









PART 1

PART 2

[MUSIC] Okay so here I am in Sublime Text and

I'm looking at the file called grid-before.html which is located

in the examples Lecture26 folder. And the first thing I'm going to

do is I'm going to go ahead and remove this HTML5 shim. It's really only needed here as

it says here for IE 8 support. And we really want to drop by

IE 8 support at this point. It's such a minute number of people

that actually are still using IE 8. We're going to go ahead and

just disregard that. But other than that our file is a regular

HTML starter file for a Bootstrap page. It's got the CSS that we imported. We have some custom styles imported

as well and we have jQuery Bootstrap, JS, and again the script.js which we

might need further down the line. Let's go ahead and examine our HTML. It's very simple, I just got a bunch of

divs, and I already kind of premade it so we'll be able to follow the Bootstrap

structure that it needs to have, in order to have a Bootstrap grid system. And it's got just three column,

column one, column two, column three. Let's go ahead and save that and let's take a look as to what

this looks like in the browser. As you can see right now they're just

behaving just like regular divs, they're stacking one up on the other,

column one, column two, column three. So let's begin to build this out by first

of all following the rule that we spoke about is that, every grid system

has to be inside a container class. And in this case, we're going to go

ahead and go with container fluid. What that should do, is that should

give us some padding on the side. So if we go back to the browser and

refresh, you'll see that we got some padding here

and it's actually 15px to be precise. Next we need to place our

columns inside a row class. So we're going to say class and say row. And once that's done we can go ahead and

now specify a class for each of one of our columns. And in this case we'll go ahead and

specify class equals to and we'll say column and

we'll do nd and let's say -4. And when we do that, that will give us 4,

4 and 4, so three columns across. Let's save that. Let's go ahead and take a look

at what we have in the browser. We'll refresh. And voila, we have three columns across. Now, where did this gray and red border,

gray inside and red border, come from? Well, if we go back and we actually

open up our css and look at the css > styles.css, you'll see that actually

just premade some styles for row. I gave it margin 15px. So between different rows that we're

going to have, we have some spacing. And row div, I basically gave it a red

border and a background color of gray. So that's where that's coming from. So as you hopefully remember from

the first part of the lecture is, md is a sizing identifier that stands for

some break point. Let's go ahead and go to the browser,

and go to getbootstrap.com, and we'll go to CSS. And we'll look at the grid system. And scroll down and take a look at

this table that they have here. And it will tell you what md stands for,

md stands for anything above 992px. So according to this, it means that

anything below this will cause our columns to stack one up on each other. So right now they're behaving

like regular columns. So if I open Chrome developer tools,

as I squeeze the browser in the top right corner we'll see how

wide the browser is in pixels. And I should be able to get

to 992px wide for the screen without anything collapsing or the columns

not stacking one up on the other. But the second I get to 991,

which will be one less, so that means the md style set of

styles will no longer apply and these columns will stack

one up on the other. So let's go ahead and do that. So we are right now at 993,

92 and then we'll go to 91. It becomes stacking one up on the other. So hopefully, that kind of shows

you what that md really means. It means when the columns will start

collapsing one up on the other. So as I mentioned in part one, that's only because there's no other

styles really that are specified here. However if we went ahead and specified a

different style that will be in the range that we spoke about which is below 992px,

991 and below. So for that we'll say small,

we'll make it six. So if we make this six that means that

column one will take half the screen. Column two will take half the screen. And since we're still in the same row,

column three, we'll go ahead and wrap to the next line. And it will still take half the screen. So let's go ahead and save that and

go back to our browser. We'll refresh. And as we start squeezing the browser,

when we get to less than 992, what's going to happen is, is the first

column is going to take half the screen, the second column is going to take half

the screen, and the third one will also take half the screen but

it will wrap itself on the next line. However that's not the end of it. If we go back to the documentation and

take a look, this small that we specified has a break point of 768px. So which means that these

columns will still collapse and stack one up on the other If I

squeeze the browser below 768px. So let's go ahead and try to do that. And we'll go back here, and 768, right around here, 768, 67, and

therefore, it starts collapsing. So, the collapsing will still happen, it will just happen at

a different break point. Okay so let's make the browser

back up to the full screen. So let's go back to our code. And we'll go ahead and copy this

whole row, and we'll paste it again. And if we do that, now we visually separate it,

row number one from row number two. So if we go back to the browser. And refresh, remember I created

some margin below the row. So now you see exactly the same thing and

as we squeeze the browser, both sets of rows will

behave exactly the same way. So you can see the first set of

rows is behaving, one two and three on the second line. And so is the second row behaving one two. And the third column is

on the line right below. Okay, so there's one more variation

that I should show you that you should really have to pay attention to. So, let's go back to the documentation and let's take a look at

this table one more time. You could see that the grid behavior

is a little bit different for extra small,

the col-xs versus the rest of them. The grid behavior says here,

it's going to be horizontal at all times as opposed to the ones

that are small, medium, or large. They'll be collapsed to start and

then will be horizontal above break point. So, basically, they'll be

stacking below these numbers but anything above that, they will actually be

horizontal and try to lay themselves out. Which means that if you want to guarantee

that there is always going to be a particular layout, you could just

specify it in the col-xs and it will stay. No matter what the size of the screen,

that's the layout that will stay. Let's go ahead and go back to

the code editor and test this out. Let's scroll down a little bit here,

and we'll copy this row right here and we'll make another row just like that. And what we'll do is we'll go ahead and

remove, actually, just about everything here. And we'll say, column xs,

extra small, and 6. So now what we're saying is, is that

we want the first column to take up half the screen,

the second to take up half the screen, and the third to wrap around and

take up half the screen. But we want that to always happen. In other words, we don't want

it to collapse at any point or do anything different at any point. So, if we go back to the browser and,

we'll go ahead and refresh this page. You see that even above 1200px, which that is what it is right now, it is

still staying in this particular layout. And if I squeeze the browser more and more, it will continue

staying in this layout. So at a certain point even those columns

will collapse because they're specified as small. So go back to our code editor, as you

can see they're specified as small so the second we go below the small

breakpoint, this too will collapse. However, since this is specified

as xs that will never collapse. It will always stay exactly the same. So to summarize, we spoke about the

structure that the Bootstrap expects for the grid-based layout to work. In other words, the grid needs to be

inside of the container class or container fluid class and also all columns must be

inside an element with the row class. We spoke about the Bootstrap

grid system column classes and the fact that they have a size identifier

that identifies at which breakpoints specified column spans will be ignored and

then all elements will collapse. In other words, the columns will

start stacking one up on the other. And as part of our demo we spoke about

the fact that if no other rules apply, specifying col-xs- will keep that layout

no matter what size of the screen. Next we're going to switch

gears a little bit. And we're going to get into a little bit

more of a business/requirements analysis and requirements gathering type of mode. And I'll try to go over some basic ideas

as to what to expect when you visit with a real client.