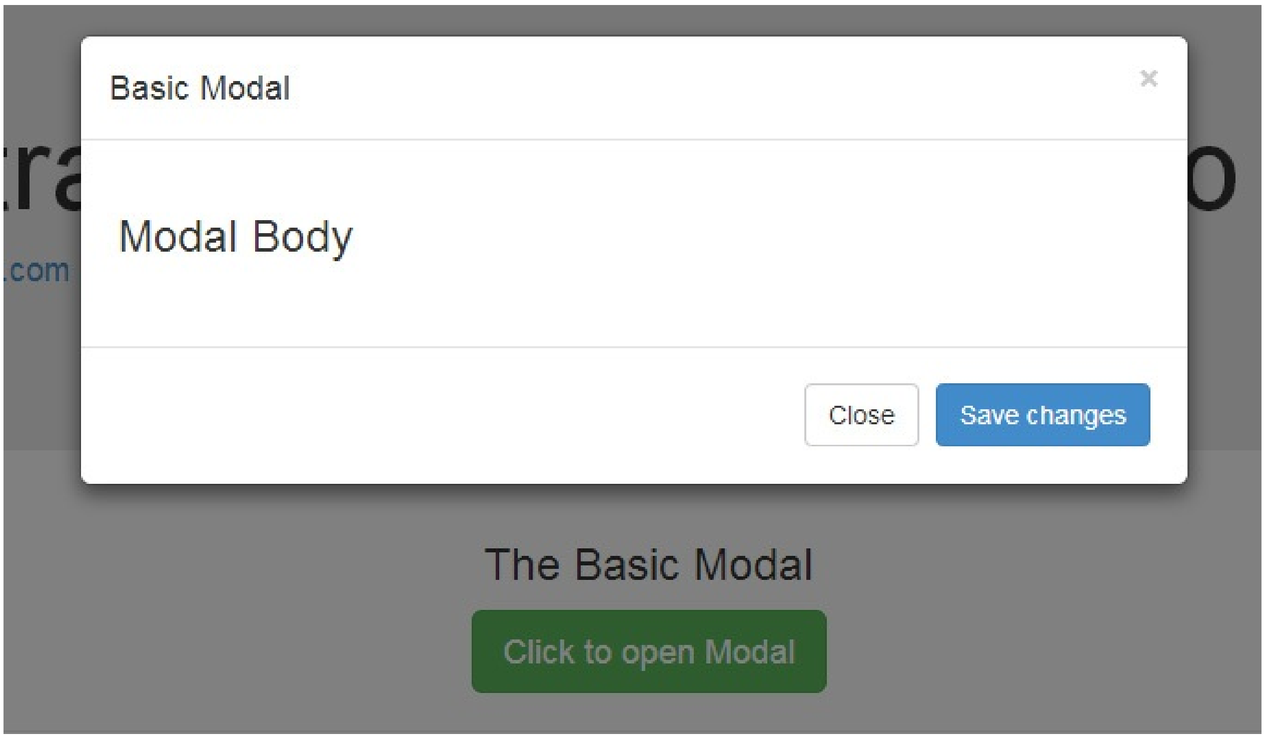
**The Bootstrap Modal** is a lightweight multi-purpose JavaScript popup that is customizable and responsive. It can be used to display alert popups, videos, and images in a website. Websites built with Bootstrap can use the modal to showcase (for example) terms and conditions (as part of a signup process), videos (similar to a standard light box), or even social media widgets.

Now let’s examine the different parts of Bootstrap’s modal, so we can understand it better.

The Bootstrap Modal is divided into three primary sections: the header, body, and footer. Each has its own significance and hence should be used properly. We’ll discuss these shortly. The most exciting thing about Bootstrap’s modal? You don’t have to write a single line of JavaScript to use it! All the code and styles are predefined by Bootstrap. All that’s required is that you use the proper markup and the attributes to trigger it.

## The Default Modal

The default Bootstrap Modal looks like this:



To trigger the modal, you'll need to include a link or a button. The markup for the trigger element might look like this:

```
<a href="#" class="btn btn-lg btn-success" data-toggle="modal" data-target="#basicModa
l">Click to open Modal</a>
```

Notice the link element has two custom data attributes: `data-toggle` and `data-target`. The toggle tells Bootstrap what to do and the target tells Bootstrap which element is going to open. So whenever a link like that is clicked, a modal with an id of "basicModal" will appear.

Now let's see the code required to define the modal itself. Here is the markup:

```
<div class="modal fade" id="basicModal" tabindex="-1" role="dialog" aria-labelledby="b
asicModal" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&a
mp;times;</button>
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">
        <h3>Modal Body</h3>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</butt
on>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div>
  </div>
</div>
```

The parent div of the modal should have the same id as used in the trigger element above. In our case it would be `id="basicModal"`.

> **Note**: Custom attributes like `aria-labelledby` and `aria-hidden` in the parent modal element are used for accessibility. It is a good practice to make your website accessible to all, so you should include these attributes since they won't negatively affect the standard functionality of the modal.

In the modal's HTML, we can see a wrapper div nested inside the parent modal div. This div has a class of `modal-content` that tells bootstrap.js where to look for the contents of the modal. Inside this div, we need place the three sections I mentioned earlier: the header, body, and footer.

The modal header, as the name implies, is used to give the modal a title and some other elements like the "x" close button. This should have a `data-dismiss` attribute that tells Bootstrap which element to hide.

Then we have the modal body, a sibling div of the modal header. Consider the body an open canvas to play with. You can add any kind of data inside the body, including a YouTube video embed, an image, or just about anything else.

Lastly, we have the modal footer. This area is by default right aligned. In this area you could place action buttons like "Save", "Close", "Accept", etc., that are associated with the action the modal is displaying.

Now we are done with our first modal! It should look like in the image above.

## Changing the Modal’s Size

Earlier I mentioned that the Bootstrap modal is responsive and flexible. We will see how to change its size in this section.

The modal comes in two new flavors in Bootstrap 3: Large and Small. Add a modifier class `modal-lg` to the `modal-dialog` div for a larger modal or `modal-sm` for a smaller modal.

## Activating the Modal with jQuery

The modal is a jQuery plugin, so if you want to control the modal using jQuery, then you need to call the `.modal()` function on the modal’s selector. For Example:

```
$('#basicModal').modal(options);
```

The “options” here would be a JavaScript object that can be passed to customize the behaviour. For example:

```
var options = {
    "backdrop" : "static"
}
```

Available options include:

- **backdrop**: This can be either `true` or `static`. This defines whether or not you want the user to be able to close the modal by clicking the background.
- **keyboard**: if set to `true` then the modal will close via the ESC key.
- **show**: Used for opening and closing the modal. It can be either `true` or `false`.
- **remote**: This is one of the coolest options. It can be used to load remote content using jQuery’s `load()` method. You need to specify an external page in this option. It is set to `false` by default.

## Bootstrap Modal’s Events

You can further customize the normal behaviour of the Bootstrap modal by using various events that are triggered while opening and closing the modal. These events have to be bound using jQuery’s `.on()` method.

Various events available are:

- **show.bs.modal**: fired just before the modal is open.

- **shown.bs.modal**: fired after the modal is shown.
- **hide.bs.modal**: fired just before the modal is hidden.
- **hidden.bs.modal**: fired after the modal is closed.
- **loaded.bs.modal**: fired when remote content is successfully loaded in the modal's content area using the `remote` option mentioned above.

You can use one of the above events like this:

```
$('#basicModal').on('shown.bs.modal', function (e) {
    alert('Modal is successfully shown!');
});
```

## Loading Remote Content in the Modal

There are three different ways to load remote content inside a Bootstrap modal.

The first way is by using the `remote` option inside the `options` object, as mentioned above. The other two ways are done without JavaScript, as shown below.

You can provide a value to the `href` attribute inside the modal's trigger element. In our case, the trigger is a link. For example, instead of the `#` value we included earlier, we can include a URL to a specific page:

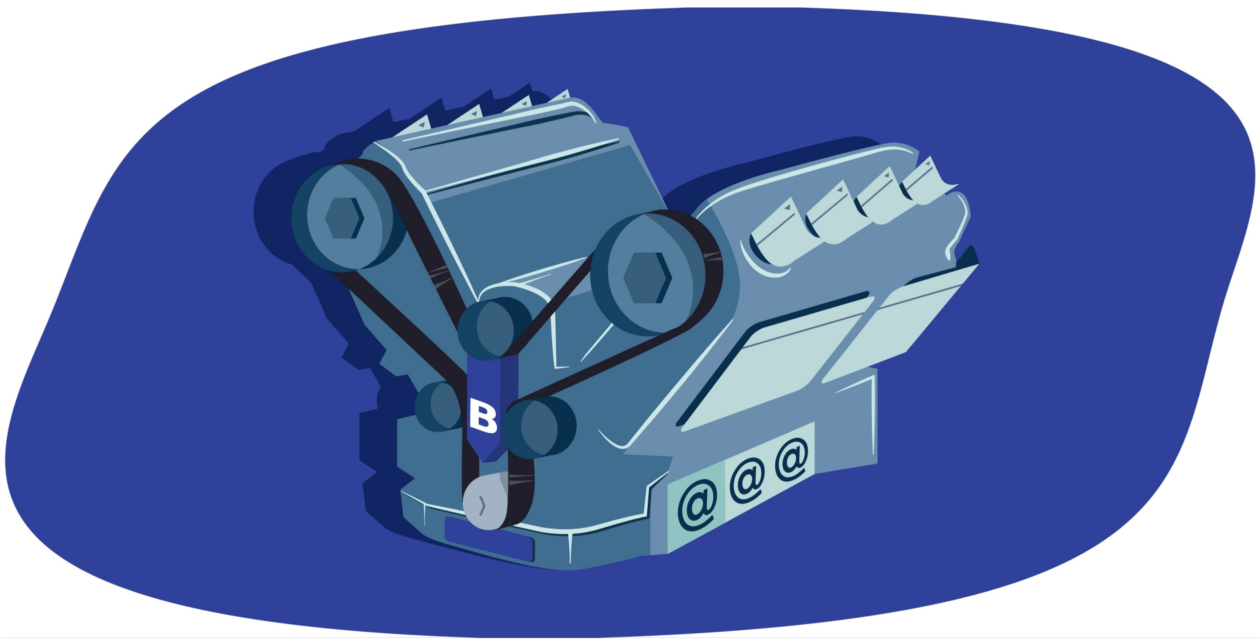
```
<a class="btn btn-lg btn-default"
   data-toggle="modal"
   data-target="#largeModal"
   href="remote-page.html">Click to open Modal</a>
```

You can also provide a custom data attribute of `data-remote` to the trigger element instead of using the `href` attribute. For example:

```
<a class="btn btn-lg btn-default" data-toggle="modal"
   data-target="#largeModal"
   data-remote="remote-page.html">Click to open Modal</a>
```

## Conclusion

The modal is one of the best plugins offered by Bootstrap 3. For a novice designer, it is one of the best ways to load content inside a popup screen without writing any JavaScript.



# Bootstrap JavaScript Components

By Syed Fazle Rahman

Bootstrap happens to be the easiest and the best CSS framework on the Internet today. It allows developers with no CSS knowledge to build basic templates without any efforts. But this doesn't stop designers from using Bootstrap. Bootstrap has one of the best sets of powerful JavaScript components. These components are easy to use and are usable in your web project, today. Here I will discuss some of the best Bootstrap JavaScript components and how to use them.

Let's get started!

The first thing that we should understand is that Bootstrap’s JavaScript components are written in jQuery. So we need jQuery to work with them. After you have downloaded Bootstrap 3, copy the contents of the `dist` folder and paste it inside a new work-space. You must be thinking why do we need the CSS and fonts folder when we are going to learn JavaScript? There are many Bootstrap JavaScript components that depends on CSS to work properly.

So unless you include the Bootstrap CSS it won’t function properly. Bootstrap 3 also allows us to use each module individually instead of downloading all the JavaScript components. We will see at the end of this tutorial how to use a single module instead of including all the components. Some of the main Bootstrap JavaScript components explained in this tutorial are:

1. Modal

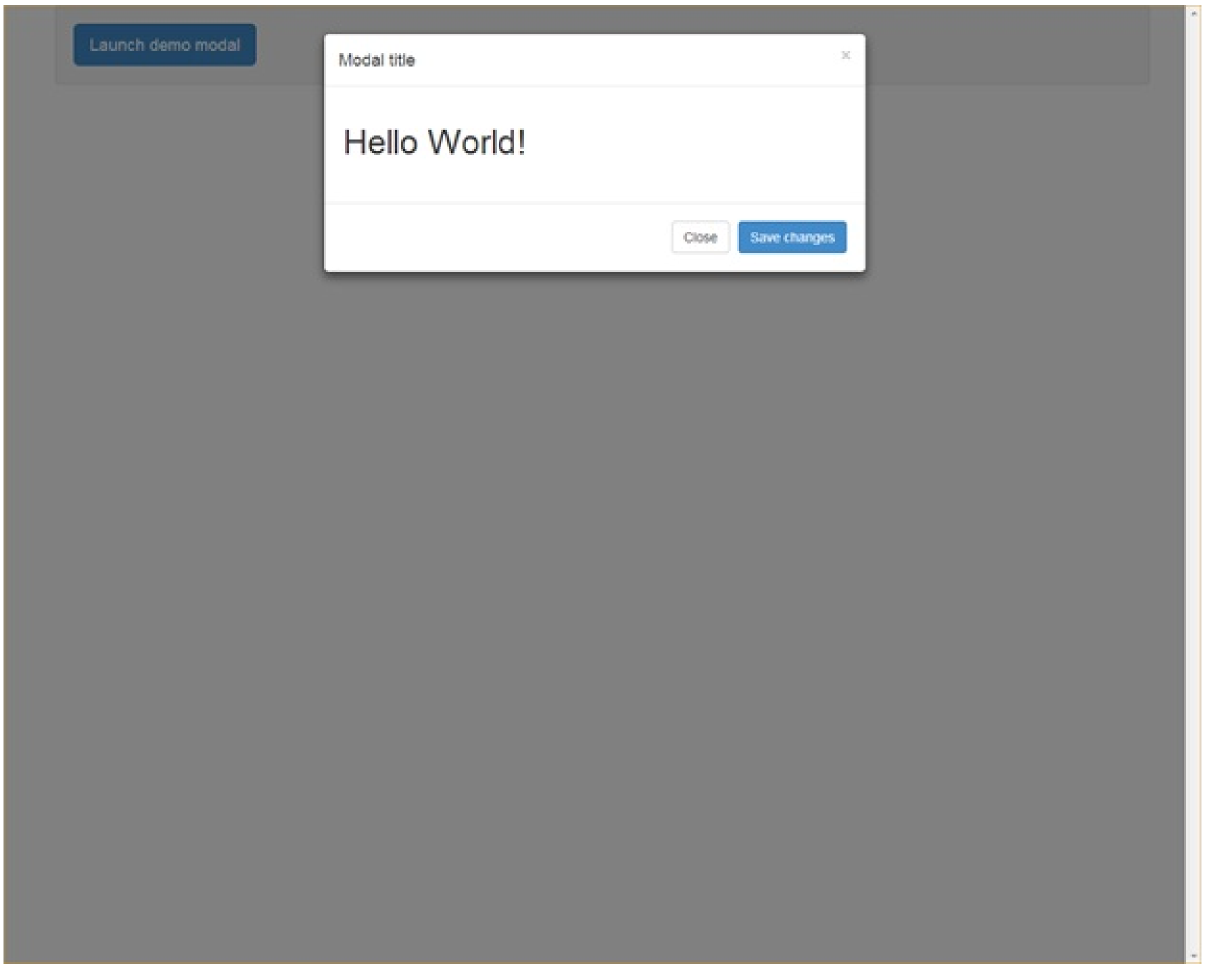
2. Dropdown
3. ScrollSpy
4. Tab
5. Tooltip
6. Popover
7. Alert

We will cover each of them in this tutorial. We will also experiment a bit with each component so that we get a customized Bootstrap 3 JavaScript component.

## Modal

A modal is a dialog prompt just like a traditional alert. It comes with advanced features like modal title, modal body, modal footer, close button and a close symbol at the top right corner. It can be used as a confirmation window in many applications such as before making a payment, or deleting an account, etc. A Modal has three sections: header, body and footer. You can decide what to place in each of them.

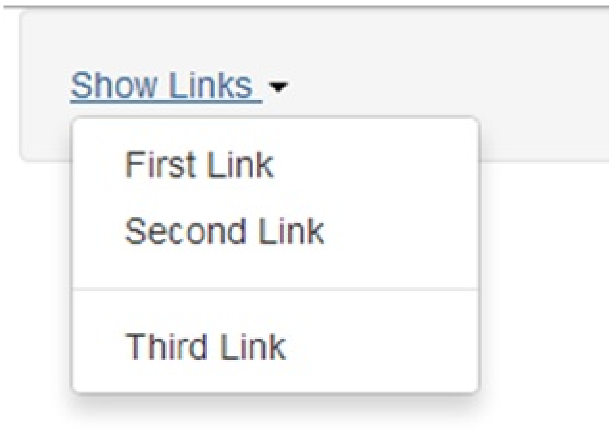
```
<!-- Button trigger modal -->
<button class="btn btn-primary btn-lg" data-toggle="modal" data-target="#myModal">
  Launch demo modal
</button>

<!-- Modal -->
<div class="modal fade" id="myModal" tabindex="-1" role="dialog" aria-labelledby="myMo
dalLabel" aria-hidden="true">
  <div class="modal-dialog">
    <div class="modal-content">
      <div class="modal-header">
        <button type="button" class="close" data-dismiss="modal" aria-hidden="true">&t
imes;</button>
        <h4 class="modal-title" id="myModalLabel">Modal title</h4>
      </div>
      <div class="modal-body">
        <h1>Hello World!</h1>
      </div>
      <div class="modal-footer">
        <button type="button" class="btn btn-default" data-dismiss="modal">Close</butt
on>
        <button type="button" class="btn btn-primary">Save changes</button>
      </div>
    </div><!-- /.modal-content -->
  </div><!-- /.modal-dialog -->
</div><!-- /.modal -->
```



## DropDown

Creating a drop down menu in Bootstrap 3 gets extremely easy. You just have to understand right markup required. You can use this DropDown in a navigation bar or inside any div you wish.

```
<div class="dropdown">
  <a data-toggle="dropdown" href="#">Show Links <b class="caret"></b></a>
  <ul class="dropdown-menu" role="menu">
    <li><a href="#">First Link</a></li>
    <li><a href="#">Second Link</a></li>
    <li role="presentation" class="divider"></li>
    <li><a href="#">Third Link</a></li>
  </ul>
</div>
```

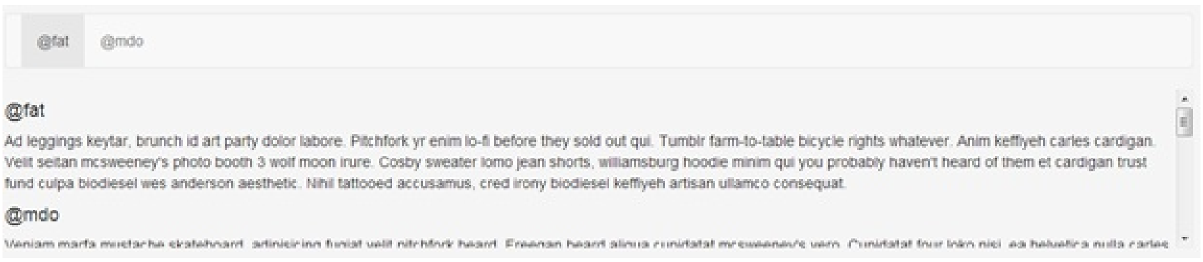


First, you have to give the class "dropdown" to any parent element that you want to treat as a drop down element. In my case, I have used a div element. You can even make an li element as "dropdown". Then you have to place an `<a>` tag immediately inside the dropdown element. Add a new attribute "data-toggle" to the link tag and give the value as "dropdown". Finally add a ul list below the link tag. You have to add class as "dropdown-menu" to the ul tag. To add a separator between li elements, add a new emplty li element with class as "divider" to the list. if you are not comfortable with the `data-*` attributes then you can even trigger drop down using jQuery. Give a unique id to the link element and call the dropdown method as below:

```
$('#myDropDown').dropdown();
```

## ScrollSpy

ScrollSpy is an interesting JavaScript module added to the Bootstrap library. It is basically a combination of navigation menu and contents below. Its role is to update the active item in the navigation bar as you scroll down the content area. To use the ScrollSpy feature you have to add `data-spy="scroll"` and `data-target="#top-navigation"` attribute to the body element. Here `#top-navigation` is the id of my navigation bar. Make sure the links in the navigation bar are internal links.



@fat

Ad leggings keytar, brunch id art party dolor labore. Pitchfork yr enim lo-fi before they sold out qui. Tumblr farm-to-table bicycle rights whatever. Anim keffiyeh carles cardigan. Velit seitan mcsweeney's photo booth 3 wolf moon irure. Cosby sweater lomo jean shorts, williamsburg hoodie minim qui you probably haven't heard of them et cardigan trust fund culpa biodiesel wes anderson aesthetic. Nihil tattooed accusamus, cred irony biodiesel keffiyeh artisan ullamco consequat.

@mdo

Veniam marfa mustache skateboard, adipisicing fugiat velit pitchfork beard. Freegan beard aliqua cupidatat mcsweeney's vero. Cupidatat four loko nisi, ea helvetica nulla carles

## Tabs

Bootstrap 3's tabs take inspiration from traditional jQuery tabs. They both look and function alike. To use Bootstrap Tabs you need to define two separate sections: the tabs navigation and tab areas. The markup goes like below:

```
<!-- Nav tabs -->
<ul class="nav nav-tabs">
  <li class="active"><a href="#home" data-toggle="tab">Home</a></li>
  <li><a href="#profile" data-toggle="tab">Profile</a></li>
  <li><a href="#messages" data-toggle="tab">Messages</a></li>
  <li><a href="#settings" data-toggle="tab">Settings</a></li>
</ul>

<!-- Tab panes -->
<div class="tab-content">
  <div class="tab-pane active" id="home">...</div>
  <div class="tab-pane" id="profile">...</div>
  <div class="tab-pane" id="messages">...</div>
  <div class="tab-pane" id="settings">...</div>
</div>
```

The navigation is created using a ul element with the class "nav-tabs" while the additional class `nav` is used to apply the navigation CSS style. Each li element is composed of an internal link that should define the attribute `data-toggle` as "tab". This triggers Bootstrap's Tabs JavaScript and the respective tab area is displayed. Coming to the tabs area, it consists of a set of div elements. The parent div should have a class as `tab-content` and the child divs should have a class `tab-pane`. Each tab-pane must have an id corresponding to the internal links defined in the tabs navigation. In the above example, I have set a class of the first tab-pane as active. This makes it visible by default.

## ToolTip

ToolTip is an extremely useful JavaScript plugin provided by Bootstrap 3. It helps in showing help texts on any HTML element. It's cross-browser compatible, too! To use ToolTip, the markup goes like this:

```
<button id="myButton" type="button" class="btn btn-default"
    data-toggle="tooltip" data-placement="left" title=""
    data-original-title="Tooltip on left">Tooltip on left</button>
```

The above markup displays a button with the tooltip feature. Attribute `data-toggle` is used by Bootstrap to identify on which element it has to display the tooltip. Attribute `data-original` is used to define what goes inside the tooltip. Attribute `data-placement` is used to help bootstrap where to show the tooltip. For performance reasons, Bootstrap will not initialize the ToolTip and Popover components by default. You have to initialize them manually by using the following jQuery:

```
$('#myButton').tooltip();
```

## Popovers

If you have ever been a hardcore iBook reader, then you would understand what popovers are. They are the extended version of ToolTip with some more functionalities. You can display more HTML elements like img tags, links, additional divs, etc inside Popovers.

```
<button type="button" class="btn btn-default"
    data-toggle="popover" data-placement="left"
    data-content="Vivamus sagittis lacus vel augue laoreet rutrum faucibus."
    data-original-title="" title="">Popover on left </button\>
```

The HTML snippet displays a button with popover functionality. It also has set of custom `data-*` attributes that you must necessarily understand. Attribute `data-toggle` identifies which element must control the popover. Attribute `data-content` contains the data that should be displayed inside the popover. Attribute `data-placement` tells on which side should the popover appear. In the above case, the data must be plain text only. If you want to display HTML content inside the popover, then you have to add the additional attribute `data-html` as true. The HTML data-content must go inside the double quotes with escaped characters wherever necessary. The markup for the HTML data contents inside the popover should be like below:

```
<button id="myPopover" type="button" class="btn btn-default"
    data-toggle="popover" data-placement="left" data-html="true"
    data-content="<a href=\"http://www.google.com\">Go to google</a>" data-original-ti
tle="" title="">
    Popover on left
</button>
```

Use the below jQuery to initialize popovers:

```
$('#myButton').popover();
```

## Alerts

Alerts in Bootstrap are not like window popups. They are a set of divs with predefined background colors and a dismiss button. The markup goes like below:

```
<div class="alert alert-warning fade in">
    <button type="button" class="close" data-dismiss="alert" aria-hidden="true">&times;
</button>
    <strong>Holy guacamole!</strong> Best check yo self, you're not looking too good.
</div>
```

The above alert has a pale yellow background, since it is a warning message. You can change the color to red by changing the class of the alert to `alert-danger`. Every alert div must have a close button with a set of `data-*` attributes as defined above. Attribute `data-dismiss` hides the alert div when clicked.

## Conclusion

You might now have a better understanding of how Bootstrap helps us using JavaScript components without writing a single line of jQuery in our code. These JavaScript components are one of the main reasons why the Bootstrap framework is so popular in the web today.

# Less: Beyond Basics with the Bootstrap Mixins Library

By Maria Antonietta Perna

Preprocessors like Less, Sass, Stylus, etc., extend the capabilities of CSS by including such programming language features as variables, functions, mathematical operations, etc. Appropriate use of this technology aims to ensure maintainable stylesheet documents and an improved workflow.

This article takes a step beyond the basics of the Less preprocessor language by using some of the Bootstrap 3 Less code as both a learning and development tool.

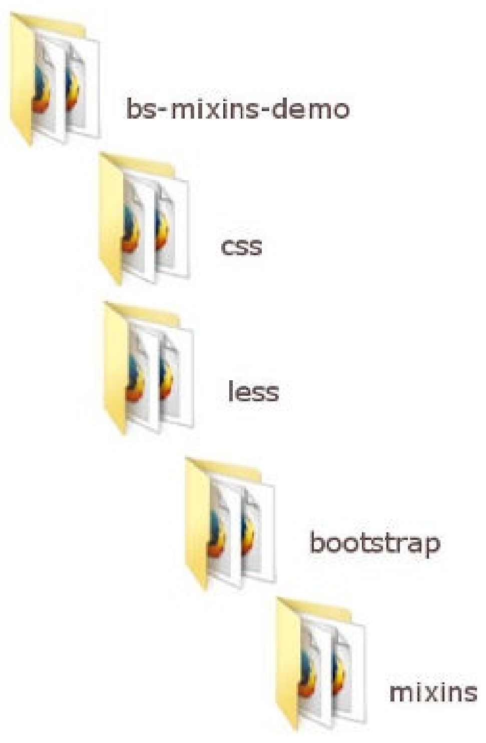
If you're just starting out, enjoy Speed Up Your Web Development Process with Less, a clear and concise video introduction by Sandy Ludosky.

## The Demo Page Set Up

The Less features discussed here are all implemented in this compiled CodePen page and as this zip file so that you can freely see how the page looks, check the code details for yourself and experiment with it.

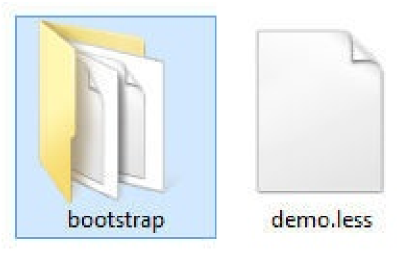
### The Folders Structure

Here's what the project's directory structure looks like.



- All the Bootstrap files go into the bootstrap folder. This also includes a mixins folder where we place the Bootstrap mixins.
- The index.html file goes straight inside the bs-mixins-demo folder.
- The demo.css file will be added to the css folder, once the Less files are compiled.
- The project's Less file, demo.less, is placed inside the less folder. All the Less code I write for this demo goes here.

In a real world project, I'd break its contents into separate .less files. However, given the reduced size of this demo, we can get away with just one file.

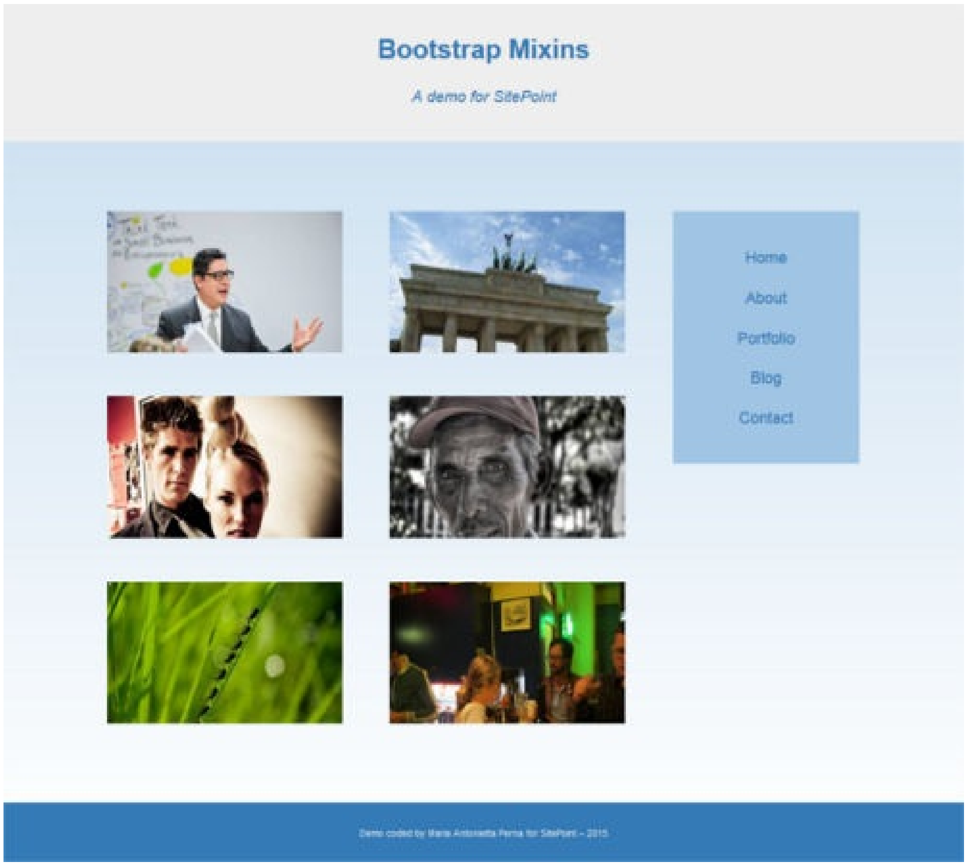


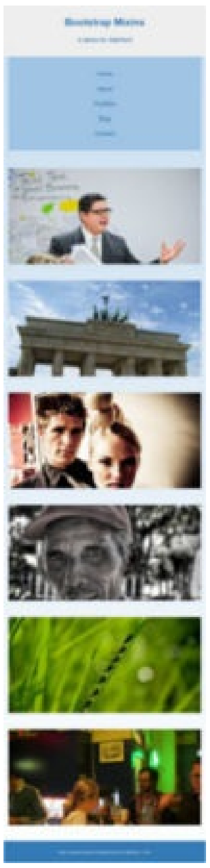
## The HTML Structure

The demo page consists of a simple two-column layout with header and footer. Bootstrap components and JavaScript plugins are not used. But don't be fooled - there's enough in there to show the Bootstrap's mixins goodness in action:

- adaptive layout
- nested columns
- columns offset

- different column display order on desktop and mobile view
- Bootstrap buttons
- CSS3 gradients
- CSS3 card-flipping effect on hover





Below is what the outline of index.html looks like.

```
<!-- HEADER -->
<header role="banner">
  <div class="container">
    <h1><a href="#">Site Title</a></h1>
    <p>Site Tagline</p>
  </div>
</header>

<!-- PAGE CONTENT -->
<div class="container">
  <div class="page-content">

    <!-- NAVIGATION -->
    <aside class="sidebar" role="complementary">
      <nav role="navigation">

      </nav>
    </aside>

    <!-- MAIN CONTENT -->
    <main class="main-content" role="main">

      <!-- NESTED COLUMNS GRID -->
      <article class="card">
      <!-- Column 1 -->
      </article>

      <article class="card">
      <!-- Column 2 -->
      </article>

      ...

    </main>
  </div>
</div>

<!-- FOOTER -->
<footer role="contentinfo">
  <div class="container">

  </div>
</footer>
```

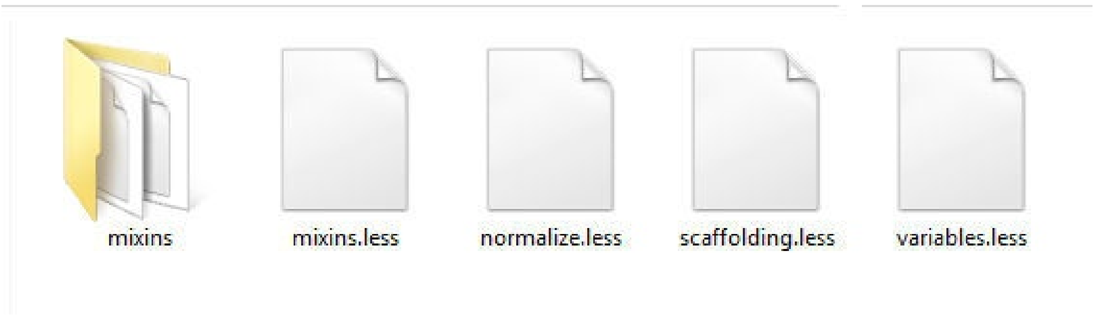
Here's an added bonus of using the Less source code: a clean HTML document without any typical Bootstrap grid classes.

## The Bootstrap Files

The Bootstrap files we need for this demo are available on the Bootstrap website. Make sure, you grab the **Less source** files.

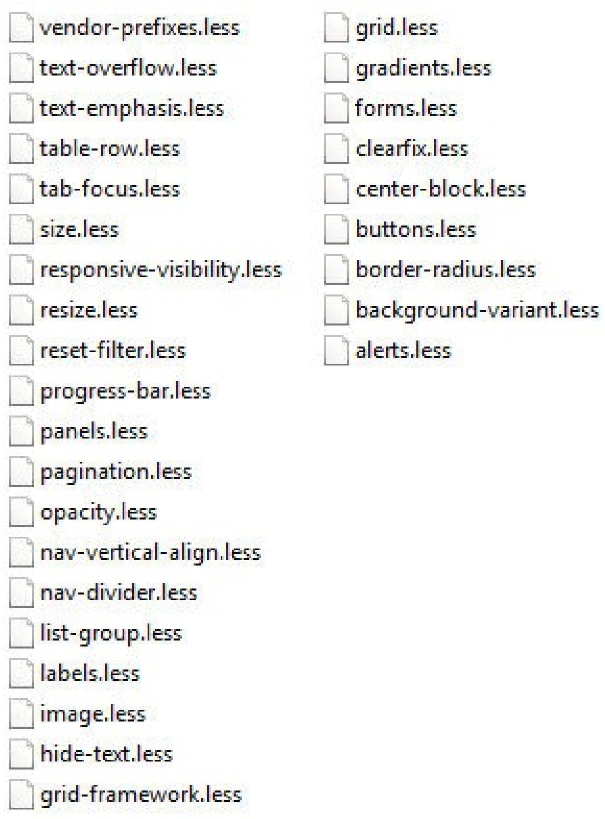
The files listed below need to be copied over from the Less folder of the downloaded source code into the demo's bootstrap folder:

- mixins.less
- normalize.less
- scaffolding.less
- variables.less



mixins mixins.less normalize.less scaffolding.less variables.less

Also, the entire content of the mixins directory of the Bootstrap source code needs to be copied over into the demo's mixins folder.



vendor-prefixes.less
text-overflow.less
text-emphasis.less
table-row.less
tab-focus.less
size.less
responsive-visibility.less
resize.less
reset-filter.less
progress-bar.less
panels.less
pagination.less
opacity.less
nav-vertical-align.less
nav-divider.less
list-group.less
labels.less
image.less
hide-text.less
grid-framework.less
grid.less
gradients.less
forms.less
clearfix.less
center-block.less
buttons.less
border-radius.less
background-variant.less
alerts.less

## The Less @import Directive

To make the Bootstrap .less files available to our demo, Less offers the `@import` directive. In this demo, we import our copy of the Bootstrap files into the demo.less file like so:

```
@import (reference) "bootstrap/variables.less";
@import (reference) "bootstrap/mixins.less";
@import "bootstrap/normalize.less";
@import "bootstrap/scaffolding.less";
```

Less has six keywords that can be used with the `@import` directive: reference, inline, less, css, once, and multiple.

The **reference** keyword is a great feature: now we have everything contained in bootstrap/variables.less and bootstrap/mixins.less at our fingertips. However, only what we actually use for our project will be compiled into the resulting CSS document.

## The Compiler

You can compile Less code both **server side** and **client side**.

Client side compilation is as quick as adding demo.less and less.js (downloadable from the LssCss.org website) in the `head` section of your HTML document.

```
<link href="less/demo.less" rel="stylesheet/less">
<script src="less/less.js"></script>
```

Client side compiling is great to get started with Less and experimenting with it. However, for a production site, the best option is server side precompiling with **node.js** or a third party tool.

This demo uses Prepros, a precompiler for Windows, Mac, and Linux available both as a free and a paid download. You're free to use your favorite tool, it won't affect the end result.

For an in-depth guide on how to use Prepros to precompile your Less code, Multilingual Preprocessing with Prepros by Ivaylo Gerchev will tell you all you need to know.

# What are Less Mixins?

Mixins in Less offer a way to package all the properties of a class so that we can reuse them as a property inside another class.

From the LessCss website:

> Mixins are a way of including ("mixing in") a bunch of properties from one rule-set into another rule-set.

Here's an example.

If we choose to build a web layout using floats, we will need a technique to clear those floats.

Below is the Clearfix Hack by Nicolas Gallagher, also used by the Bootstrap framework, turned into a Less mixin.

```
.clearfix() {
    &:before,
    &:after {
        content: " ";
        display: table;
    }
    &:after {
        clear: both;
    }
}
```

The advantage of having it packaged into a mixin is that we can now add it wherever we need to clear floats with just one line of code. For example, here's how to add it to a container element that encompasses a number of floated elements:

```
.container {
    property1: value1;
    property2: value2;
    .clearfix();
}
```

When compiled into CSS, this code outputs the following:

```
.container:before,
.container:after {
  display: table;
  content: " ";
}
.container:after {
  clear:both;
}
.container {
  property1: value1;
  property2: value2;
}
```

The level of complexity of a mixin varies in relation to what you intend to achieve with it. It's time to see what it's possible to achieve using the Bootstrap mixins as a library for our project.

## Taking Advantage of the Bootstrap Mixins

If you're ready to move from the basics and embark on your journey to mixins ninjahood, using an excellent mixins library not only helps you write awesome CSS code, but is also a great way to learn from the best. Let's open the mixins folder and explore the way Bootstrap builds its Less code.

### Layout Mixins

The mixins that I personally find most useful are those that help me build the page layout. You can find these in grid.less.

### The `.container-fixed()` Mixin

This mixin is designed to generate the CSS for centering the content of a web page. Here it is:

```
.container-fixed(@gutter: @grid-gutter-width) {
  margin-right: auto;
  margin-left: auto;
  padding-left:  (@gutter / 2);
  padding-right: (@gutter / 2);
  &:extend(.clearfix all);
}
```

Let's take a closer look into the `.container-fixed()` mixin.

This is a parametric mixin, that is, a mixin that takes one or more arguments. Writing mixins like this gives us quite a bit of flexibility. We can use the same ruleset but customize it differently by changing the value we assign to the arguments.

This particular Bootstrap mixin uses **arguments with default values**. The argument `@gutter` has a default value of `@grid-gutter-width`, that you can find in the `variables.less` file. This means that, if no value for this argument is passed when using the mixin, the code will fall back on the default value.

One more interesting bit to notice about the `.container-fixed()` mixin is its use of **&:extend(.clearfix all)**.

This piece of code does the following:

- it includes a `.clearfix` class built with the `.clearfix()` mixin. This class has the functionality of clearing floated child elements;
- it extends that functionality to the elements and classes styled using the `.container-fixed()` mixin. It does this with the Less `:extend()` pseudo-class;
- by adding the "all" keyword at the end, it ensures that the compiler extends the clearfix functionality to all selectors nested inside the extended class.

Since the release of version 3.1.0 of the Bootstrap framework, `&:extend(.clearfix all)` has replaced the use of the `.clearfix()` mixin inside the `.container-fixed()` mixin (you can define a mixin inside another mixin). Let's examine why we can consider this move as an improvement in the quality of the Bootstrap CSS code.

Applying the `.clearfix()` mixin to any element or class that contains floated children ends up repeating the same clearfix hack over and over in the CSS document.

What the Less ":extend()" pseudo-class does is to produce a CSS declaration that groups together all the elements and classes that share the same CSS rules. For instance, if you were to apply the `.clearfix()` mixin to `.container`, `.row`, and `.footer`, your compiled CSS would repeat the same clearfix hack for each of the three classes.

However, using the Less `:extend()` pseudo-class instead, as Bootstrap now does, will output this CSS code:

```
.container:before,
.container:after,
.row:before,
.row:after,
.footer:before,
.footer:after {
  display: table;
  content: " ";
}
.container:after,
.row:after,
.footer:after {
  clear:both;
}
```

The gain of using the `:extend()` pseudo-class is a compliance with the DRY principle. In particular, it avoids repeating the same CSS code by merging together elements and class selectors that share the same bunch of properties.

To use the Bootstrap `.container-fixed()` mixin in your code, add `.container-fixed()` to a class or a HTML element. In this article's demo, I use it for the `.container` class, just like Bootstrap does:

```
.container { .container-fixed(); }
```

Here's what the CSS output of the `.container` class looks like:

```
.container {
  margin-right: auto;
  margin-left: auto;
  padding-left: 15px;
  padding-right: 15px;
}
```

## The .make-row() Mixin

In the Bootstrap grid system, columns live inside a wrapper element. The `.make-row()` mixin generates the styles for this element. Here's what it looks like:

```
.make-row(@gutter: @grid-gutter-width) {
  margin-left:  (@gutter / -2);
  margin-right: (@gutter / -2);
  &:extend(.clearfix all);
}
```

This mixin calculates a left and right margin for the row. Also, since the `.make-row()` mixin is designed to style wrappers for floated columns, it's extended with the `.clearfix` class for floats clearing.

In this article's demo, I use it on the `.page-content` class, like this:

```
.page-content { .make-row(); }
```

The CSS output is:

```
.page-content {
  margin-left: -15px;
  margin-right: -15px;
}
```

## The Columns Mixins

The Bootstrap grid system uses four column sizes for responsive layouts: the extra-small, the small, the medium, and the large size.

Each column size is generated by a corresponding mixin. Because these mixins have a similar pattern, let's just examine the mixin for the medium column.

```
.make-md-column(@columns; @gutter: @grid-gutter-width) {
  position: relative;
  min-height: 1px;
  padding-left:  (@gutter / 2);
  padding-right: (@gutter / 2);

  @media (min-width: @screen-md-min) {
    float: left;
    width: percentage((@columns / @grid-columns));
  }
}
```

This is another parametric mixin. The `@gutter` argument has a default value, but the value for the `@columns` argument will have to be provided when using the mixin. If you don't assign a value to the `@columns` argument, Less will throw an error.

A great feature Less offers is **nesting**. You can nest selectors in a way that reflects the parent-children relationship in your HTML document, without having to write the parent's selector every time you reference a child element's selector.

For instance, what in regular CSS looks like this:

```
article {
  background: blue;
}
article p {
  color: gray;
}
```

... using Less nesting, looks like this:

```
article {
  background: blue;
  p {
    color: gray;
  }
}
```

The Bootstrap `.make-md-column()` mixin shows us how to take advantage of this nifty Less feature inside a mixin. The LessCss website says:

> Directives such as `media` or `keyframe` can be nested in the same way as selectors. Directive is placed on top and relative order against other elements inside the same ruleset remains unchanged. This is called bubbling.

The media query inside the `.make-md-column()` mixin dictates the column's behavior when the screen's width corresponds to the value of the `@screen-md-min` variable (Bootstrap gives a default value of 992px to this variable. You can find this out in `variables.less`). When the screen's width hits the assigned value, the column's width will be equal to the percentage value of your design's number of columns divided by the total number of columns (Bootstrap's default value for the total number of columns is twelve).

The demo uses the `.make-md-column()` mixin for the sidebar, the main content column, and the six nested columns inside the main content.

```
.sidebar {
  .make-md-column(4);
}
.main-content {
  .make-md-column(12);
}
.card {
  .make-md-column(5);
}
```

The CSS output for the .sidebar is:

```
.sidebar {
  position: relative;
  min-height: 1px;
  padding-left: 15px;
  padding-right: 15px;
}
@media (min-width: 992px) {
  .sidebar {
    float: left;
    width: 33.33333333%;
  }
}
```

## The Button Mixin

The mixin Bootstrap uses to build the styles for the button element is a great example of how convenient it is to use Less for CSS development.

The `.button-variant()` mixin is located in the `buttons.less` file inside the mixins folder. It outputs default button styles, as well as styles for common button states. This means that you add this mixin to a button element's selector and never worry about writing other rules for all those button states. What a time saver!

Here's the mixin's code:

```
.button-variant(@color; @background; @border) {
  color: @color;
  background-color: @background;
  border-color: @border;

  &:hover,
  &:focus,
  &.focus,
  &:active,
  &.active,
  .open > .dropdown-toggle& {
    color: @color;
    background-color: darken(@background, 10%);
    border-color: darken(@border, 12%);
  }
  &:active,
  &.active,
  .open > .dropdown-toggle& {
    background-image: none;
  }
  &.disabled,
  &[disabled],
  fieldset[disabled] & {
    &,
    &:hover,
    &:focus,
    &.focus,
    &:active,
    &.active {
      background-color: @background;
      border-color: @border;
    }
  }

  .badge {
    color: @background;
    background-color: @color;
  }
}
```

This parametric mixin needs three values from you and spits out a major chunk of CSS code that covers all buttons' states.

There are some selectors that are part of the Bootstrap framework in there, like `.badge` or `.dropdown-toggle`. But nothing prevents you from copying over this mixin into demo.less (or into your project's specific mixins library folder) and customizing it to fit your own needs. If you didn't want any extraneous Bootstrap selectors, you could rewrite the `.button-variant()` mixin like so:

```
.demo-button-variant(@color; @background; @border) {
  color: @color;
  background-color: @background;
  border-color: @border;

  &:hover,
  &:focus,
  &.focus,
  &:active,
  &.active {
    color: @color;
    background-color: darken(@background, 10%);
    border-color: darken(@border, 12%);
  }
  &:active,
  &.active {
    background-image: none;
  }
  &.disabled,
  &[disabled],
  fieldset[disabled] & {
    &,
    &:hover,
    &:focus,
    &.focus,
    &:active,
    &.active {
      background-color: @background;
      border-color: @border;
    }
  }
}
```

This is how I use the `.demo-button-variant()` mixin in the demo page:

```
.action-btn {
 .demo-button-variant(@gray; lighten(@brand-info, 20%); darken(@brand-info, 10%));
}
```

The CSS output is:

```
.action-btn {
  color: #555555;
  background-color: #b0e1ef;
  border-color: #31b0d5;
}
.action-btn:hover,
.action-btn:focus,
.action-btn.focus,
.action-btn:active,
.action-btn.active {
  color: #555555;
  background-color: #85d0e7;
  border-color: #2289a7;
}
.action-btn:active,
.action-btn.active {
  background-image: none;
}
.action-btn.disabled,
.action-btn[disabled],
fieldset[disabled] .action-btn,
.action-btn.disabled:hover,
.action-btn[disabled]:hover,
fieldset[disabled] .action-btn:hover,
.action-btn.disabled:focus,
.action-btn[disabled]:focus,
fieldset[disabled] .action-btn:focus,
.action-btn.disabled.focus,
.action-btn[disabled].focus,
fieldset[disabled] .action-btn.focus,
.action-btn.disabled:active,
.action-btn[disabled]:active,
fieldset[disabled] .action-btn:active,
.action-btn.disabled.active,
.action-btn[disabled].active,
fieldset[disabled] .action-btn.active {
  background-color: #b0e1ef;
  border-color: #31b0d5;
}
```

What about all those ampersand symbols (&)? The **ampersand selector** refers to the parent selector inside a nested selector. The most common use is with pseudo-classes, like in the example below. Instead of repeating `.action-btn` like in vanilla CSS, slamming an ampersand does the job and saves precious time.

```
.action-btn {
  color: gray;
  &:hover {
    text-decoration: none;
  }
}
```

This compiles into the following CSS:

```
.action-btn {
  color: gray;
}
.action-btn:hover {
  text-decoration: none;
}
```

You can also use the ampersand selector inside mixins, like in the Bootstrap `.button-variant()` mixin. When the ampersand is placed at the end of the selector list, it allows you to reverse the order of nesting. To use a small chunk of the `.button-variant()` mixin as example:

```
.button-variant(@color; @background; @border) {
...
  &:active,
  &.active,
  .open > .dropdown-toggle& {
    background-image: none;
  }
...
}
```

When the mixin above is used on the `.action-btn` selector, it outputs the following CSS

```
.action-btn:active,
.action-btn.active,
.open > .dropdown-toggle.action-btn {
  background-image: none;
}
```

## Gradients Mixins

In gradients.less inside the mixins folder, you'll find a bunch of mixins encapsulated within a **namespace**. This is a great technique for grouping mixins for organizational purposes and making your code more portable.

Here's the general structure of the Bootstrap gradient mixin to illustrate this point (the complete source code is available with the demo's files).

```
#gradient {
  .horizontal(@start-color: #555; @end-color: #333; @start-percent: 0%; @end-percent:
100%) {
    //mixin's code goes here
  }
  .vertical(@start-color: #555; @end-color: #333; @start-percent: 0%; @end-percent: 10
0%) {
    //mixin's code goes here
  }
  .directional(@start-color: #555; @end-color: #333; @deg: 45deg) {
    //mixin's code goes here
  }
  ...
}
```

As you can see, namespaces are defined in the same way as CSS ID selectors.

In the example above, the code that goes inside each grouping is not different from what you would write in your CSS code for a gradient, only using the mixin's arguments instead of fixed values.

In the demo for this article, I use the vertical gradient mixin to style the background of the entire web page:

```
body {
  #gradient > .vertical(lighten(@brand-primary, 40% ); lighten(@brand-primary, 60%) );
}
```

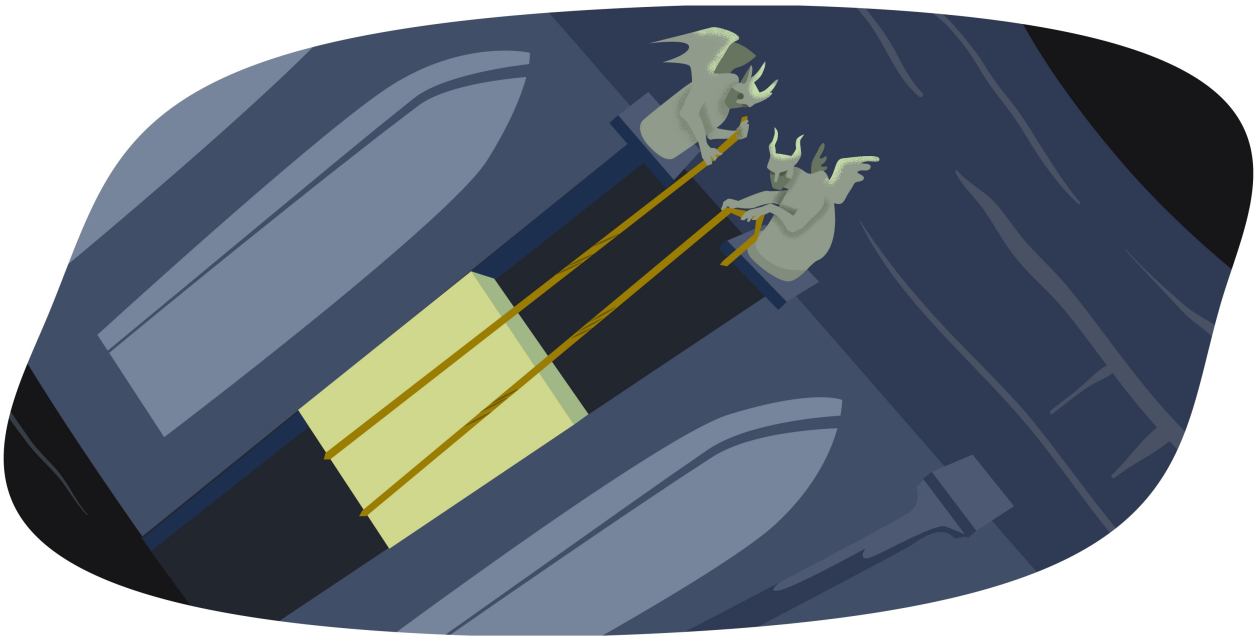
The Bootstrap gradient mixin has default values as arguments, therefore you can just use empty brackets and it works. Here, I change the start and end color of the gradient using the Less pre-built `lighten()` and `darken()` color functions.

Less has a great variety of color functions. The ones used above are designed to produce a lighter or darker variant of the input color. Really handy when you need a quick way of generating a palette of different shades of the same basic color value.

## Conclusion

In this article I've highlighted some features of the Less preprocessor language by getting close and personal with the Bootstrap mixins and using a good number of them in a small project.

Exploring and using Bootstrap as a rich mixins library means translating all the theory on the Less preprocessor language we gain from books, courses, and tutorials, into a real-world context. It's a great way of making the transition from handling the basics of Less to becoming a pro.



# Getting Bootstrap to Play Nice With Masonry

By Maria Antonietta Perna

On the Masonry website, we read that Masonry is...

> … a JavaScript grid layout library. It works by placing elements in optimal position based on available vertical space, sort of like a mason fitting stones in a wall.

Bootstrap is one of the most widely adopted open source front-end frameworks. Include Bootstrap in your project, and you'll be able to whip up responsive web pages in no time.

If you tried using Masonry together with the Tabs widget, one of the many JavaScript components Bootstrap has to offer, chances are you've stumbled on some kind of annoying behavior. I did, and this article highlights what the issue is and what you can do to solve it.

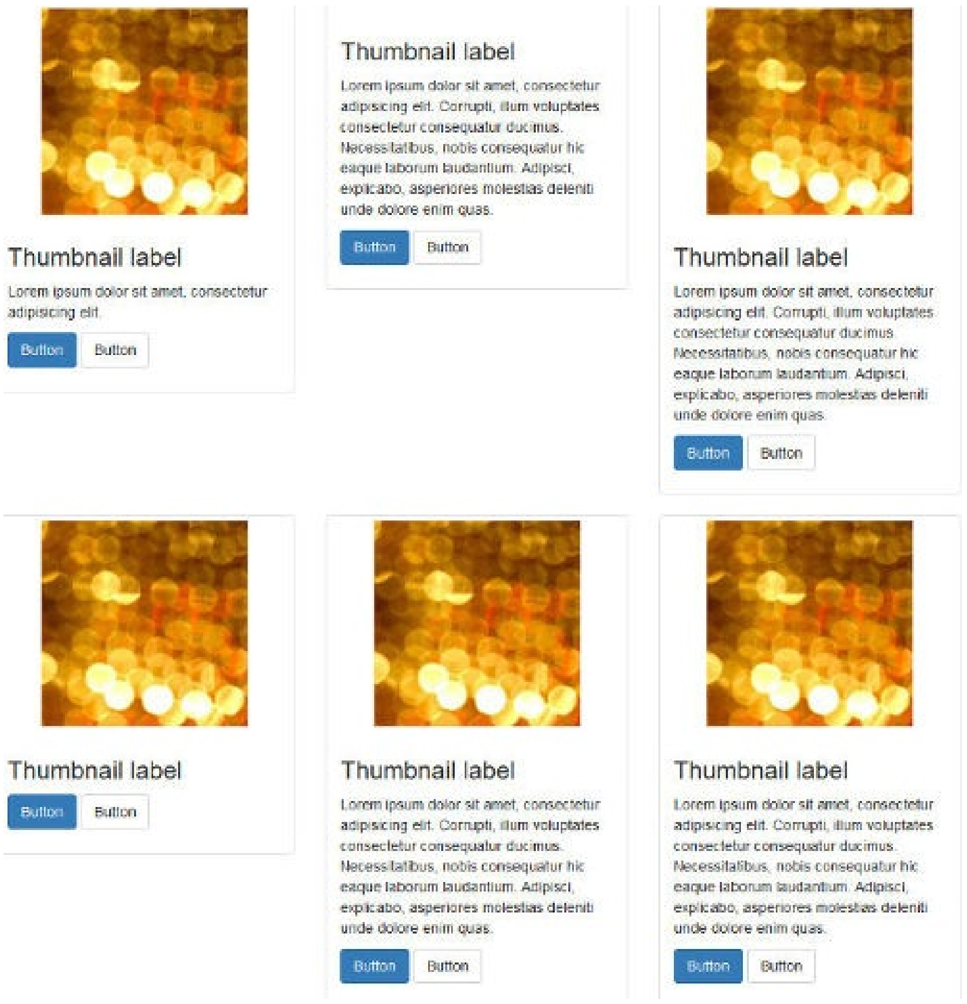
## Bootstrap Tabs Explained

Bootstrap's Tabs component includes two key, related pieces: a tabbed navigation element and a number of content panels. On page load, the first panel has the class `.active` applied to it. This enables the panel to be visible by default. This class is used via JavaScript to toggle the panel's visibility via the events triggered by the tabbed navigation links: if `.active` is present the panel is visible, otherwise the panel is hidden.

If you have some web content that's best presented in individual chunks, rather than crammed all in one spot, this kind of tabs component might come in handy.

## Why Masonry?

In some cases, the content inside each panel is suited to being displayed in a responsive grid layout. For instance, a range of products, services, and portfolio items are types of content that can be displayed in grid format. However, if grid cells are not of the same height, something like what you see below can happen.


Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Thumbnail label

Thumbnail label

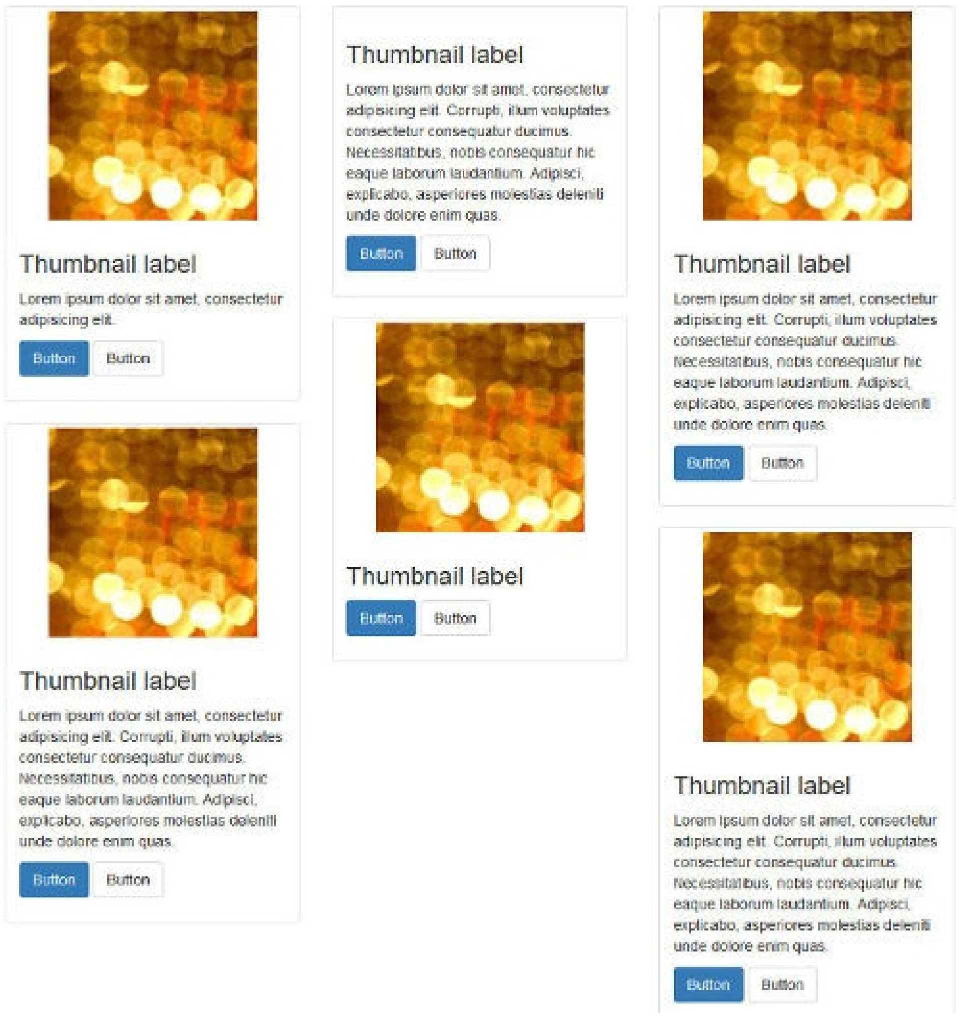
Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

A wide gap separates the two rows of content and the layout appears broken.

That's when Masonry saves the day. Add some Masonry magic to the mix and your grid dynamically adapts to the screen real estate, eliminating all ghastly gaps.



Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit.

Button Button

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Button Button

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Button Button

Thumbnail label

Button Button

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Button Button

Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Button Button

## Setting Up a Demo Page

Getting a demo page up and running helps to show how integrating Bootstrap's Tabs with Masonry is not as straightforward as one expects. This article's is based on the Starter Template, available on the Bootstrap website.

CodePen demo page - http://codepen.io/SitePoint/pen/mywEMR/

Each grid item inside the tab panels is built with the Bootstrap grid system and the Bootstrap Thumbnails component. Here's a code snippet to illustrate its structure:

```
<div class="col-sm-6 col-md-4">
  <div class="thumbnail">
    <img src="http://lorempixel.com/200/200/abstract" alt="">
    <div class="caption">
      <h3>Thumbnail label</h3>
      <p>...</p>
      <p>
        <a href="#" class="btn btn-primary" role="button">Button</a>
        <a href="#" class="btn btn-default" role="button">Button</a>
      </p>
    </div>
  </div>
</div>

<!-- Repeat two more times ... -->
```

The code above builds a three-column grid on large to medium screens and a two-column grid on smaller screens. If you need a refresher on the Bootstrap grid system, Understanding Bootstrap's Grid System by Syed Fazle Rahman is a great read.

The Tabs widget in the demo page has the following HTML structure:

```
<div role="tabpanel">
  <!-- Nav tabs -->
  <ul class="nav nav-tabs" role="tablist">
    <li role="presentation" class="active">
      <a href="#panel-1" aria-controls="panel-1" role="tab" data-toggle="tab">Panel 1</
a>
    </li>
    <li role="presentation">
      <a href="#panel-2" aria-controls="panel-2" role="tab" data-toggle="tab">Panel 2</
a>
    </li>
    <li role="presentation">
      <a href="#panel-3" aria-controls="panel-3" role="tab" data-toggle="tab">Panel 3</
a>
    </li>
    <li role="presentation">
      <a href="#panel-4" aria-controls="panel-4" role="tab" data-toggle="tab">Panel 4</
a>
    </li>
  </ul>

  <!-- Tab panels -->
  <div class="tab-content">
    <div role="tabpanel" class="tab-pane active" id="panel-1">
      <div class="row masonry-container">
        <div class="col-md-4 col-sm-6 item">
          <!-- Thumbnail goes here -->
        </div>
        <div class="col-md-4 col-sm-6 item">
          <!-- Thumbnail goes here -->
        </div>
        <div class="col-md-4 col-sm-6 item">
          <!-- Thumbnail goes here -->
        </div>
        ...
      </div><!--End masonry-container  -->
    </div><!--End panel-1  -->
    <div role="tabpanel" class="tab-pane" id="panel-2">
      <!-- Same as what goes inside panel-1 -->
    </div><!--End panel-2  -->
    ...
  </div><!--End tab-content  -->
</div><!--End tabpanel  -->
```

Here are a few things to note about the code snippet above:

- HTML comments point to the Tab's key components: **Nav tabs** marks the tabbed navigation section, and **Nav panels** marks the content panels.
- The tabbed links connect to the corresponding content panel through the value of their

  `href` attribute, which is the same as the value of the `id` attribute of the content panel. For instance, the link with `href="#panel-1"` opens the content panel with `id=panel-1`.
- Each anchor tag in the navigation section includes `data-toggle="tab"`. This markup enables the tabs component to work without writing any additional JavaScript.
- Finally, the elements that Masonry needs to target have a class of `.masonry-container` that apply to the wrapper `div` element that encompasses all the grid items and a class of `.item` that apply to each single grid item.

To see the full power of the Masonry library, make sure the grid items are of varying heights. For instance, delete the image on one item, shorten a paragraph on another, etc.

For the full code, check out the code panels in the CodePen demo.

## Adding the Masonry Library

You can download Masonry from the official website by clicking on the Download masonry.pkgd.min.js button.

To avoid layout issues, the library's author recommends using Masonry together with the imagesLoaded plugin.

Masonry doesn't need the jQuery library to work. However, because the Bootstrap JavaScript components already use jQuery, I'll be making life easier for myself and initialize Masonry the jQuery way.

Here's the code snippet we need to initialize Masonry using jQuery and imagesLoaded.

```
var $container = $('.masonry-container');
$container.imagesLoaded( function () {
  $container.masonry({
    columnWidth: '.item',
    itemSelector: '.item'
  });
});
```

The code above caches the `div` that wraps all the grid items in a variable called `$container`.

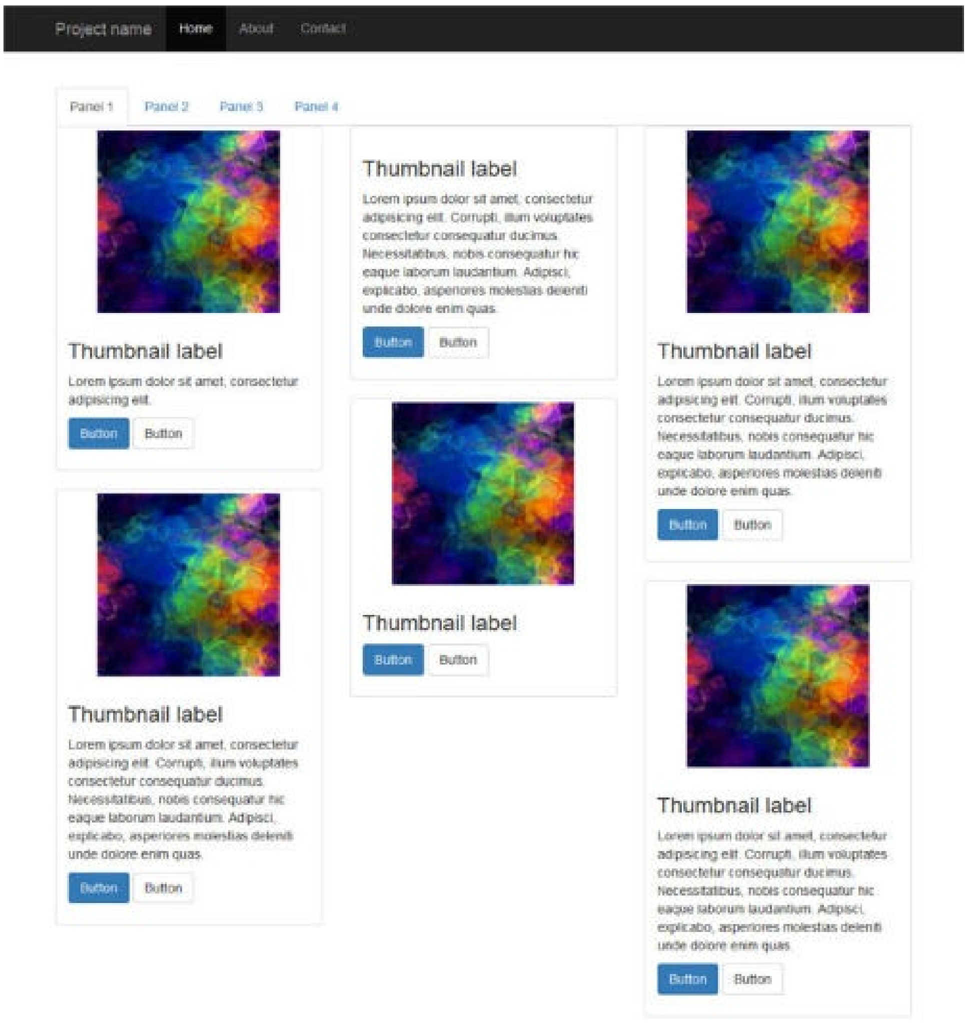
Next, Masonry is initialized on `$container` with a couple of recommended options. The `columnWidth` option indicates the width of a column of a horizontal grid. Here it is set to the width of the single grid item by using its class name. The `itemSelector` option indicates which child elements are to be used as item elements. Here, it's also set to the single grid item.

It’s now time to test the code.

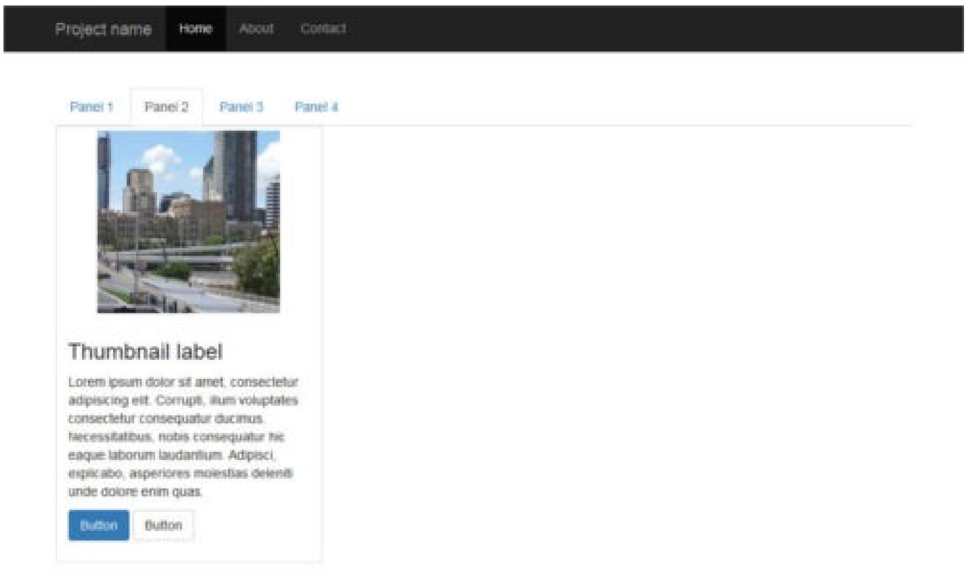
## Oops! What’s up with the Hidden Panels?

On a web page that doesn’t use Bootstrap Tabs, the code above works like a charm. However, in this case, you soon realize a kind of funny behavior occurs.

First, it seems fine because the grid inside the default active tab panel is displayed correctly:



However, if you click on a tabbed navigation link to reveal the hidden panel’s content, here’s what happens:



Thumbnail label

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Corrupti, illum voluptates consectetur consequatur ducimus. Necessitatibus, nobis consequatur hic eaque laborum laudantium. Adipisci, explicabo, asperiores molestias deleniti unde dolore enim quas.

Peeking inside the source code reveals that Masonry has fired as expected, but the position of each item is not being calculated correctly: grid items are all stacked on top of each other like a pack of cards.

And that's not all. Resizing the browser window causes the grid items to position themselves correctly.

## Let's Fix the Layout Bug

Since the unexpected layout bug becomes apparent after clicking on a tabbed navigation link, let's look into the events fired by Bootstrap's Tabs a bit more closely.

The events list is quite short. Here it is.

- **show.bs.tab** fires on tab show, but before the new tab has been shown.
- **shown.bs.tab** fires on tab show after a tab has been shown.
- **hide.bs.tab** fires when a new tab is to be shown (and thus the previous active tab is to be hidden).
- **hidden.bs.tab** fires after a new tab is shown (and thus the previous active tab is hidden).

Because the grid layout gets messed up after a tab has been shown, we go for the shown.bs.tab event. Here's the code, which we place just below the previous snippet:

```
$('a[data-toggle=tab]').each(function () {
  var $this = $(this);

  $this.on('shown.bs.tab', function () {
    $container.imagesLoaded( function () {
      $container.masonry({
        columnWidth: '.item',
        itemSelector: '.item'
      });
    });
  });
});
```

Here’s what happens in the code above:

The jQuery .each() function loops over each tabbed navigation link and listens for the `shown.bs.tab` event. As the event fires, the panel becomes visible and Masonry is re-initialized after all images have finished loading.

## Let’s Test the Code

If you’ve been following along, try out the CodePen demo below to check out the result:

Getting Bootstrap Tabs to Play Nice with the Masonry Library

Click on a tabbed navigation link and notice how this time the grid items fit evenly inside each content panel. Resizing the browser causes the items to reposition themselves correctly with a nice animation effect.

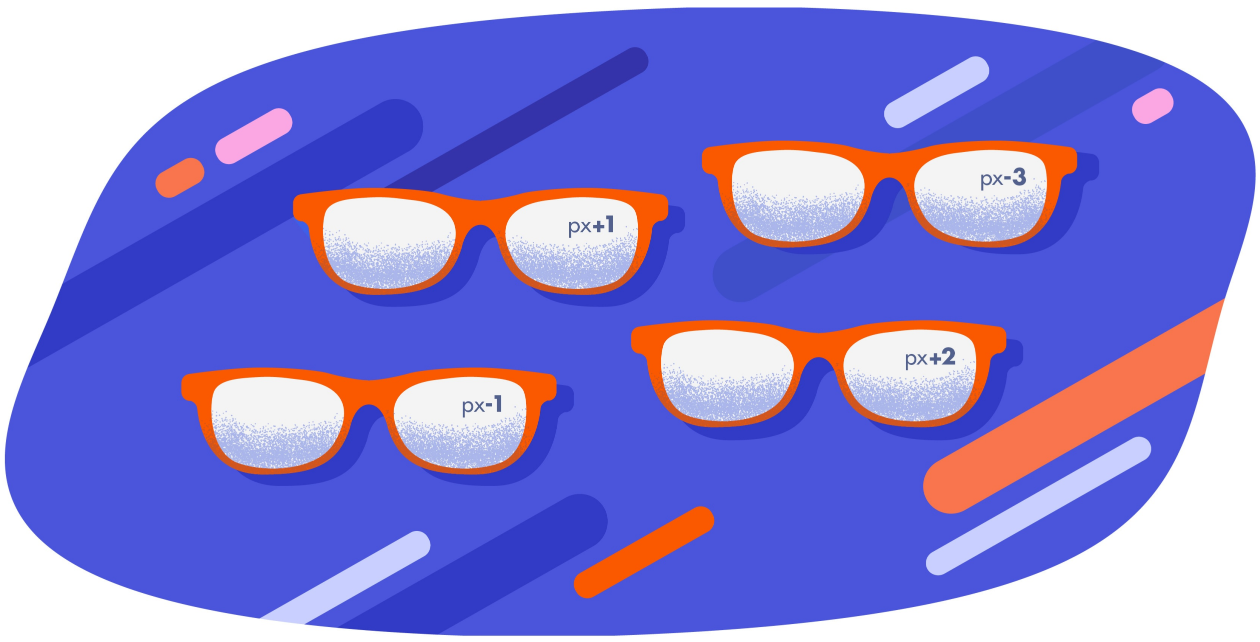
That’s it, job done!

## Conclusion

In this article I’ve shown how to integrate Bootstrap’s Tabs component with the Masonry JavaScript library.

Both scripts are easy to use and quite powerful. However, put them together and you’ll face some annoying layout bugs affecting the hidden tabs. As shown above, the trick is to re-initialize the Masonry library after each panel becomes visible.

With this solution in your toolbox, achieving great tiled layouts will be a breeze.



# Making Bootstrap a Little More Accessible

By Rhiana Heath

Like many front-end developers, I work with Bootstrap often. Sometimes I might need it for a quick proof of concept.

Other times I may inherit a larger project which has Bootstrap entrenched in the Web Application. It's versatility and success has meant that you start to see it everywhere -- it is clearly a very useful and popular tool.

However, when I was tasked with making a Bootstrap-based Web Application accessible, I ran into a few problems. As Bootstrap is mainly used for design I evaluated how accessible their base design was for people with visual impairments in terms of the colors.

But first a quick recap.

## What Does Accessibility Mean?

Accessibility has recently gained a lot more momentum as awareness of accessibility issues has steadily grown in the programming community. It is the practice of ensuring your website or web-application can be easily used by people with disabilities. This can include many different types of impairments with a wide range of severity, however the main four types of disability referred in the Web Content Accessibility Guidelines (WCAG) are:

1. Visual impairments
2. Hearing impairments

3. Motor impairments
4. Intellectual disabilities

### POUR Accessibility Principles

Today there are four principles to consider when developing accessible content for the web. They are referred to by the acronym 'POUR': Perceivable, Operable, Understandable and Robust. Color choices comes under the 'Perceivable' column for Web Accessibility. That is, the content on the website should be easy to see. In particular, text and images should be easy to discern against their background with a high color contrast ratio.

## So How Can You Tell If the Colors Are Accessible?

People with visual impairments of various degrees may view your website in different ways. Some may need to have the screen zoomed in, some may need to have the color settings changed to high contrast and others will also require the help of a screen reader or Braille reader.

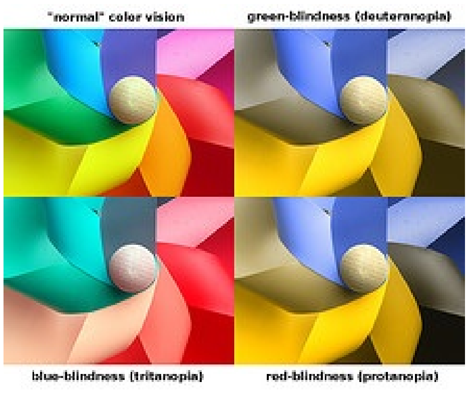

"normal" color vision
green-blindness (deuteranopia)
blue-blindness (tritanopia)
red-blindness (protanopia)

Photo: entirelysubjective

Additionally, nearly 10% of males will have some type of color blindness, having difficulty with certain color combinations. There are two ways I tested this. Firstly I tested for the color contrast ratio, and then I tested how the site performed under a high-contrast setting. This way you can ensure that no content will be lost on your site if people are viewing it in a different way.

## Color Contrast Ratio Test

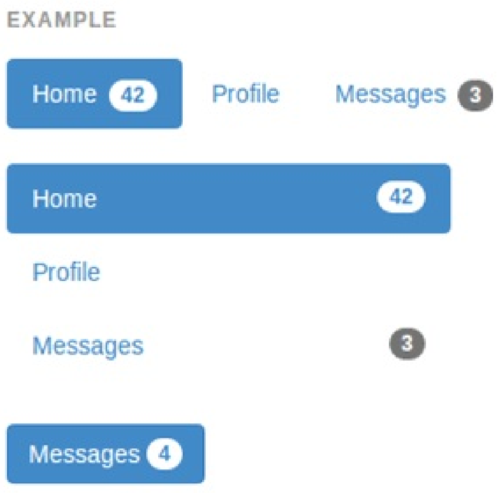
The first test I performed was to put the colors of the text and background into a color contrast checker. It then calculates if it meets the 4.5:1 color contrast ratio for accessibility. Some checkers can also calculate what it would like like for different levels of color blindness, and if it has enough contrast for that as well. From the Bootstrap site I tested their links, progress bars, navigation bars and their alerts to see if they were a high enough ratio for accessibility.

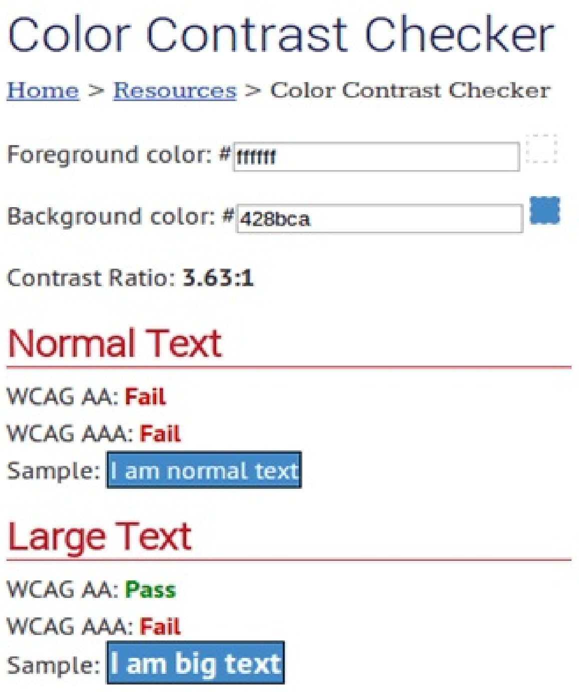
Taking into account their font-size and font-weights as per the contrast guidelines in the table below.

| Text | Normal | Bold | Ratio AA | Ratio AAA |
|---|---|---|---|---|
| Small | < 24px | < 19px | 4.5:1 | 7:1 |
| Large | > 24px | > 19px | 3:1 | 4.5:1 |

For example when testing their very common Bootstrap Blue color used in links, progress bars and navigation bars. When these were put into the checker below shows that this scheme meets the requirements only if the text is of large size. However the text shown is 14px and normal font-weight, which is classified as small so does not pass this requirement.

```
a {
  color: #428bca;
  text-decoration: none;
}

a {
  color: #3277b3;
  text-decoration: none;
}
```




Color Contrast Checker

Home > Resources > Color Contrast Checker

Foreground color: #ffffff

Background color: #428bca

Contrast Ratio: **3.63:1**

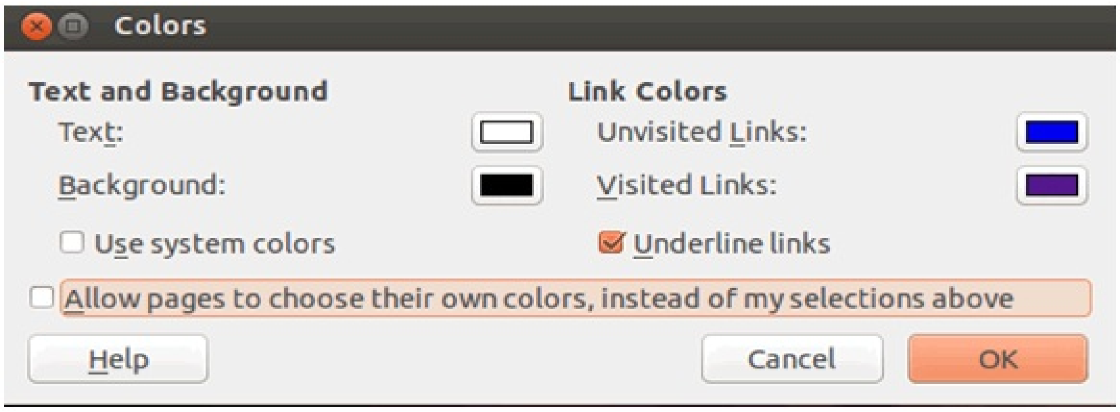
Normal Text

WCAG AA: **Fail**
WCAG AAA: **Fail**
Sample: I am normal text

Large Text

WCAG AA: **Pass**
WCAG AAA: **Fail**
Sample: **I am big text**

They have made some progress, for example the alerts in their previous version 2, the ratios varied from 3.3:1 to 3.9:1, just below the minimum of 4.5:1. Which would be required as this text is 14px, so classified as small text. In the new version 3, with a very slight change, all ratios are now above 4.5:1. With this little change they were able to comply with guidelines, without having to compromise the design. This was after someone noticed it and raised an issue on their GitHub repository requesting them to address this.


Colors

Text and Background
Text: ⬜
Background: ⬛
☐ Use system colors

Link Colors
Unvisited Links: 🟦
Visited Links: 🟪
☑ Underline links

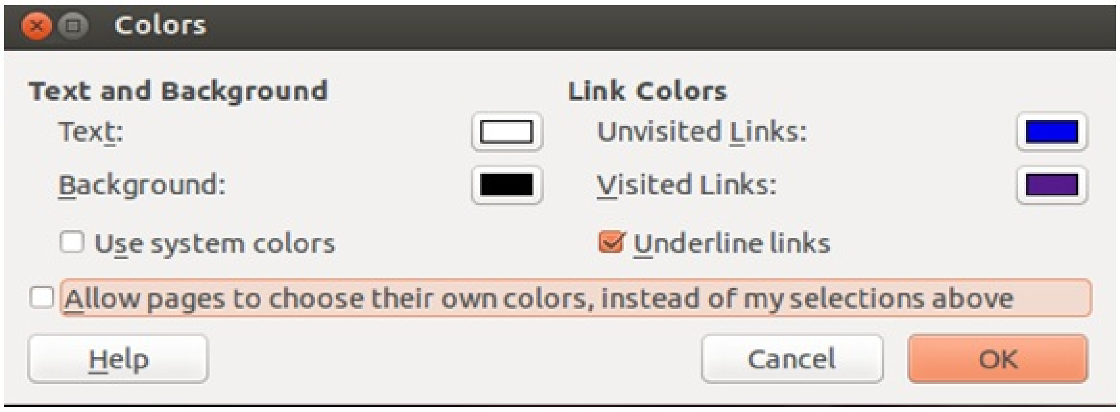
☐ Allow pages to choose their own colors, instead of my selections above

Help | Cancel | OK

I have spent a lot of time then going through and manually changing the colors in Web Applications so they meet this requirement. Alternatively the font-size and weight could be changed so it is classified as large text. I forked the Bootstrap Repository and updated their code with the changes I made. This can be downloaded and used instead of the standard one or used as a reference point as you may want to make your own which suit your scheme and also meet the color contrast ratio requirements.

## High Contrast Settings

The second way I tested was by viewing the site with high contrast settings. Generally what this means is inverting the colors, so instead of black text on a white background, you have white text on a black background which can be easier to see for people with partial blindness. In Chrome there is a plugin to view pages in high contrast. Both Windows and Mac's have a high contrast mode in their desktop settings, which will show the entire screen in high contrast. In Firefox this involves going into the settings, preferences, content and colors, changing the text to white, the background to black then un-ticking the following option.


Colors

Text and Background
Text: ⬜
Background: ⬛
☐ Use system colors

Link Colors
Unvisited Links: 🟦
Visited Links: 🟪
☑ Underline links

☐ Allow pages to choose their own colors, instead of my selections above

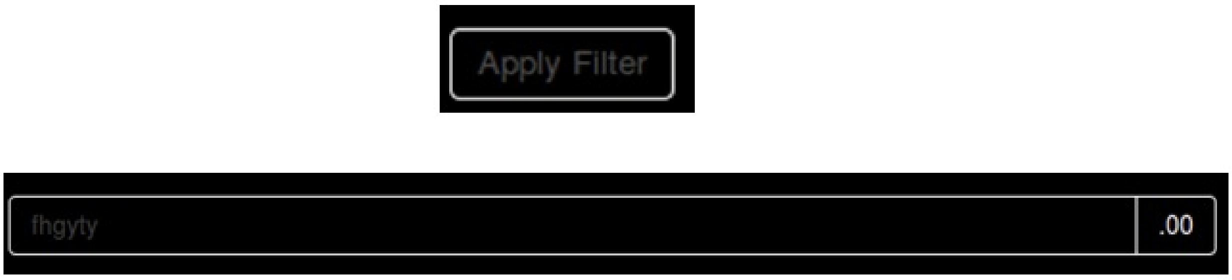
The high contrast test yielded some varying results. This is a common test during an accessibility audit. The most unforgiving combination seems to be Firefox in Windows, so most of my testing was using this browser/operating system as it is also a very popular combination for people with visual impairments to use. However testing in as many combinations of browsers and operating systems as possible is ideal as they all render slightly differently. One of the major problems I found was that bootstrap buttons either have very faint text under high contrast or none at all (with Firefox in Windows). I found this was due to the `background-color` property in CSS.

```
.btn-default {
  color: #333;
  background-color: #fff;
  border-color: #ccc;
}
```

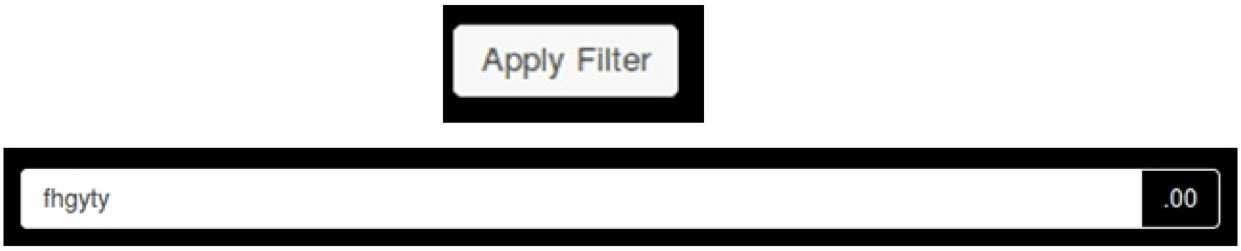
Once the background-color was removed completely it was much easier to see under high contrast, see below. Another area where this was affected were text inputs. The same sort of principle applied, they have a white background-color by default.

```
input {
    background-color: #fff;
}
```

Which means under high contrast you cannot make out the text you are typing. However after you remove this one line of CSS the visibility significantly improves in this mode with no visual difference for sighted users.

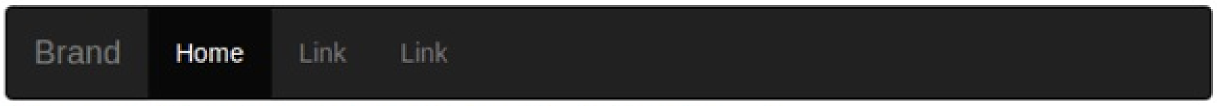


Current Bootstrap inputs: High contrast

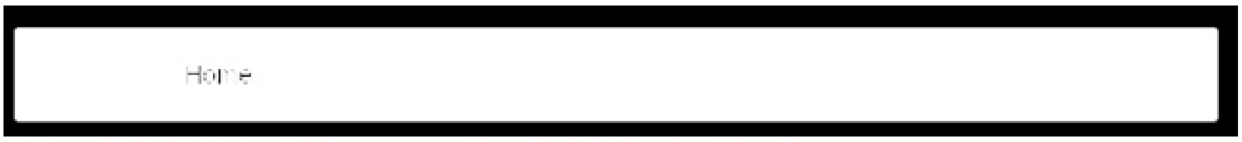


Modified Bootstrap inputs: High contrast

However this introduces a new problem: how to maintain the color for sighted users, however not using background-color in order to have it display in high-contrast settings. In Bootstrap 2 I found this was no problem as there were other CSS properties to control the look of the button, so removing the background-color had no visual impact to sighted users -- a 'win-win' there. In the custom GitHub repository mentioned, I've removed the background-color and replaced it with the CSS used in previous versions, slightly modified to meet the color contrast ratio and are also now visible in high contrast. Keep in mind, I only noticed this issue in Firefox. When checking this with high contrast in Chrome for example these buttons and inputs displayed fine. One thing that didn't though were Bootstraps inverted navbars. Ordinarily they look like this below.



Current Bootstrap navbar: High Contrast



Modified Bootstrap navbar: High Contrast

However in high contrast mode in Chrome it becomes unreadable. This is due largely to the color contrast ratio between the links and background not being high enough, making it disappears in this mode. The background would need to be darker or the links lighter. However even then it is difficult to see with this tool, so perhaps it's wiser to avoid just this color combination if possible.

## So, Should I Use Bootstrap?

Bootstrap has added a lot of features to the codebase to assist screen reader users. However in terms of pure color palette choices, most components were tested, and unfortunately not many of them passed the color contrast ratio tests or high-contrast readability settings.

## 3 Tips

1. Keep using Bootstrap in your website or application -- but keep it's accessibility blind-spots in mind.
2. Always upgrade to newer versions whenever possible as they do keep incrementally improving.
3. Feel free to use or reference the modified Bootstrap CSS file on GitHub instead of the standard one. Paypal also provides an accessibility plug-in for bootstrap which handles a lot of issues with keyboard accessibility for people with motor impairments.

If you find anything else, please take a moment to raise an issue with Bootstrap on their GitHub page, definitely submit code which would address the problem. Otherwise perhaps look into using a front-end framework that has accessibility features already included.

While they're aren't many out there yet, Accessible Template offer one and their site is an fantastic example best accessibility practices.

# Spicing Up the Bootstrap Carousel with CSS Animations

Adding a slider or carousel to showcase content on a website is a common client's request for developers. The amount of free and premium carousel plugins available is overwhelming, and a good many of them offer many useful configuration options and dynamic effects.

There are times, however, when a lightweight carousel with minimal options is all you need. In this case, if your project uses Bootstrap, the popular open source front-end framework, you won't need to look any further than the Bootstrap Carousel component.

In this article, I'm going to show how to add some fun animation effects to the Bootstrap Carousel, while still making sure this handy JavaScript component remains bloat-free and quick to implement.

## Introducing Animate.css

As rewarding as crafting my own animation effects can be, I'm going to use a well-known open source CSS3 animation library most aptly called Animate.css, by Dan Eden.

This is so that I can focus on the task at hand, rather than on explaining the code for CSS3 animations. However, if you want to delve into that topic, you'll enjoy the CSS3 Animations series here on SitePoint, by Craig Buckler.

Using Animate.css requires two steps:

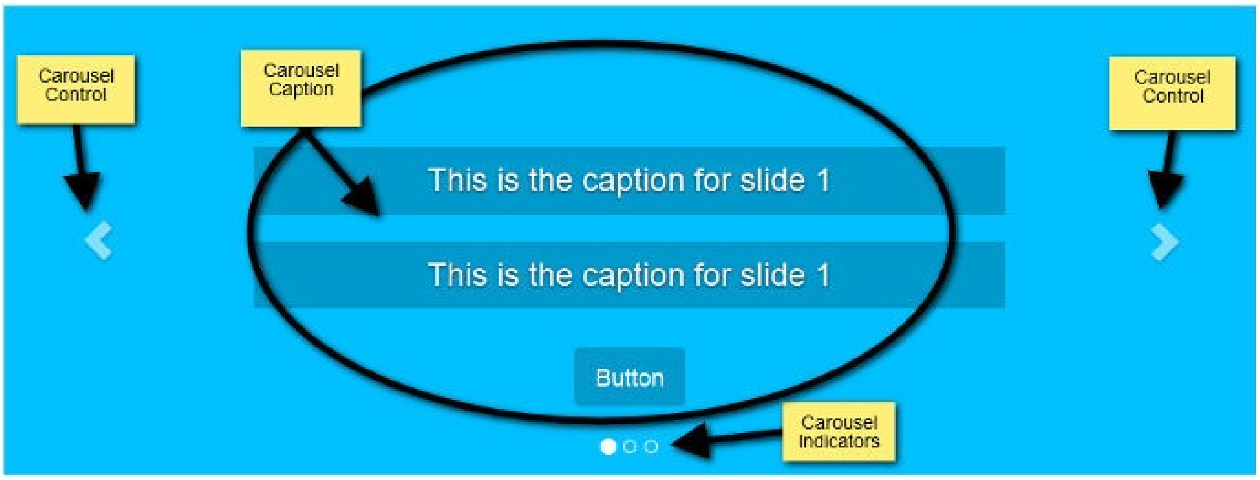
1. Include animate.min.css in the `<head></head>` section of your HTML document.
2. Add the classes of `animated yourchosenanimation` to the elements you intend to animate on your web page.

In the latter step you would replace `yourchosenanimation` with the class name corresponding to any of the numerous animations you see on the Animate.css website.

## Introducing the Bootstrap Carousel

The Bootstrap Carousel component has three main sections:

- The **Carousel indicators** track the overall number of slides, give users a visual clue of the position the slide currently being viewed occupies, and offer an alternative navigation for the slider.
- The **Carousel item**, located inside a wrapper container with a class of `.carousel-inner`, represents each individual slide. It's inside each item that you place your images. You can also add **captions** to your slides. The nice thing is that you can put pretty much any HTML element inside a container with the class of `carousel-caption` and Bootstrap will take care of the styling and formatting. It's these captions that we're going to animate.
- Finally, the **Carousel controls** are the navigation arrows that enable users to access the next and previous slides.



If you'd like to explore the Bootstrap Carousel component in detail, be sure to check out Creating JavaScript Sliders Using Bootstrap 3, by Syed Fazle Rahman.

To keep this demo simple, I'm not going to add images to the carousel. The focus is all on the carousel captions as the object of our animations.

## Building the HTML Structure

If you’re following along, here’s what you need to include in your project:

- jQuery
- Bootstrap’s CSS and JavaScript
- Animate.css
- A custom stylesheet and JavaScript document (both of which we will edit in this article)

To speed up the process, grab a starter template from the Bootstrap website and add the necessary files.

Here’s the code for the Bootstrap Carousel:

```
<div id="carousel-example-generic" class="carousel slide" data-ride="carousel">

  <!-- Indicators -->
  <ol class="carousel-indicators">
    <li data-target="#carousel-example-generic" data-slide-to="0" class="active">
    </li>
    <li data-target="#carousel-example-generic" data-slide-to="1"></li>
    <li data-target="#carousel-example-generic" data-slide-to="2"></li>
  </ol>

  <!-- Wrapper for slides -->
  <div class="carousel-inner" role="listbox">

    <!-- First slide -->
    <div class="item active">
      <div class="carousel-caption">
        <h3 data-animation="animated bounceInLeft">
          This is the caption for slide 1
        </h3>
        <h3 data-animation="animated bounceInRight">
          This is the caption for slide 1
        </h3>
        <button class="btn btn-primary btn-lg"
                data-animation="animated zoomInUp">Button</button>
      </div>
    </div><!-- /.item -->

    <!-- Second slide -->
    <div class="item">
      <div class="carousel-caption">
        <h3 class="icon-container" data-animation="animated bounceInDown">
          <span class="glyphicon glyphicon-heart"></span>
        </h3>
        <h3 data-animation="animated bounceInUp">
          This is the caption for slide 2
        </h3>
        <button class="btn btn-primary btn-lg"
                data-animation="animated zoomInRight">Button</button>
      </div>
    </div><!-- /.item -->

    <!-- Third slide -->
    <div class="item">
      <div class="carousel-caption">
        <h3 class="icon-container" data-animation="animated zoomInLeft">
          <span class="glyphicon glyphicon-glass"></span>
        </h3>
        <h3 data-animation="animated flipInX">
          This is the caption for slide 3
        </h3>
        <button class="btn btn-primary btn-lg"
                data-animation="animated lightSpeedIn">Button</button>
      </div>
    </div><!-- /.item -->

  </div><!-- /.carousel-inner -->

  <!-- Controls -->
  <a class="left carousel-control" href="#carousel-example-generic"
     role="button" data-slide="prev">
    <span class="glyphicon glyphicon-chevron-left" aria-hidden="true"></span>
    <span class="sr-only">Previous</span>
  </a>
  <a class="right carousel-control" href="#carousel-example-generic"
     role="button" data-slide="next">
    <span class="glyphicon glyphicon-chevron-right" aria-hidden="true"></span>
    <span class="sr-only">Next</span>
  </a>

</div><!-- /.carousel -->
```

If you've included the correct files and you open the above code in your browser, you should be able to see a nice working carousel, and all this without writing a single line of JavaScript. If you didn't add any images to the carousel, just assign a `min-height` value to the `.carousel .item` selector in your CSS document to prevent the carousel from collapsing.

The elements inside the carousel caption that we'll be animating have a `data-animation` attribute added to them with the specific animation class name as their respective value.

If you'd like to experiment with other animations from the Animate.css library, feel free to replace the values in the `data-animation` attribute with your chosen animation class names.

We'll be using the `data-animation` attribute in our JavaScript code shortly.

Although a simple auto-playing carousel is what you could be looking for in some cases, for this demo we're after more control.

As a first step in this direction, delete the `data-ride="carousel"` attribute from the `.carousel` element. The `data-ride` attribute initializes the carousel without having to write any JavaScript code. However, we're going to take control of the carousel using JavaScript, therefore the `data-ride` attribute won't be necessary.

## Adding CSS to the Carousel

Now, give free rein to your creativity and style the carousel captions according to your taste. The style rules that I'm going to focus on here are those relevant to the smooth working of this demo.

More specifically, we're taking control of the `animation-delay` property, which will define when each animation will start (note that vendor prefixes are omitted for brevity).

```
.carousel-caption h3:first-child {
  animation-delay: 1s;
}
.carousel-caption h3:nth-child(2) {
  animation-delay: 2s;
}
.carousel-caption button {
  animation-delay: 3s;
}
```

The snippet above ensures that the elements start their animation sequentially. There's room for play here. For instance, you can choose to start animating the first two headings at the same time, followed by the button animation. It's up to you, have fun with it!

## Writing the jQuery

Let's start by initializing the carousel. In your custom JavaScript file, add this code snippet:

```
var $myCarousel = $('#carousel-example-generic');

// Initialize carousel
$myCarousel.carousel();
```

We've set the carousel in motion. Next, we tackle the animation.

To animate the captions in the first slide, the script has to fire as soon as the page finishes loading in the browser. However, to animate subsequent slides as they come into view, our code will have to fire on the `slide.bs.carousel` event. This means that the same code will be used twice: on page load and on the `slide.bs.carousel` event.

Because we love the DRY ("Don't Repeat Yourself") principle, we're going to wrap our code in a function and attach it to the appropriate events as required.

Here's the code:

```
function doAnimations(elems) {
  var animEndEv = 'webkitAnimationEnd animationend';

  elems.each(function () {
    var $this = $(this),
        $animationType = $this.data('animation');

    // Add animate.css classes to
    // the elements to be animated
    // Remove animate.css classes
    // once the animation event has ended
    $this.addClass($animationType).one(animEndEv, function () {
      $this.removeClass($animationType);
    });
  });
}

// Select the elements to be animated
// in the first slide on page load
var $firstAnimatingElems = $myCarousel.find('.item:first')
                           .find('[data-animation ^= "animated"]');

// Apply the animation using our function
doAnimations($firstAnimatingElems);

// Pause the carousel
$myCarousel.carousel('pause');

// Attach our doAnimations() function to the
// carousel's slide.bs.carousel event
$myCarousel.on('slide.bs.carousel', function (e) {
  // Select the elements to be animated inside the active slide
  var $animatingElems = $(e.relatedTarget)
                        .find("[data-animation ^= 'animated']");
  doAnimations($animatingElems);
});
```

There's quite a lot going on in the chunk of code above, so let's break it down.

## Looking into the `doAnimations()` Function

The `doAnimations()` function performs the tasks described below.

It starts by caching a string in a variable containing the name of the `animationend` event. This event tells us, you might have guessed, when each animation ends. We need this bit of information because each time the animation ends, we remove the Animate.css classes. If we fail to do this, the carousel captions will be animated only once, that is, just the first time the carousel shows a particular slide.

```
var animEndEv = 'webkitAnimationEnd animationend';
```

Next, our function loops over each element that we want to animate and extracts the value of the `data-animation` attribute. As you recall, this value contains the Animate.css classes that we need to add to our element in order to animate it.

```
elems.each(function () {
  var $this = $(this),
      $animationType = $this.data('animation');
  // etc...
});
```

Finally, the `doAnimations()` function dynamically adds the Animate.css classes to each element that we want to animate. It also attaches an event listener that fires only once, when the animation ends. After the animation ends, the Animate.css classes that we just added are removed. This ensures that the next time the carousel comes back to the same slide, the animations take place again (try removing this bit of code and you'll see the animations happen only once).

```
$this.addClass($animationType).one(animEndEv, function () {
  $this.removeClass($animationType);
});
```

## Animating the First Carousel Caption

As soon as the page loads in the browser, we animate the content inside the first slide like so:

```
var $firstAnimatingElems = $myCarousel.find('.item:first')
                           .find("[data-animation ^= 'animated']");
doAnimations($firstAnimatingElems);
```

Int this code, we begin by finding the first slide. From there, we select the content we want to animate inside the caption by using the values of the `data-animation` attribute starting with *animated*. We then use the piece of data thus obtained as an argument in our `doAnimations()` function and let the function do its job.

## Pausing the Carousel

After the content in the first slide has performed its animations, we pause the carousel.

```
$myCarousel.carousel('pause');
```

This is a feature of the Bootstrap Carousel designed to stop it from cycling. You're free not to pause the carousel, but you run the risk of annoying your website visitors.

In this case, I recommend you make sure the carousel doesn't cycle to the next slide until all animations on the active slide have run their course. You can control this by setting the "interval" option in the initialization code as follows:

```
$myCarousel.carousel({
  interval: 4000
});
```

I my opinion, an infinitely looping carousel with captions jumping around each time a slide comes into view is far from ideal.

## Animating the Carousel Captions as They Slide

Animating the carousel captions as each slide becomes visible requires the steps described below.

First we attach an event listener to the `slide.bs.carousel` event. According to the Bootstrap Carousel documentation:

> This event fires immediately when the slide instance method is invoked.

```
$myCarousel.on('slide.bs.carousel', function (e) {
  // do stuff...
});
```

Next we select the **active slide**, that is, the slide currently in view, and from there we find the elements we wish to animate. The code below uses the `.relatedTarget` property of the `slide.bs.carousel` event to get hold of the active slide.

```
var $animatingElems = $(e.relatedTarget).find("[data-animation ^= 'animated']");
```

Finally, we call our `doAnimations()` function, passing our selection of the elements to be animated as an argument.

```
doAnimations($animatingElems);
```

The full demo is shown in the CodePen below.

Bootstrap Carousel with Animate.css by SitePoint

## Conclusion

As many of you probably know, Carousels do have issues that developers need to take into consideration.

With the Bootstrap Carousel component adding a slider or carousel to a web page is just a matter of entering the appropriate HTML markup.

In this article, I've shown how to add some extra pizzazz to the basic Bootstrap Carousel component with a few lines of jQuery and the Animate.css library. However, any other similar CSS library, or coding the CSS3 animations from scratch, will do just as well.

# Bootstrap Sass Installation and Customization

By Reggie Dawson

Bootstrap is a popular, open source framework. Complete with pre-built components it allows web designers of all skill levels to quickly build a site.

The only drawback I can find to Bootstrap is that it is built on Less. Less is a CSS preprocessor, and although I could learn Less, I prefer Sass. Normally the fact that it is based on Less would exclude me as a user of Bootstrap, as I do no write plain CSS anymore. Fortunately Bootstrap now comes with a official Sass port of the framework, bootstrap-sass. If you are not familiar with Bootstrap implementing the Sass version can be a little tricky. In this article I will show you how to configure and customize Bootstrap with Sass.

It is worth noting that the upcoming version of Bootstrap, Bootstrap 4, will use Sass by default. Until it's released, unless you want to use the alpha version versions of bootstrap 4, you'll need to use bootstrap-sass.

## Installation

There are multiple ways to obtain and install bootstrap-sass.

### Download

You can download bootstrap-sass from the Bootstrap download page. Once you have it downloaded extract the contents of the file to the folder you are going to create your project in.

## Compass

If you are using Compass then you'll have Ruby installed. With Ruby installed we can use gems, in this case the bootstrap-sass gem. To install:

```
gem install bootstrap-sass
```

If you have an existing Compass project and want to add bootstrap-sass, run this command:

```
compass install bootstrap -r bootstrap-sass
```

If you want to start a new Compass project with bootstrap-sass use:

```
compass create my-new-project -r bootstrap-sass --using bootstrap
```

## Bower

We can also install it with the package manager Bower. To me this option is the best as the other options install a lot of 'fluff' that will confuse someone not familiar with Bootstrap. To install with Bower make sure you are in the folder where you want to create your project and run:

```
bower install bootstrap-sass
```

## Configuration

Once we have installed our desired version of bootstrap-sass we need to configure our project. The type of install we performed will determine where the files we need are located.

## Download

The download includes a lot of folders that we will not need if we aren't using Rails. The only folder we need is the assets folder. We can copy the contents of this folder to the root of our project or use it as is. If you intend to use Javascript components you will have to manually download jQuery.

## Compass

Using the Compass version creates a `styles.scss` and `_bootstrap-variables.scss` file. Folders for fonts, javascript, sass, and stylesheets are created. The important thing to note is that `styles.scss` imports Bootstrap as a whole, there is no way to pick and choose what Bootstrap components you want to use. You will also have to download jQuery.

## Bower

An install from Bower includes everything you need for Bootstrap, even jQuery. All components installed are located in the `bower_components` directory.

## Setup

Once we have Bootstrap installed we need to setup our project to use it. The first thing we want to do is create folders for sass and stylesheets (the Compass setup has already created these). The sass folder will hold our scss files while stylesheets will be where compiled css will be stored. After that create a file named app.scss inside the sass folder. If we are using Compass this file has already been created as `styles.scss`.

The `app.scss` file (or `styles.scss` in Compass) is used to import bootstrap components. For example:

*Download*

```
@import "bootstrap-sass-3.3.4/assets/stylesheets/bootstrap";
```

*Compass*

```
@import "bootstrap";
```

*Bower*

```
@import  "../bower\_components/bootstrap-sass/assets/stylesheets/bootstrap";
```

The next thing we want to do is navigate to the Bootstrap folder and find the stylesheets folder. Inside of stylesheets there is a bootstrap folder. Copy the `_variables.scss` file to your sass folder. Rename the file to `_customVariables.scss`. Add an import statement for `_customVariables.scss` to `app.scss`. Make sure to import `_customVariables.scss` first for reasons I will explain in a moment.

If you are using Compass you can skip this step as the `_bootstrap-variables` file serves the same purpose. The file has already been imported into `styles.scss` for you. If you are using Compass with Bower it is advisable to import `bootstrap-compass.scss`.

The last import is an optional `_custom.scss` file. Many people will include custom css rules directly after their import statements or in their `app.scss` file, but I prefer to separate any custom rules into their own partial. At any rate our app.scss should have three import statements now (or four if using Compass).

```
@import "customVariables";
@import "../bower\_components/bootstrap-sass/assets/stylesheets/bootstrap";
@import "../bower\_components/bootstrap-sass/assets/stylesheets/bootstrap-compass";
@import "custom";
```

Notice we import our `_customVariables.scss` file first. The reason being is all of Bootstrap's variables are set to default! values,so we need to override these values by importing our variables first.

## Customize

When we edit variables it is advisable to make a copy of the original and change the copy. After copying, comment out the original variable. That way we can go back to what it was previously set to in case we don't like the result. For example lets say we wanted to change the base font size to 20px.

Firstly we will look in our `_customVariable.scss` file. The variables are broken down by section, we are looking for the Typography section. There we want the `$font-size-base:14px !default;` variable. Copy and paste and comment out the original. After that it is as simple as changing the value to 20px.

```
$font-size-base:14px !default;
$font-size-base:20px !default;
```

As you can see I have commented out the original variable and changed the copy.

When trying to customize Bootstrap bear in mind there are a lot of variables to deal with. When looking for a variable to change it is advisable to make full use of your text editors search feature. It is also a good idea to look over the `_customVariables.scss` file and get familiar with the variables present.

Another effective method for finding what variables you need to change is to look at the raw SCSS files that make up Bootstrap before they are compiled. From there we can see what variables are used in that module. For example lets say I am not happy with the color of the

`.navbar-default` element. Instead of me trying to figure out what variable I need to change I can look inside of the `_navbar.scss` file. I scroll down (or use my search function) to find a reference to a color variable.

```
// Default navbar
.navbar-default {
    background-color: $navbar-default-bg;
    border-color: $navbar-default-border;

    .navbar-brand {
        color: $navbar-default-brand-color;
        &:hover,
        &:focus {
        color: $navbar-default-brand-hover-color;
        background-color: $navbar-default-brand-hover-bg;
    }
}
```

From looking at this rule I determine the variable I need to change is `$navbar-default-bg`. I would then go into my `_customVariables.scss` and copy/comment out original variable and create my own.

When using bootstrap-sass you also have the advantage of being able to use and look at the mixins included with Bootstrap. My first article for Sitepoint was about 5 useful mixins in Bootstrap, so it's no secret I am a fan of the Bootstrap mixins. Not only will they help with understanding how Bootstrap fits together, they may actually help you build your site. For example looking at `@mixin make-row`:

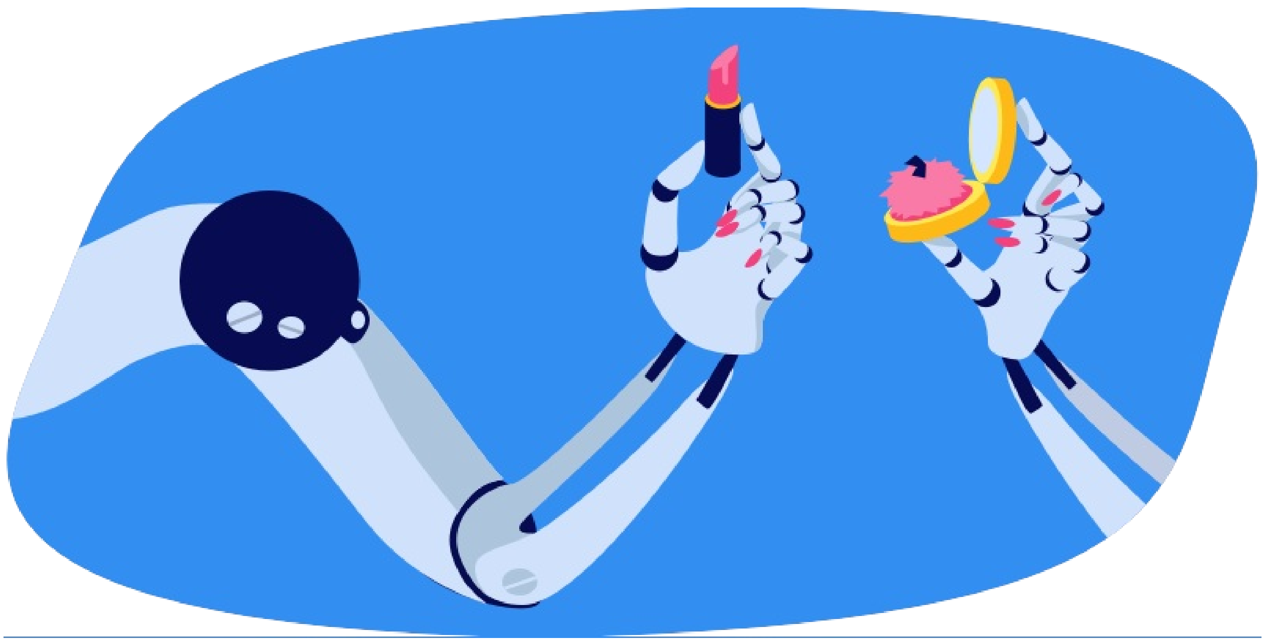
```
@mixin make-row($gutter: $grid-gutter-width) {
    margin-left:  ($gutter / -2);
    margin-right: ($gutter / -2);
    @include clearfix;
}
```

From this mixin we can see what variables affect our row. We now know we can alter the `$grid-gutter-width` to make changes to the margins of a row. Also we see that the `clearfix` mixin has been included. We can also look that over to see what that affects.

## Conclusion

Using Bootstrap can be complicated especially for someone that is not familiar with the framework. With the methods I demonstrated you should be able to get Bootstrap setup with Sass and customize the framework with ease. Finding the variables you need to change should be more manageable now that you know what to look for.

Just remember your text editors search functions are your friend, and when in doubt looking at the mixins can help.



# Using Sass to Semantically @Extend Bootstrap

By Brad Barrow

Bootstrap provides a quick and easy way to create a scaffold for your latest site or app whether you're a novice or a professional developer. For this reason, more and more developers are including the framework in their personal toolbox.

There is, however, a dark side to Bootstrap, in that it makes it incredibly easy to write cluttered, non-semantic and non-reusable markup that will render correctly across all browsers.

In this article, I'll explain how you can use Bootstrap in a more robust and semantic way. Bootstrap's latest version shipped with an official Sass port of the framework so we'll use one of Sass's most powerful features to achieve this: the @extend directive.

## Some Basics on Semantics

First, let's cover what we mean when we say that we want to make things more "semantic". HTML documents are intended to be descriptive of their contents from an information hierarchy perspective. One should be able to read them and know what they are about, *not* how they will look.

With the rise in popularity of CSS and especially CSS frameworks, HTML is often written with a CSS sheet in mind rather than a reader/developer.

It's not uncommon to see HTML markup like this in the source code of modern websites:

```html
<div class="row">
    <div class="col-md-4">
        Name:
    </div>
    <div class="col-md-4">
        John
    </div>
    <div class="col-md-4">
        Smith
    </div>
</div>
<div class="row">
    <div class="col-md-12">
        <p>
            "I enjoy writing code in my spare time."
        </p>
    </div>
</div>
```

Bootstrap's CSS files know exactly what to do with that. Two rows, one split into thirds and the other full width. As developers who are familiar with Bootstrap, that's easy enough for us to figure out, but what if you'd never used the framework before? What is this snippet of HTML for? What does the information pertain to? The markup isn't revealing anything practical.

Semantic markup is code that describes it's content rather than its appearance. We could write the code above semantically as follows:

```html
<div class="author-name">
    <label class="author-nameLabel">
        Name:
    </label>
    <span class="author-nameFirst">
        John
    </span>
    <span class="author-nameLast">
        Smith
    </span>
</div>
<div class="author-bio">
    <p>
        "I enjoy writing code in my spare time."
    </p>
</div>
```

It’s not all that different but it immediately tells us what each element is for and how each one might relate to other elements. What we’ve lost is the knowledge of how this will be presented visually. But that’s not the purpose of HTML after all.

## How is Bootstrap Supposed To Style That?

You might be wondering how you’re going to use Bootstrap without any of its built-in class names in your markup? Well, we can leverage the power of Sass’s `@extend` functionality to solve this problem.

From Sass’s documentation:

> `@extend` works by inserting the extending selector anywhere in the stylesheet that the extended selector appears.

This means we can use our semantic selectors and extend Bootstrap’s selectors to get all of its goodness. Here’s how:

```
/* Import bootstrap-sass so that we have access to all of its selectors */
@import "bootstrap";

/* Author Bio and Author Name are just Bootstrap .row elements */
.author-bio,
.author-name {
  @extend .row;
}

/* Author nameLabel, nameFirst and nameLast need
   to be a third of their container's width */
.author-nameLabel,
.author-nameFirst,
.author-nameLast {
   @extend .col-md-4;
}

/* The paragraph inside the author's bio should be full width */
.author-bio p {
    @extend .col-md-12;
}
```

Through the magic of `@extend`, our selectors are slotted in to the compiled stylesheets alongside the Bootstrap selectors they extend. As you’ll see by examining the compiled code, our selectors have all the exact same properties as the classes they’re extending.

For example, among other things, you should see this in the compiled CSS:

```
* one of the compiled rule sets */
.col-md-4, .author-nameLabel,
.author-nameFirst,
.author-nameLast {
  width: 33.3333333333%;
}
```

## Reusability

Another great thing about this method is that it creates readable and reusable components. For example, you don't have to rebuild the author component out of rows and columns each time you need to use it. Instead, you have sensible names for each part of the component, making it easy to construct. You can also rely on Bootstrap to make sure it's presented uniformly across your site.

## Portability

While Boostrap is one of the best frameworks available, the time may come when you want to move your site to a different framework or even write your own. With the above outlined method you can do so easily because your markup is cleanly decoupled from your CSS.

## Fill Your Toolbelt

If you haven't yet tried the Sass version of Bootstrap, you're in for a treat. It's easy to get started on their github page.

Sass isn't the only thing at play here. If you truly want next-level semantics in your HTML and CSS, I recommend adopting a naming convention such as BEM or SMACSS to keep your selectors standardized and easy to remember.

## Go Forth And Extend

Bootstrap gives us an incredibly powerful set of styles – but it's all too easy to cobble them together into something presentable at the loss of quality markup. With Sass's `@extend` directive, you're free to write markup that speaks clearly both to its contents and to the developers reading it.

# Bootstrap Alpha: Super Smart Features to Win You Over

By Maria Antonietta Perna

After months of anticipation, anxious tweets asking for the disclosure of a release date, and a few scattered scraps of news by Mark Otto and Jacob Thornton, having the effect of intensifying rather than quenching our curiosity, Bootstrap 4 alpha is out.

As a designer, I love crafting my own CSS. However, I confess, I find Bootstrap such a well thought-out and strongly supported front-end framework that I've immensely enjoyed using it, both to build my projects and to learn more about writing better, modular CSS.

After the much awaited news, as you can probably guess, I downloaded the source files for Bootstrap 4 and spent some time going back and forth between reading the docs and digging into the code to find out more.

I expect the alpha release of Bootstrap 4 will undergo a number of changes in the coming weeks, even months. The curious among you, can keep an eye on the issues section of the project's GitHub repository.

However, the features I'm going to list here are more like broad coding principles and practices that keep improving on each new release of the framework. That's why I think it's likely they're here to stay. If anything, they can only get better.

Here they are. I hope you find them awesome too!

## New Interactive Documentation

The Bootstrap documentation has been exemplary since the framework's early days. It's always had the crucial role of being a **living document**, that is, a tool in sync with the collaborative effort of building the framework and communicating it to others:

> Abstracting and documenting components became part of our process for building this one tool and Bootstrap in tandem. *Mark Otto in 2012*

Mark himself is quite a fan of great documentation. His Code Guide by @mdo is evidence of his approach to high quality documentation as being part and parcel of outstandingly coded projects.

The documentation for version 4 has been rewritten from scratch using Markdown. Understandably, given the alpha stage of this version at the time of writing, it's still a work in progress.

The Bootstrap docs …

- are a pleasure to navigate, both using the traditional sidebar navigation and the brand new **search form**.
- structure information in a logical manner; content is never overwhelming or confusing.
- include instructions and how-tos covering all areas of the framework, from different ways of installing Bootstrap to using each component and dealing with browser quirks.

If you take the time, you'll soon find a few valuable nuggets scattered throughout various sections of the docs. For instance, dealing with over-sized SVG images that use the `.img-responsive` class in IE9-10, **accessibility** best practices, enabling the mq4-hover-shim to fix sticky `:hover` styles on mobile devices, and much more.

Finally, if you'd like to run the Bootstrap docs locally on your computer, install Jekyll, a great website building tool, and follow these instructions.

## Top-notch Modular Architecture

Bootstrap has often been the target of complaints about code bloat, too opinionated CSS styling, and a profuse quantity of components. The good news is: Bootstrap 4 has both simplified and further modularized its structure.

To begin with, some components have been eliminated altogether: the Glyphicons icon library is not bundled with the framework any more; panels, wells, and thumbnails are replaced by the Cards component. Also, all CSS reset/normalize code and basic styling are now dealt with in a single brand new module called Reboot.

Further, more than ever before, using Bootstrap now feels like assembling and arranging Lego blocks in different ways. Here are some examples to clarify what I mean.

## Opt-in Modules

You can quickly control the application of complex CSS features by toggling the value of a Boolean SCSS variable. Grab your copy of the Bootstrap 4 alpha release source files, open `_variables.scss` in a code editor, and find the following snippet (at about line 46):

```
$enable-flex:                false !default;
$enable-rounded:             true !default;
$enable-shadows:             false !default;
$enable-gradients:           false !default;
$enable-transitions:         false !default;
```

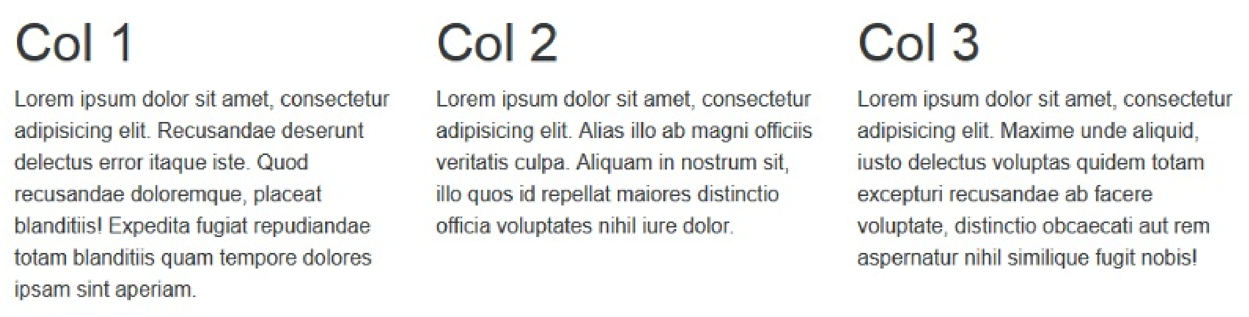
Toggling the value of one of the above variables from `true` to `false` or vice versa, and compiling the code, will enable or disable the corresponding CSS property in your project.

Let’s have a go at turning a traditional Bootstrap grid into a cool Flexbox-powered grid.

Here’s the HTML for a regular Bootstrap three-column grid:

```
<section class="container">
  <div class="row">
    <article class="col-md-4 col-sm-6">
      <header>
        <h1>Col 1</h1>
      </header>
      <p>Lorem ipsum dolor sit amet, consectetur adipisicing elit.</p>
    </article>
    <article class="col-md-4 col-sm-6">
      <!-- Same code as above -->
    </article>
    <article class="col-md-4 col-sm-6">
      <!-- Same code as above -->
    </article>
  </div><!-- .row -->
</section><!-- .container -->
```

This is what it looks like in the browser:



Col 1

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Recusandae deserunt delectus error itaque iste. Quod recusandae doloremque, placeat blanditiis! Expedita fugiat repudiandae totam blanditiis quam tempore dolores ipsam sint aperiam.

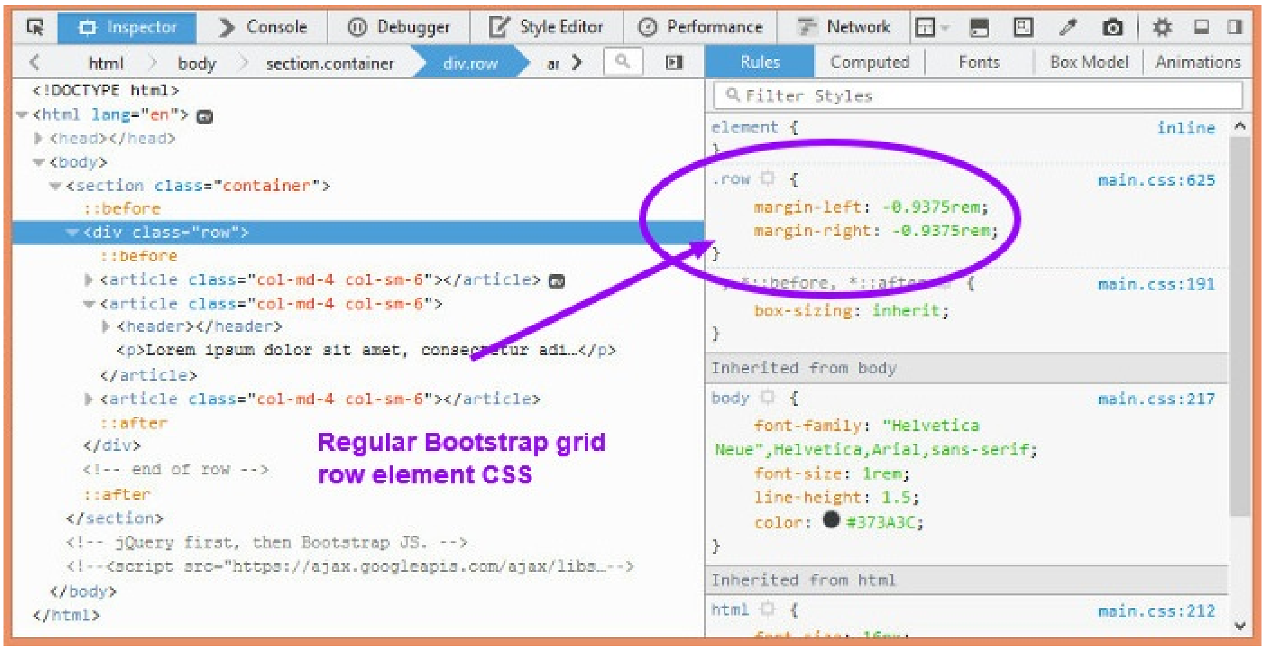
Col 2

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Alias illo ab magni officiis veritatis culpa. Aliquam in nostrum sit, illo quos id repellat maiores distinctio officia voluptates nihil iure dolor.

Col 3

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Maxime unde aliquid, iusto delectus voluptas quidem totam excepturi recusandae ab facere voluptate, distinctio obcaecati aut rem aspernatur nihil similique fugit nobis!

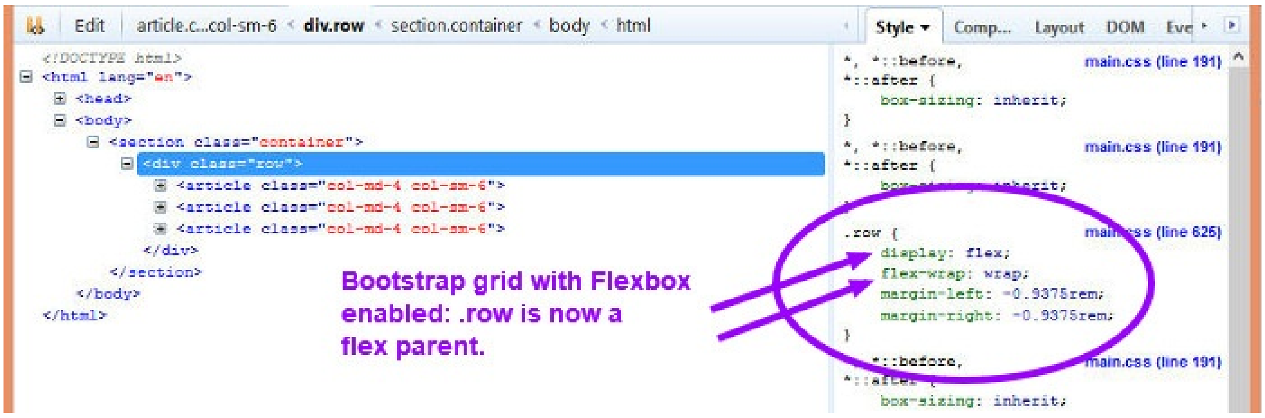
Look into the source code using the developer tools of your browser of choice. The CSS rules corresponding to the `.row` class should look like the image below:



```
<!DOCTYPE html>
<html lang="en">
  <head></head>
  <body>
    <section class="container">
      ::before
      <div class="row">
        ::before
        <article class="col-md-4 col-sm-6"></article>
        <article class="col-md-4 col-sm-6">
          <header></header>
          <p>Lorem ipsum dolor sit amet, consectetur adi…</p>
        </article>
        <article class="col-md-4 col-sm-6"></article>
        ::after
      </div>
      <!-- end of row -->
      ::after
    </section>
    <!-- jQuery first, then Bootstrap JS. -->
    <!--<script src="https://ajax.googleapis.com/ajax/libs…-->
  </body>
</html>
```

```
element {                                   inline
}
.row {                                      main.css:625
    margin-left: -0.9375rem;
    margin-right: -0.9375rem;
}
*::before, *::after {                       main.css:191
    box-sizing: inherit;
}
Inherited from body
body {                                      main.css:217
    font-family: "Helvetica
Neue",Helvetica,Arial,sans-serif;
    font-size: 1rem;
    line-height: 1.5;
    color: ● #373A3C;
}
Inherited from html
html {                                      main.css:212
```

Next, set the `$enable-flex` variable to `true` , compile and refresh the browser. Although the browser display is the same as before, the CSS is different. The `.row` class turns its element into a **flex container** with the `flex-wrap` property set to `wrap` . This ensures that child elements exceeding the container's width wrap to the next row.



```
<!DOCTYPE html>
<html lang="en">
  <head>
  <body>
    <section class="container">
      <div class="row">
        <article class="col-md-4 col-sm-6">
        <article class="col-md-4 col-sm-6">
        <article class="col-md-4 col-sm-6">
      </div>
    </section>
  </body>
</html>
```

```
*, *::before,                      main.css (line 191)
*::after {
    box-sizing: inherit;
}
*, *::before,                      main.css (line 191)
*::after {
    box-sizing: inherit;
}
.row {                             main.css (line 625)
    display: flex;
    flex-wrap: wrap;
    margin-left: -0.9375rem;
    margin-right: -0.9375rem;
}
*::before,                         main.css (line 191)
*::after {
    box-sizing: inherit;
```

## Ready-made Light-weight Versions

Besides `bootstrap.scss` , which includes the entire framework, you'll also find the following files: `bootstrap-flex.scss` , `bootstrap-grid.scss` , and `bootstrap-reboot.scss` .

Each of these files includes only **selected portions** of Bootstrap. If you don't need the full-blown framework in your project, this is a great head-start: just compile one of the light-weight options and you're good to go.

Corresponding cut-down compiled packages will be made available for download from the Bootstrap 4 docs page.

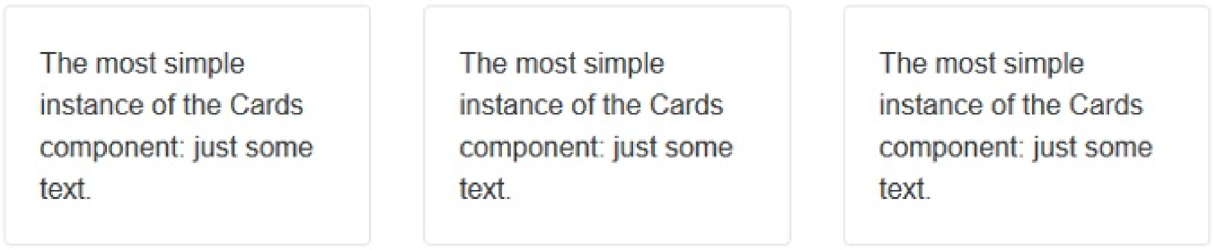
## Reusable Components

You can skin and modify components by mixing and matching a few classes. For instance, the brand new Cards component is a great example of this versatility in action.

Here's all the HTML you need for the simplest instance of this component:

```
<div class="card">
  <div class="card-block">
    <p class="card-text">Just some text.</p>
  </div>
</div>
```

Below is how the code looks in the browser in a three-column layout:



The most simple instance of the Cards component: just some text.

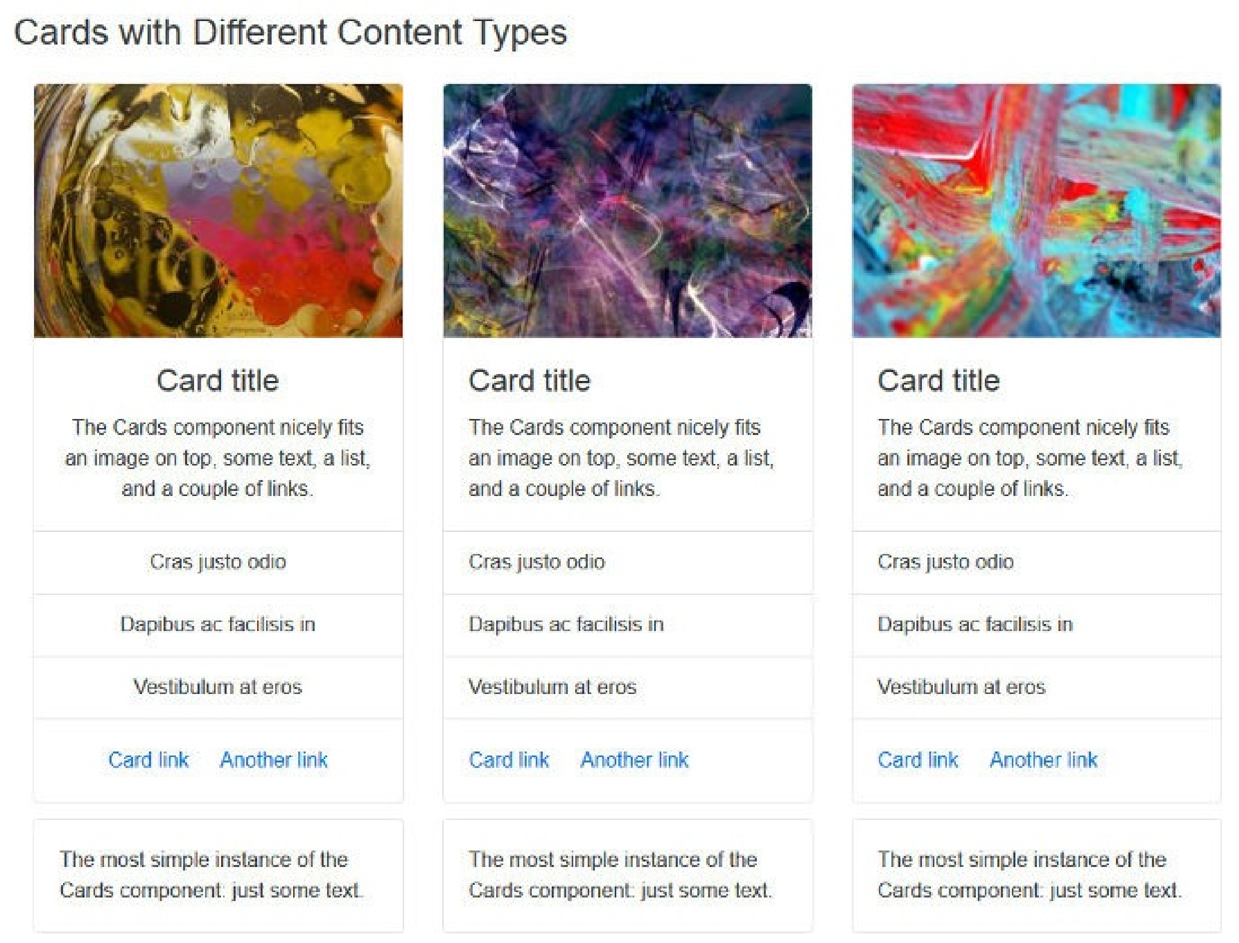
The most simple instance of the Cards component: just some text.

The most simple instance of the Cards component: just some text.

This flexible component easily adapts to a variety of content types. Here's an instance that includes an image, text, links, and a list:

```
<div class="card">
  <img class="card-img-top" src="http://lorempixel.com/290/200/abstract/8" alt="">
  <div class="card-block">
    <h4 class="card-title">Card title</h4>
    <p class="card-text">
      The Cards component nicely fits an image on top,
      some text, a list, and a couple of links.
    </p>
  </div>
  <ul class="list-group list-group-flush">
    <li class="list-group-item">Cras justo odio</li>
    <li class="list-group-item">Dapibus ac facilisis in</li>
    <li class="list-group-item">Vestibulum at eros</li>
  </ul>
  <div class="card-block">
    <a href="#" class="card-link">Card link</a>
    <a href="#" class="card-link">Another link<</a>
  </div>
</div>
```

And below is the result in the browser in a three-column layout:


Cards with Different Content Types

Card title

The Cards component nicely fits an image on top, some text, a list, and a couple of links.

Cras justo odio

Dapibus ac facilisis in

Vestibulum at eros

Card link Another link

The most simple instance of the Cards component: just some text.

Card title

The Cards component nicely fits an image on top, some text, a list, and a couple of links.

Cras justo odio

Dapibus ac facilisis in

Vestibulum at eros

Card link Another link

The most simple instance of the Cards component: just some text.

Card title

The Cards component nicely fits an image on top, some text, a list, and a couple of links.

Cras justo odio

Dapibus ac facilisis in

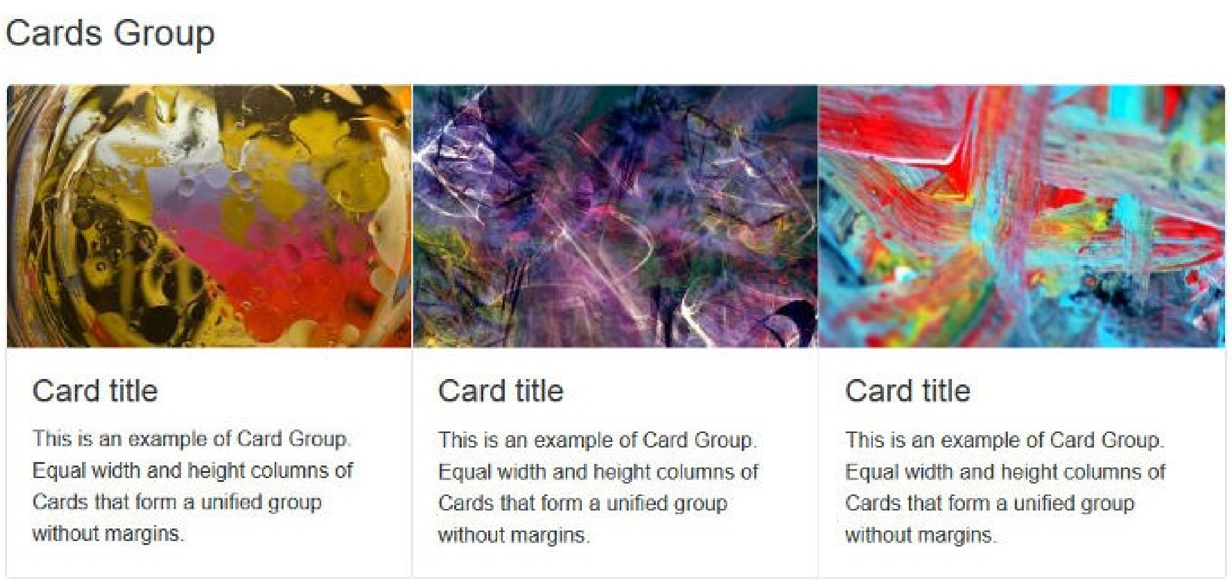
Vestibulum at eros

Card link Another link

The most simple instance of the Cards component: just some text.

You can also arrange cards in touching equal width and height columns by wrapping them in a `.card-group` container:

```
<div class="card-group">
  <div class="card">
    <!-- card code here -->
  </div>
  <div class="card">
    <!-- card code here -->
  </div>
  <div class="card">
    <!-- card code here -->
  </div>
</div>
```



Card title

This is an example of Card Group. Equal width and height columns of Cards that form a unified group without margins.

Card title

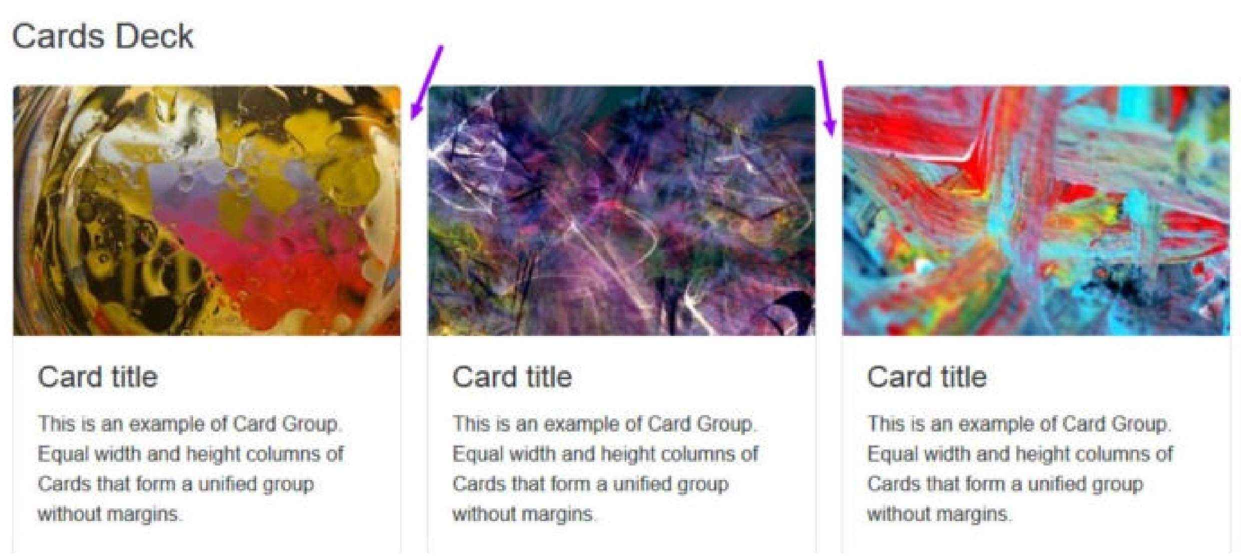
This is an example of Card Group. Equal width and height columns of Cards that form a unified group without margins.

Card title

This is an example of Card Group. Equal width and height columns of Cards that form a unified group without margins.

Alternatively, you can group Cards having equal width and height columns with margins, using `.card-deck-wrapper` and `.card-deck` as follows:

```
<div class="card-deck-wrapper">
  <div class="card-deck">
    <div class="card">
      <!-- card code here -->
    </div>
    <div class="card">
      <!-- card code here -->
    </div>
    <div class="card">
      <!-- card code here -->
    </div>
  </div>
</div>
```


Cards Deck

Card title

This is an example of Card Group. Equal width and height columns of Cards that form a unified group without margins.

Card title

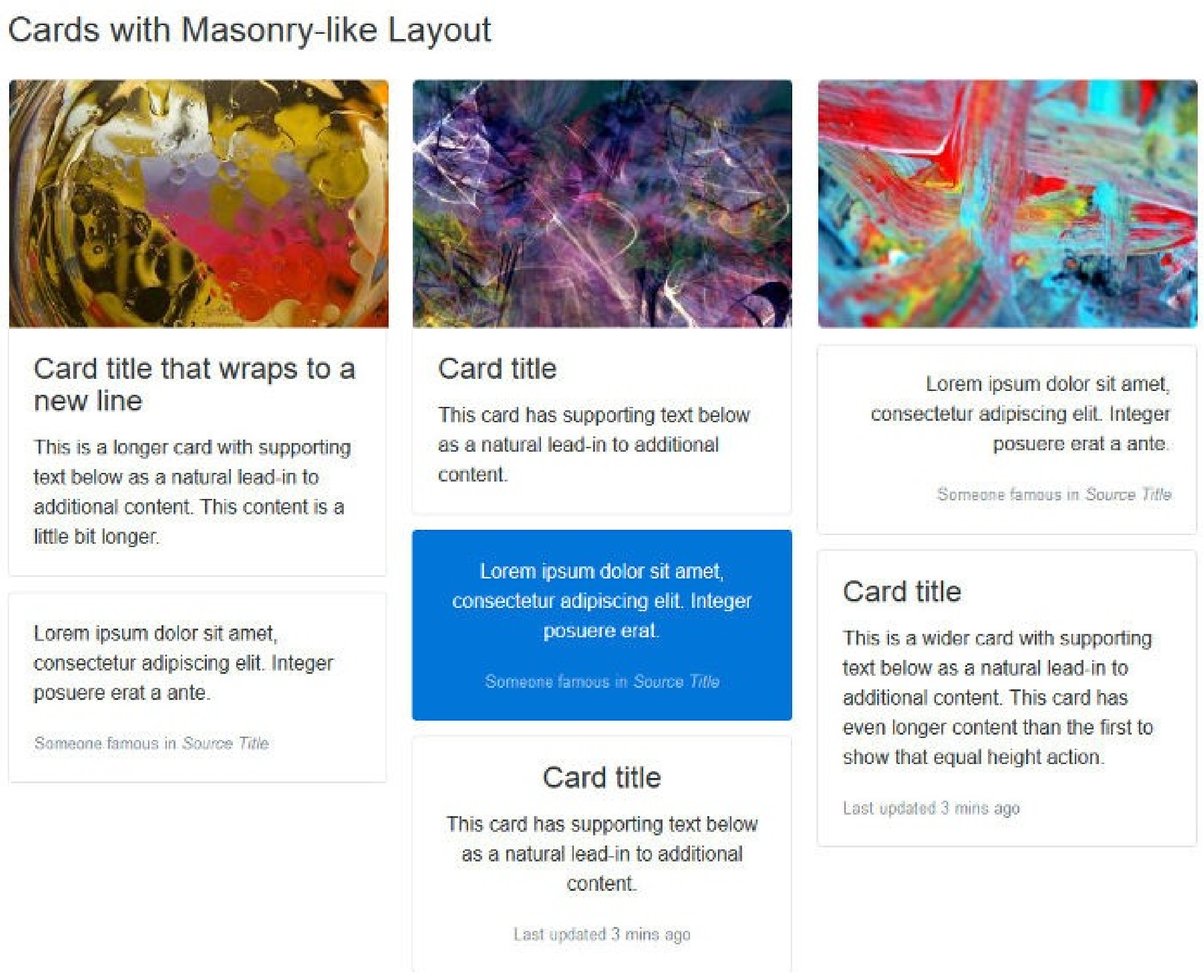
This is an example of Card Group. Equal width and height columns of Cards that form a unified group without margins.

Card title

This is an example of Card Group. Equal width and height columns of Cards that form a unified group without margins.

Another cool thing you can do with Cards is build a Masonry-like layout. Just wrap the cards in a container with the `.card-columns` class and leave the rest to Bootstrap.

```
<div class="card-columns">
  <div class="card">
    <!-- card code here -->
  </div>
</div>
```


Cards with Masonry-like Layout

Card title that wraps to a new line

This is a longer card with supporting text below as a natural lead-in to additional content. This content is a little bit longer.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Someone famous in *Source Title*

Card title

This card has supporting text below as a natural lead-in to additional content.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat.

Someone famous in *Source Title*

Card title

This card has supporting text below as a natural lead-in to additional content.

Last updated 3 mins ago

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer posuere erat a ante.

Someone famous in *Source Title*

Card title

This is a wider card with supporting text below as a natural lead-in to additional content. This card has even longer content than the first to show that equal height action.

Last updated 3 mins ago

To dive into the details of the Masonry-like Cards layout, as well as exploring further what you can do with Cards, check out the Cards docs.

Here I've offered only a few examples of Bootstrap's modular architecture. I think these suffice to show how flexibility and extensibility are built into the framework as a whole, which makes it fun and convenient to use.

## Easier Scaling Across Screen Sizes

Since version 3, Bootstrap has introduced a **mobile-first** approach to web design, i.e., start developing for smaller screens first and progressively add or adjust features as you target larger screens. Version 4 makes further improvements towards adaptive web design by taking the following steps.

### The Move to `rem`

Bootstrap 4 replaces most instances where the absolute unit of measurement in `px` was applied in the earlier version with relative `rem` and, for media queries, `em` units. The goal is to have all elements on a web page harmoniously scale with the screen size.

Let's have a look at how Bootstrap sets the global `font-size`. Start by opening `_reboot.scss` in a code editor and find the following snippet (about line 60):

```
html {
  // Sets a specific default `font-size` for user with `rem` type scales.
  font-size: $font-size-root;
  /* etc... */
}

body {
  // Make the `body` use the `font-size-root`
  font-size: $font-size-base;
  /* etc... */
}
```

If you dig into `_variables.scss`, you'll see that `$font-size-root` is set to `16px` and `$font-size-base` is set to `1rem`. This means that dividing any measurement in `px` by 16, you'll come up with the corresponding `rem` measurement, e.g., to get the corresponding `rem` measurement for `40px`, perform this operation: *40 / 16 = 2.5*.

Most importantly, this means it's easier to build web pages where all elements proportionally scale up or down with the screen size without messing up your beautiful design.

## Here Comes the Extra-large Breakpoint

The introduction of the new **extra large breakpoint** for the grid system further helps building layouts that scale well across different screen sizes.

This breakpoint is applied using the `.col-xl-` class and is triggered on screen sizes from `75em` upwards.

## Global Margins Reset and Utility Spacer Classes

Forcing consistent spacing between elements in a design is something most front-end developers, including myself, obsess over. It's a demanding task and the plethora of screen resolutions available doesn't make the job easier.

To help us keep both vertical and horizontal spacing between elements under tight control, Bootstrap 4 resets `margin-top` to `0` while keeping a consistent `margin-bottom` value on all elements.

Further, the framework offers an impressive number of utility classes to make it easier for us to adjust margins and padding at a more granular level across varying screen sizes.

## Conclusion

At the dawn of the Bootstrap 4 alpha release, I've introduced three broad features that in my view make this front-end framework really stand out:

- Great documentation
- Mega Lego type architecture
- Easier scaling across devices

Did you notice I didn't mention Bootstrap's move from Less to Sass? Or the rewrite of all JavaScript plugins in ECMAScript 6?

I consider these to be more like indications of Bootstrap staying current and taking advantage of the latest tools, rather than features integral to the framework itself.