# ***RESPONSIVE DESIGN***

[MUSIC]

Play video starting at ::5 and follow transcript0:05

Responsive design was born primarily out of the need to deal

with the explosion of mobile devices that started being able to

browse the web much in the same way that a desktop browser user browsed the web.

Play video starting at ::17 and follow transcript0:17

I found this line graph on the web and

it's attributed to some research done by Morgan Stanley Research.

I don't really know how exact any of these numbers are,

but the general point is well known.

Browsing on the mobile devices exceeds that of regular desktop browser.

And if you look at this graph,

that sounds just about right, that this actually happened around 2013, 2014.

So it's been a little bit since that happened.

So, the point is, you just can't ignore mobile.

You have to design your sites for mobile, as well.

But when it comes to mobile, there's so many devices.

So, which devices do we want to target and which devices do we want to support?

And the answer is really, all of them.

You can't really afford to ignore any of them.

There's such a plethora of these devices out there,

and ignoring some of them would ignore a chunk of your user base.

So how does responsive design help us do that?

Well, I think the answer lies in what responsive design is.

So what is a responsive web site?

Well, that's a site that's designed to adapt its layout to the viewing

environment by using fluid, proportion-based grids,

flexible images, and CSS3 media queries.

And by proportion-based grids, what we mean is or

hinting to Is that the width of these grids or

these columns as we'll see in a moment should be specified using a percentage.

I really like this visualization that somebody put together and

put on Wikipedia when you search for responsive design client.

So this is one of the images on that page.

And this is basically paraphrasing or maybe actually taking verbatim,

a quote from Bruce Lee, a famous martial arts expert.

And basically it says you put water in a a cup, it becomes the cup.

You put water in a bottle, it becomes the bottle.

And you put it in a teapot, it becomes the teapot.

And the idea is pretty simple that content should be like water.

You put it in whatever different devices you put it into and

it should still display, the content should be the same, and

the information conveyed should be the same.

And perhaps the only thing that's different is the shape of that

content, right?

The way it's being displayed.

So using kind of the idea of the previous graphic, we can certainly understand now,

that the site's layout is supposed to adapt to the size of the device.

In practice, most of the time if not all the time,

this means that the site should adjust based on the width of the device.

And I should also point out that content verbosity or

its visual delivery may change.

For example for a restaurant site, a phone number which used to be displayed

somewhere in the site without attracting too much attention to itself for

a desktop version of that site Is now front and

center, and the most prominent item on the page on a mobile site.

And that's, perhaps, because we want people to have their phones.

Once they have their phone, to be able to just tap and call, and put their order in,

or perhaps, we just don't have the flexibility in the mobile device.

To let them go find that phone number somewhere else.

Or just glance at it because of our real estate being so small.

That's number one, number two is some parts of the site may even be

hidden on the mobile version as we now have to measure very

carefully which parts are the most crucial.

Now, are there any alternatives to the responds of design principles?

Well actually they are?

What you could do is you could have a service high technology that detects

your user agent, in other words the type of browser that you using and

figure out whether or not you're on na mobile device or on a desktop device and

then based on our information either serve up a regular desktop version of

the website or serve up to the client the mobile version of that website.

And this is basically what the websites that wanted to have a mobile

version as well.

Used to do before the responsive design kind of came to the forefront.

But the problem is there's a couple of issues with this type of approach.

Number one is, you have high risk of feature diversion between apps,

because now you really technically speaking now,

we now have really two different applications running your website.

One is a mobile version, one is a desktop version.

Number one, number two is,

well mobile devices are just too varied in size among themselves.

So it's hard to make a mobile site that satisfies every client.

So what ends up happening is that you end up sacrificing user experience because

your trying to kind of make one, mobile website that fits them all and

that, basically, never works.

So, nowadays, by and

large, because of these issues, people really stay away from this approach.

Play video starting at :4:37 and follow transcript4:37

The largest part of responsive design is, obviously, the layout.

And the most common layout out there or responsive layout, is a 12-column grid

responsive layout and this is what, a bootstrap, Twitter bootstrap uses and

just about almost really every responsive framework out there uses nowadays.

And, the reason 12 is what's used is because of the factors of 12.

You could basically, t's evenly, nicely divisible by 1, 2, 3, 4, 6, and

obviously 12 itself, which means, we could sub-divide our

page into sections that are evenly split and nicely layout themselves.

Play video starting at :5:14 and follow transcript5:14

So for example we could have content that is split into three columns

out of the 12 so it's three, three, three, three so that all adds up to 12.

Now in case you're confused, I just left some padding here,

I kind of assumed there's some padding just for visualization and

be able to see the different columns easier.

And obviously you can six and six in this layout.

And four, four and four.

And any other combination you want.

You can have eight and four, you could have nine and three and so on.

When I approach these responsive layouts,

Is that we know that the browser window is 100%, right?

That's the width of the browser window.

Which means that we calculate what one column in this 12 column grid layout is.

That's 100% divided by 12.

And that's turns out to be about 8.33% which means

when we specify in our framework, in our bootstrap or

whatever it is we were using, how wide a particular column should be.

We can convert this threes into 25% because

three columns is 25% of 12 columns.

And the 6 columns become 50% of the 12 columns and the 444 becomes 33.33%.

And obviously we're not limited to having just one 12 grid.

12 column grid layout.

We can certainly have nested ones so if you have 4 4, 4, and 4.

Inside each 4 columns we could consider that 100% and

then have another 12-column grid layout inside of it.

So here in the first in the left one we see.

6 and 6.

In the middle one we see a bunch of 3's.

And the most right one we see 4, 4, and 4.

And that's totally acceptable, in fact is done all the time.

Okay, so enough theory for part one of this lecture.

And in part two of this lecture we're going to go ahead and

jump into the code editor.

And see these concepts in action.

[SOUND] Okay, so

here I am in Sublime Text, and I'm looking at the file name responsivebefore.html,

which is located in the examples Lecture 24 folder.

First, we're going to style a paragraph tag, because we're going to use it as our

visual kind of anchor, to see where the elements fall.

And basically, I'm just giving it some background color, some border and

giving a specific height of 150 pixels and I'm giving the width of 90 pixels

kind of to simulate some padding, so it's not going to take up the entire cell and

algrid, it's going to take up 90% of it.

And here actually, I'm using this little trick,

to center our paragraph tag inside a block level element, which is going to be a div.

And the way to center it is to make sure the width is specified, and

then to give it margin right auto, and margin left auto.

Play video starting at ::54 and follow transcript0:54

Okay, so here comes our framework, our responsive framework we're going to code

up, it's going to be pretty simple but pretty effective as well.

So, here first we define our row and

all that is just saying that it's going to be 100% and wherever you put the row is,

it's going to try to take up 100% of that space.

Play video starting at :1:11 and follow transcript1:11

And then we're going to start having some media queries, so

the first one is min width is 1200 pixels.

Which means that if the display, the device display,

is larger than 1200 pixels, all these styles will apply.

And notice that what we're doing is we're defining these columns and

we are marking them with lg for large and one, two, three, four and so on.

And what we're saying is every single column they should float left, so,

basically, we're going to have a float based layout.

Also gave each and one of them a boarder but, that's just so

we could see it on the screen.

And then what I'm doing is pretty simple and something that I've described before.

And what I'm doing is I'm just saying okay so, if it's a 12 column layout that

means each column is 100 % divided by 12 so, that becomes a 0.33%.

Well the two columns is 100 divided by 12 times two,

and that's 16.66% and so on and so on until we get to

column large 12 and that's going to take up 100% of the entire grid and

neither these are the styles we have defined for large screen devices.

Now what about medium screen devices?

What we're going to do, is we're going to have a range and I picked the lower part

of the range to be 950 pixels which is not exactly the standard,

bootstrap has a different number that it uses, but I just pick this for

a demo purposes because I'm going to needed to demonstrate something for

you and you’ll see that in a minute.

And again I'm listing all this classes and

again it going to be left, floating left, and again I'm giving him a boarder.

And this time I'm doing the same thing except, I'm now calling this md for

medium.

And, since these classes don't really exist outside of these media queries,

it means that I can specify for

different devices, different layouts depending on device size.

So for a smaller device, for example, specify layout expressed in md,

in medium devices columns, or

medium devices classes and for a larger device I could express the same thing.

With lg, larger devices, so I could give the same element two different classes and

I know I'm guaranteed that only one of them

will apply at times since my media queries don't overlap.

So both classes will never really be existing at the same time.

And I could define more of these media queries, but for

demonstration purposes, let's stop here and let's take a look at our HTML.

Well again, we have our h1 just to announce to the world

what we're trying to do.

And here we have one single row, and a single row contains eight divs,

it can contain eight items, eight divs.

Each one has a paragraph that says item 1, 2, 3, 4 all the way to 8.

And notice the classes that I'm assigning to each one.

What I'm trying to tell the browser is that when it is a large device,

I want only the classes with lg to apply.

And it will happen automatically since the classes with md don't exist according

to the browser, when the browsers larger in width then 1200 pixels.

And since I know that floating elements just flow to the next line when they can't

fit, I could very easily set up three, three, three, and continue to three.

So that's right there at 12 columns but

I could keep going because it's just going to flow to the next line.

It's three, three, three, three again.

So really I should have basically four of these things, right?

Three, three and three, and three, that's four columns across, however,

when the browser size, or the device size becomes smaller, and

it becomes less than 1200 pixels, or

1199 pixels and lower, I want the layout to switch to these classes.

And these classes are specifying that,

they should really each item, each one of these things should take half the screen.

So six columns plus six columns will take up the entire width of the screen.

And again another couple of six columns will take the entire

width of the screen and so on.

So as I squeeze the browser and make it narrower,

I should see that my four column lay-out, which is expressed in these threes, right?

It's a little confusing, but

it's 3 columns and there's 12 columns all together so you have 4 columns across.

So my four column lay-out, as I squeeze the browser and

make it narrower, should become a two column layout, because I'm now specifying

in a smaller device size six and six instead of three, three, and three.

Now when the browser size gets to less than 950 pixels,

none of these classes will exist as far as the browser will concern.

And what's going to happen then?

Well our divs will no longer be floated because the floating is only defined

inside the media queries which means they will behave just like regular block level

elements and they will automatically stack one on top of the other.

So as the devices get smaller and smaller, I should really have one column

with each one of these items stacked one on top of the other.

The truth be told, I could've specified that media query for

that size of the device, but, in this case, it's just a demo, and

I knew that the items will stack anyway, I just left it alone this way.

Let's go ahead and take a look at what this looks like in the browser.

Okay, so here's our three, three, three and

three layout which gives us four items across and it's just spilling over

to the next line and gives us another four items across again.

And notice if I squeeze the browser,

in fact what we're going to do is we're going to open Chrome Developer Tools.

So as I squeeze in this corner right here,

you'll see the pixel size of the view port.

And as I start squeezing the browser you see it jumps to two.

As it gets closer and closer to 1200, once it gets past 1200 pixels,

it becomes now 6 and 6 column layout, which means

that now we're taking up half the space of the entire width of the browser.

And if we squeeze past 950 let's take a look and

squeeze it past 950 pixels, right here, boom.

The items are now stacking one on top of the other, and

obviously as a pull it back apart, or pull it wider,

it becomes back to our three, three, and three layout, three, three, three and

three lay out, four times, and now we have a four column layout across.

So that's pretty cool, so it seems we're done, and

we could now serve this website to regular mobile phones.

So let's take a look at what this site looks like on an iPhone six.

And I'll go ahead and record the screen of it.

Play video starting at :7:27 and follow transcript7:27

Okay, so here I am on my my phone and

I'm serving up the website using this tool called Browser Sync.

And we're going to talk more about this tool when we get to the development of

a real restaurant website just in a few lectures, but for now, bear with me and

you could barely see here, so let me zoom in here, right here.

Responsive before the HTML, I'll go ahead and click that.

And here we go, here we have our website but something's really strange.

I thought if we go on the mobile site that's a pretty small device, we wouldn't

have two columns, we would have one column because it's such a small device.

It seem to have worked in our browser, why isn't it working on the phone?

Let's go back to the browser and investigate this a little bit.

So let's go ahead and investigate this and

we're going to use Chrome Developer Tools to do that.

And if you look at Chrome Developer Tools here,

you'll see a little mobile phone looking like icon and if you click that It'll give

you basically the layout that we've seen before when we investigated media queries.

And right now, it's set to laptop device, so we're going to go and

switch that to iPhone six.

And when we switch it to iPhone six, it tells us that we should reload.

Let's go ahead and do that, and now that we reloaded, our iPhone simulator device.

It's showing us exactly the same thing as the real iPhone was showing us before.

Let's take a look at one of these elements that was supposed to take up the entire

screen and for some reason is only taking up half.

Let's go ahead and take a look at its box model sizing.

So let's take a look here, this div right here is saying that it's 480 pixels

wide and that's really impossible because the screen itself is only 375 pixels wide.

So what's going on here?

Well, Chrome Developer Tools is actually giving us a hint as to what's going

on here.

Take a look at the right side of the screen,

its showing you the zooming level of the font simulator.

What its telling you is is that its only 40% of its 100% zoom.

It basically just means the fonts zoomed out and

this is really a default feature on all the phones.

They try to zoom out on websites that don't tell them to do anything different.

So, they could try to fit the entire content into the viewport of the phone.

So, how do we tell the phone's browser that, no, this is actually a responsive

website and you don't need to try to zoom out, just stay at the regular zoom level.

Well the way that we do that is by specifying a special meta tag.

Let's go ahead and switch to our code editor again,

and we'll go all the way to the top and we'll place one more meta tag here.

And the name of this meta tag, is called viewport.

And what we need to tell it is, that the content of our viewport, is first of all,

consider its width to be device width, don't try to zoom out and also,

its initial scale to be one and we'll close the meta tag.

So, now with this meta tag in place, the browser will consider the device

width as the width of the viewport and its initial scale factor will be one,

meaning it won't scale anything up or down.

Now let's go ahead and switch back to the browser and

you see that now it switched to being one column.

Just for fun, let's go ahead and try it out on a real phone and see if it worked.

Okay, so here we are again, we're trying to serve up

this responsive before the HTML is go ahead and click on it.

And sure enough, we could see that it worked, it now is only one column, and

we could scroll it, scroll up and down.

And it is indeed one column layout now.

So in summary we went over the idea and need of responsive design.

And we talked about the 12-column grid layout, which is the most common and

most convenient grid layout there is out there right now.

And we spoke about the fact that in order to achieve fluid width we

actually use percentages to specify the width of each column and

different items within the columns as well.

We also spoke about the fact that we needed to tell the devices out there

that they should not try to zoom out and try to fit all the contents they can.

Instead they should behave as they are, mobile devices with

more narrow screens and smaller screens, and the way we would tell those devices

all of this information is by using this meta tag, the viewport meta tag.

Next, we're going to start getting introduced to

the Twitter Bootstrap Framework.