Notes from the Bootsrap edX class

Taken in October, 2015 using Bootstrap 3.3.5 and jQuery 2.1.4

GITHUB REPO FOR THE COURSE: https://github.com/MicrosoftLearning/Bootstrap-edX

COMMON BOOTSTRAP STARTING CLASSES

When putting together the layout for a page, there are a few classes that you typically use to start. The first two help define how much space the content will take on the overall page, while the last will allow you to highlight content, such as the site's title and introductory information.

container	The container class places content inside of a horizontal container. The size of the container will vary based on screensize. The container will stay the same width for that particular screensize. This helps developers size content appropriately for different devices.
container-fluid	The container-fluid class places content inside of a container that will always be the width of the screen. This is helpful for scenarios where the content to be displayed must use the entire browser window.
jumbotron	The jumbotron class is a common class for displaying titles for different sections of a website, or the landing page. The jumbotron typically has a highlighted background and an increased fontsize.

GRID NOTES

- Always 12 columns
- Smaller sizes set the default for larger sizes
- **col-sm-** would set the size for small, medium and large screens unless overridden by a setting specific to one of the larger sizes

- There is a 30px "gutter", or empty space between each column (15px on each side of the column)
- The gutter on the outside of the container varies based on the current screen size
- Extra small devices always use the entire width of the screen

Grid System for fixed container (class=”container”)

	Extra small devices e.g. phones (< 768 px)	**Small devices e.g. tablets (≥ 768 px)**	**Medium devices e.g. laptops (≥ 992 px)**	**Large devices e.g. desktops (≥ 1200 px)**
Grid behavior	Horizontal	Collapse to start, horizontal above breakpoints		
Container width	None (auto)	750px	970px	1170px
Column class prefix	col-xs-	col-sm-	col-md-	col-lg-
Column width	Auto	60px	75px	95px
Gutter width	30px (15px on each side of the column)			

The screens sizes that Bootstrap uses are not random, but rather based on research on common device sizes. While you can change the screen sizes, because Bootstrap does use CSS, you want to use the defaults provided.

GRID SIZE CLASSES

Structure of Bootstrap classes for sizing grid content

- **col**
    - Short for column
    - Required prefix
- **size abbreviation**
    - xs for Extra small
    - sm for Small
    - md for Medium
    - lg for Large

  Each size applies to itself and larger sizes
- **Number of columns**
    - Integer to represent the number of columns

EXAMPLES

Let's take a look at a couple of sample Bootstrap content sizing classes:

- col-md-4 would indicate 4 columns for medium and large screens
- col-sm-6 would indicate 6 columns for small, medium and large screens
- col-xs-2 would indicate 2 columns for extra small, medium and large screens

Remember that screen size rules apply for the specified screen size as well as those larger than that size, unless overriden. So the following combination would indicate 2 columns for extra small screens, 6 columns for small screens, and 4 for medium and large screens.

```
<div class='col-xs-2 col-sm-6 col-md-4'>Content</div>
```

BOOTSTRAP AND VISIBILITY

Bootstrap offers you the ability to show or hide content based on both screen size and printing state. The syntax is, as with much of Bootstrap, **option-screen**.

The two options are either hidden or visible. Both of the options will change the default for all other screen sizes and print. In other words, **hidden-sm** will make the content hidden for small screens, but visible on all others; **visible-sm** will make the content visible on small screens, but visible on all others.

One thing to note about **visible-print**. **visible-print**, while perfect for a demonstration, as you saw in the video, does have a couple of issues. First, it is deprecated, meaning it will eventually be removed from Boostrap. Second, while it will be visible only when printing, all other content currently being displayed on the page will also be visible. If you want something to be hidden when printing, use **hidden-print**.

visible summary

Class	Extra small	Small	Medium	Large
visible-xs	Displayed			
visible-sm		Displayed		
visible-md			Displayed	

Class	Extra small	Small	Medium	Large
visible-lg				Displayed

hidden summary

Class	Extra small	Small	Medium	Large
hidden-xs		Displayed	Displayed	Displayed
hidden-sm	Displayed		Displayed	Displayed
hidden-md	Displayed	Displayed		Displayed
hidden-lg	Displayed	Displayed	Displayed	

NESTING COLUMNS

The Bootstrap grid system allows developers to lay out their pages using a system that's similar to tables, but uses CSS positioning instead of tables behind the scenes. One feature tables offer is the ability to create tables inside of a cell, so you can create complex layouts through what's frequently referred to as "nesting". The grid system offers the same capability.

Every row you create has 12 columns, regardless of where that row is created. If you create a section of content, and then add a row inside of there, that row has

12 columns.

Let's take the following HTML example.

```
<div class='row'>
  <div class='col-md-6'>Sample content here</div>
</div>
```

In that example, we would have created a section for content that is 6 columns wide, or half of the container. If we then added a new row into that space, replacing the sample content, we'd have a brand new 12 columns. However, those new columns would only take up half of the container, as they are inside of a section that is constraining it to 6 initial columns.

```
<div class='row'>
 <div class='col-md-6'>
    <div class='row'>
      <!-- 12 new columns here -->
      <!-- This will use half of the container -->
    </div>
  </div>
</div>
```

CONTROLLING PLACEMENT

The ability to resize content based on screen size is extremely powerful. It allows you to ensure that the most relevant information is readily available to the user on a page, regardless of the size of device.

But there's more to optimizing display for different device types than just changing the size. Sometimes you need to move content higher up on the page, such as a product name, or a "Buy it now!" button. In other scenarios, you need to push content down, such as detailed information, which isn't necessarily as important to someone just browsing your page. In fact, you might even need to just hide it all together.

Bootstrap offers you complete control over the size, and placement, of content, across all devices.

You can use three main options for controlling placement in Bootstrap. Each option has a syntax similar to sizing content. You'll start with **col**, followed by the

abbreviation for the screen-size, followed by the option, and then the number of columns. For example, you can move content to the right 6 columns on extra small screens by using **col-xs-push-6**. Here are the options available to you:

•**offset**

- Offset will instruct Bootstrap to skip the specified number of columns before placing content (pushes the content to the right). With offset, the skipped columns will be left blank (content cannot be pulled into them like they can with push).

•**push**

- Push will instruct Bootstrap to move content to the right a specified number of columns. With push, the columns that were left blank can be used by pulling content to the left.

•**pull**

- Pull will instruct Bootstrap to move content to the left a specified number of columns. With pull, the columns that were left blank can be used by pushing content to the right.

BOOTSTRAP AND VISIBILITY

Bootstrap offers you the ability to show or hide content based on both screen size and printing state. The syntax is, as with much of Bootstrap, **option-screen**.

The two options are either hidden or visible. Both of the options will change the default for all other screen sizes and print. In other words, **hidden-sm** will make the content hidden for small screens, but visible on all others; **visible-sm** will make the content visible on small screens, but visible on all others.

One thing to note about **visible-print**. **visible-print**, while perfect for a demonstration, as you saw in the video, does have a couple of issues. First, it is deprecated, meaning it will eventually be removed from Boostrap. Second, while it will be visible only when printing, all other content currently being displayed on the page will also be visible. If you want something to be hidden when printing, use **hidden-print**.

visible summary

Class	Extra small	Small	Medium	Large
visible-xs	Displayed			
visible-sm		Displayed		
visible-md			Displayed	
visible-lg				Displayed

hidden summary

Class	Extra small	Small	Medium	Large
hidden-xs		Displayed	Displayed	Displayed
hidden-sm	Displayed		Displayed	Displayed
hidden-md	Displayed	Displayed		Displayed
hidden-lg	Displayed	Displayed	Displayed	

CREATING HORIZONTAL FORMS

By default, forms in Bootstrap are displayed vertically, meaning that the prompt for an input control, such as a textbox, is placed above the input control. Quite frequently, you want the prompt next to the input control. In order to do this, a little bit of structure is required for the form, and a couple of classes need to be added.

Creating the form element

In order for a form to use horizontal display, the form-horizontal class must be added to the form element.

```
<form class="form-horizontal">
 <!-- form content here -->
</form>
```

Creating the rows for the form content

Horizontal forms use the same grid system we've already covered. However, you won't create normal rows like you did in the past. Instead, you will create form groups. Form groups tell Bootstrap that the controls inside of that section are, as the name implies, a group of controls for the form. In addition, the **form-group** class also creates a row, so you do not need to add in the **row** class.

```
<div class="form-group">
 <!-- label and control -->
</div>
```

Adding the label

Adding labels to forms is a best practice. By adding a label, you've made your form more accessible for screen readers, as well as to touch displays, as clicking on the label will place the form on the associated control. The**label** element supports sizing by just adding the appropriate **col-*-*** class to the element.
In order for the label to be recognized by Bootstrap as a form label, the **control-label** class must also be added.

```
<label for="address" class="control-label col-md-3">Address:</label>
```

Adding the input control

The last item to add is the input control. In order for it to be recognized by Bootstrap, it must be decorated with the **form-control** class; if you forget that class the formatting won't be correct. In addition, the form control itself cannot be sized by using **col-*-***; in order to size the form control, you place it inside of a div tag decorated with the appropriate sizing class.

```
<div class="col-md-6">
 <input type="text" name="address" id="address" class="form-control" />
</div>
```

FULL HTML AND SCREENSHOT

```
<form class="form-horizontal">
 <div class="form-group">
  <label for="address" class="control-label col-md-3">Address:</label>
  <div class="col-md-6">
    <input type="text" name="address" id="address" class="form-control" />
  </div>
```

```
  </div>
</form>
```

CREATING BUTTONS

To create a button interface, you decorate the component with the appropriate classes. The first class tells Bootstrap the control should display as a button, and the second will be the modifier, indicating the color scheme to use.

Adding a button

To add a button, decorate the item in question, such as a hyperlink or button, with the btn class, and the appropriate modifier.

```
<button type="submit" class="btn btn-default col-md-offset-3 col-md-4">Submit form</button>
```

BUTTON GROUPS

Button groups allow you to make radio buttons or checkboxes have the same UI as buttons. The advantage to doing this, besides the fact that it just looks better, it is also more touch friendly than the default radio buttons or checkboxes.

Creating the group

The first step to converting radio buttons or checkboxes to look like buttons is to add the button group. The button group must also have the **data-toggle="buttons"** attribute set.

```
<div class="form-group">
  <div class="col-md-offset-3 btn-group" data-toggle="buttons">
    <!-- radio buttons -->
  </div>
</div>
```

Adding radio button (or checkbox)

The second step is to add in the radio button or checkbox. The radio button or checkbox will be surrounded by a label. The label will provide the button interface.

```
<label class="btn btn-primary">
  <input type="radio" id="first" name="sample" />
  First
</label>
```

DEFAULT BUTTON COLORS

The color of the button changes based on the modifier. All button modifiers have a prefix of btn-

Modifier	Default color
Default	White
Primary	Dark blue
Success	Green
Info	Light blue
Warning	Yellow
Danger	Red

FULL EXAMPLE HTML

```
<form class="form-horizontal">
 <div class="form-group">
  <div class="col-md-offset-3 btn-group" data-toggle="buttons">
   <label class="btn btn-primary">
    <input type="radio" id="first" name="sample" />
    First
   </label>
   <label class="btn btn-primary">
    <input type="radio" id="second" name="sample" />
    Second
```

```
    </label>
  </div>
 </div>
 <div class="form-group">
    <button type="submit" class="btn btn-default col-md-offset-3 col-md-4">Submit form</button>
 </div>
</form>
```

ENABLING INLINE VALIDATION

In order to enable inline validation, a couple of classes and items need to be added to the HTML.

Add validation state to the form-group

To use Bootstrap to set the color for the current validation state, add one of three classes to the form-group container:

- has-success (green)
- has-warning (yellow)
- has-error (red)

```
<div class="form-group has-error">
  <!-- label and input group here -->
</div>
```

Add icon to the textbox

To add a glyphicon to the textbox to indicate status, first update the form-group container to include the **has-feedback** attribute.

```
<div class="form-group has-error has-feedback">
  <!-- label and input group here -->
</div>
```

Then, after the textbox, or input-group, add a span element for the glyphicon. Ensure that the span element is decorated with the **form-control-feedback** class.

```
<span class="glyphicon glyphicon-remove form-control-feedback"></span>
```

FULL HTML AND SCREENSHOT

```
<form class="form-horizontal">
```

```
 <div class="form-group has-error has-feedback">
  <label for="address" class="control-label col-md-3">Address:</label>
  <div class="col-md-6">
    <input type="text" name="address" id="address" class="form-control" />
   <!-- Glyphicon is placed after textbox -->
   <span class="glyphicon glyphicon-remove form-control-feedback"></span>
  </div>
 </div>

<!-- Input group -->
<div class="form-group has-error has-feedback">
  <label for="username" class="control-label col-md-3">Username:</label>
  <div class="col-md-6">
    <div class="input-group">
     <div class="input-group-addon">
       <span class="glyphicon glyphicon-user"></span>
     </div>
     <input type="text" class="form-
control" name="username" id="username" />
    </div>
   <!-- Glyphicon is placed after input group-->
   <span class="glyphicon glyphicon-remove form-control-feedback"></span>
  </div>
 </div>
</form>
```