Now so far we've seen that we can customize our widgets by doing a number of things. We can either set

the properties of each of our widgets,

so if our car widget had a color of black, we could simply set its color property to blue.

And because our widgets are immutable the old one gets destroyed and a new one gets created with the

updated properties.

Now we can also change its style or any of the other properties that is predefined by Flutter.

So for the slider widget, we can change its activeColor,

we can change its divisions etc. But there's only a limited number of these properties that we can change

because these rely on preplanning from the Flutter team to create properties for us to change that are

commonly needed.

So if we wanted to dig deeper when we wanted even more control,

then we went into the SliderThemeData where we were able to set the theme for more specific things

such as the disabledActiveTickMarkColor or whatever it may be.

But then we also get to a dead end right?

What if we wanted more customization and we wanted to create a custom widget from scratch? Well in a

lot of programming languages if you come from Swift or Java or you've done some development for iOS

or Android, then generally the way that we go about this is through inheritance.

So we would inherit a component such as the slider, and then we would override some of the properties

or methods of those components.

And this is generally how we do things in both Android and iOS. But in Flutter, the way that it's built

favors composition over inheritance.

So what does that mean?

It means that we want to try and build things from scratch from the simplest widgets possible and that

way Flutter can keep our component's performance and keep our apps fast. So how do we actually go about

composing widgets?

Well even if we think about a basic widget such as a container, it's actually composed of smaller widgets

such as a LimitedBox with a ConstrainedBox with some Align with some Padding with a DecoratedBox. So

a whole bunch of widgets that are very basic and do very simple things came together to build a container.

And this goes back to that idea of having very simple immutable building blocks that are like small

pieces of Lego.

And by combining these simple widgets together, we can build a more complex and more interesting widget.

So rather than taking a pre-built large widget such as a FloatingActionButton and trying to somehow

inherit it and then override certain things about it, it's actually much easier in Flutter to just break

it into smaller pieces to see how it's created by looking at the open source code and then just building

it together yourself using the simplest, the most basic building blocks that Flutter gives you access

to. And that's exactly what we're going to do in this lesson. All right.

So we've already done the icon cards, the slide card.

Now all we have is this final row to create.

So let's go ahead and do that right here.

So we currently have two reusable cards and we're going to have to give them some children.

So let's add a carChild to the first one.

And in this case because we have to stack a number of things on top of each other from the top to the

bottom, it also makes sense to add in a column. And in our column, we're going to already set our main

AxisAlignment to center.

So this way everything gets bundled into the center of the column along the main axis which is the vertical.

And then we're going to give our column some children.

So the first one is going to be a text widget which simply is going to say 'WEIGHT'.

And in this case we're going to use the same style that we've been using so far, which is the klabelTextStyle

constant.

And now we should be able to see our text show up on screen right in the middle of our column.

So the next thing is to add the actual numbers for the weight that the user is going to input.

So again we're going to need a variable that's going to keep track of this.

So we're going to add it just below height and I'm going to keep it as a whole number as well.

And our weight is going to start out being let's say 60 kilos

maybe, that should be all right.

So we've got our weight number as a variable up there.

So we can now add a another text widget just as we have done for our height.

We're going to add it in here as the data.

So we'll put in the weight

and we're going to change it to a string because that's what the text widget wants.

So that's what it's going to get.

Now our weight is also going to be styled.

But this one is going to be styled using the numberTextStyle which is that super large bold looking thing

that looks very similar to this.

So now we have our weight and we have our starting weight.

So all we need to do now is to add some buttons so that the user can actually toggle the weight and

their age.

So these buttons, we've seen from previously when we used our FloatingActionButtons, look very similar

to this right? A round button with an icon child in the middle.

So if I go ahead and add in a FloatingActionButton here,

so I'm just going to use FAB for short so that trusty Android Studio pulls up the thing I need.

And now if I go ahead and hit ENTER and save, then you can see that my FloatingActionButton pops up

and it looks really similar.

All I would need to do is probably give it a child. And the child is probably going to be an icon and

the data for the icon

we could probably just use one of the icons and let's just use add.

So now you can see we have a round button with an icon in the middle, and it styled in this color because

we're kind of using the dark theme but it's pretty easy to change up the color as well by changing the

backgroundColor. So we can change it to a nice sort of light gray maybe color and or do 0xFF so fully

opaque, 4C4F5E.

So this is what we've got so far.

So let's change that icon into a white color.

So we're going to go into the icon widget and change its color to colors.white for convenience sake

and now this is what we have. Now we want two of those

so we'll need a row.

So I'm going to wrap my FloatingActionButton inside a row widget and we're going to add one more.

And I'm going to paste it right below here and maybe we'll add a sizedBox as well so to give it a little

bit of space in between. We'll give it a width of maybe 10.

Let's try that and see what that looks like.

All right.

That looks pretty neat.

All we would need to do is change my row to be centered along the main axis

and now we should have it right in the middle.

All right.

So it looks good.

But the problem is that if we actually read the FloatingActionButton documentations, you can see that

it says pretty clearly that use at most a single floating action button per screen.

And they have a specific use case.

Now it's not just that the documentation recommends against it, but also what if we wanted something

that looked different?

What if we wanted a custom shape for our floating action button or we wanted it to look exactly the

way that we want it to?

So so far we've been using pretty much Flutter components out of the box, and we've either tried to combine

them together in different ways so that we create our own widgets out of a combination of Flutter widgets,

or we've been digging deep into the themes of the widgets that we've been using from material and updating

specific parts of it to make it more like the design that we have in mind.

But in this lesson I want to show you how you can actually dig into the code for each of these widgets

because remember Flutter is completely open source, so all of the components you're looking at the floating

action button or the sliders, you can see all of the source code and you can't do that with something

like say iOS. You can't actually see how an iOS button is created.

The benefit of being able to look at the source code is that if we hold down CONTROL or COMMAND and

click on our floating action button, we get taken to our floating\_action\_button.dart file.

And in this file is all the code that's required to create a floating action button.

So if we search for its main build method, we can see how this thing is built.

So here we go. Here's our build method for our floating action button and it copies over the theme data

from the main app theme and it sets the foreground color or an accent theme and then we go about creating

the actual button. And you can see that the floating action button is created from a basic component

from Flutter called a RawMaterialButton. And this has a number of properties such as what should it do

when it's pressed or its elevation etc. and we can dig even deeper.

So if we hold down CONTROL or COMMAND, we go to the raw material button and we check out its own build method

to see how this thing is built.

Then you can see that it's built using a constrainEDbox which has an InkWell which can detect taps.

Now this is here because the raw material button is a button right? So it needs to detect touches.

But if we didn't need that at all then we could simply just create a material. And if we CONTROL + click

further on material then we go to the main material.dart page and if we have a look at its build

method, then you can see that this is also built from a number of smaller components.

Depending on how deep we want to go down the rabbit hole, we can actually create as custom a design or

a component as we need and we don't have to rely on the default Flutter component at all.

And you can see that when you go into the packages for Flutter, you can see that there's a lot of packages

that people have created when they've built their own custom things such as a custom slider.

So for example if we have a look at this slider, you can see this person has built a two thumb range

slider which is kind of neat. And you can see that they even tell you how they built this widget.

They've basically created it as a pure drawing from scratch in order to display this custom widget.

So what I'm trying to say is that the widgets that Flutter provide are super convenient and in most

cases they're all that you're going to need.

But sometimes if you have a design that you want to implement in particular, then you don't have to be

limited to the shapes or the colors or the styles of the default widgets and you can simply just build

your own.

So let's go about building our own raw material button. And we can create it inside our rows to act as

our buttons instead of having to rely on floating action buttons. So right at the bottom I'm gonna create

a new stateless widget using my shortcut stless. And I'm going to call my widget a RoundIconButton.

So I'm gonna try and be as descriptive as I possibly can.

Now my round icon button is going to be built very similar to how the floating action button is built.

It's also going to depend on a raw material button.

So let's return a RawMaterialButton.

And in this case we have a number of properties that we can tap into including things such as the fill

Color or the highlightColor or the splashColor or the elevation, but you can see that they all come

with default

so we only need to set the ones that we actually want to change. So let's go ahead and set some properties

for our raw material button.

Well firstly it's probably going to need a shape. And you can see that the shape property expects something

that is a ShapeBorder and by default it's given a RoundedRectangleBorder.

Now that might not be what we want,

so let's see what other shapes there are.

So when we look at the documentation it tells us that this is the base class for shape outlines and

it handles how to add multiple borders together and we can define various shapes.

For example we could use circleCorder if we wanted a circle or RoundRectangleBorder for a round rectangle

and there's all of these different styles of borders

we can choose from.

So in our case I'm simply going to choose just a circle border to keep it as a circle and then I'm going

to add a fill color to my raw material button and I'm simply just going to copy over the color that

I have from before.

And now we can simply go over to where we've been using our floating action buttons and we can instead

use our custom round icon button instead.

So let's check out what it looks like.

So it's a little bit smaller than the floating action button and it doesn't seem to have any sort of

elevation.

It doesn't have any shadows around it.

It's just completely flat at the moment.

So let's change it and repaint it to the way that we want it to look.

Let's maybe update its size.

Well how do we know how to do that?

Well we can either read through all of the documentation and see which property it is that we need to

change. Or even easier,

we can take a look at the floating action button and see how the Flutter team actually built this button

using the raw material

button. We can see that it's got these size constraints.

And if we go ahead and try to find this in this file then we can see that it's over here and it's got

some constants, minimum size constraints and normal size constraints.

So if we look for what these constants are, then we can find that it's actually set to have a box constraint

which is tight for a width of 56 and a height of 56.

So why do we go ahead and implement this as a constraint in our material

button?

So it lets add a constraint and put in our box constraints that we've just very conveniently copied

over from the Flutter floating action button.

Let's hit save and let's see now that our shape is exactly the same as the FAB shape. So let's go ahead

and add some elevation to our raw material button and I'm going to add about six pixels of elevation.

But right now if we have a look at our button you can see that it's still completely flat.

There's actually no shadows and you can zoom in to confirm but there's actually no elevation that's

being added unlike this floating action button we've got on the right here.

So why is that?

Well it's because when all material button doesn't have an onPressed parameter, so it's not able to

respond to touches,

it's actually in the disabled state.

So there's even a setting for the disabled elevation.

So if we add that as a six then you can see we now get our elevation. But better than that, we can actually

simply just add a onPressed and define what should happen when somebody presses on our round icon button.

And if I hit save here, you can see that my shadow stays exactly the same as the floating action button.

Now how did I know to change the constraint and also the fact that the onPressed will change whether

if the material button is is disabled or enabled?

Well of course it's the documentation.

So it mentioned here that to define the button size, you change the constraints. And the elevation is

for when the button is enabled but not pressed. So buttons are disabled by default, to enable a ,button set

it's onPressed property to a non-null value.

So there's really no magic here.

Everything comes from reading the docs really carefully. Now,

the only other things that our material button needs is a child so that we can put in an icon in there.

And the icon is going to be different between the left and right.

It's gonna be minus on the left and plus on the right.

So our raw material button luckily has a child property which we can set and we're going to pass this

in into our round icon button.

So we're going to set it as a final property and I'm just going to add it as a widget and it's going

to be called child.

And when we initialize our round icon button, we're going to have a this.child to be have to pass

it in to our child property. And then it's going to go right here and be slotted into our raw material

button.

Now we could leave this as it is and pass in an icon with some icon data but we could also make this

a little bit simpler. Given that we're calling this a round icon button, then in most cases it's probably

going to be around and it's probably going to contain an icon.

So why don't we make our lives easier and just change the child to a icon widget and then only the icon

data needs to be passed in?

So let's change our property from widget into IconData and we'll just call it icon so that we can pass

it in here and we're going to initialize icon instead of an entire child with the icon widget and all

of its family.

Now we can go into our round icon button and specify an icon which I'm going to change to a FontAwesome

icon and I'm going to use their plus sign because it's a little bit bolder and a little bit easier to

see.

So this is what we have here. And we can now also copy this over and get rid of our floating action button.

And I'm going to change this to minus so that I have a minus on the left and a plus on the right.

So it's looking pretty neat.

So now we've built our very own version of the floating action button and it currently looks pretty

much identical to the floating action button. But of course we can customize it to no end because we're

using such a basic fundamental building block that we can actually change a lot of its components.

For example the shape,

so if you didn't want to have just a bog standard circle you could change it to maybe a rounded rectangle

for example.

So we could choose rounded rectangle border and inside we could change its border radius to use a border

radius circular.

And then we could provide maybe a radius in there.

And now if I hit save we get a rounded rectangle as our buttons.

Or you could change it to an ellipse or an oval or even a custom shape. And we could also change the

elevation.

We could make it completely flat if we don't like the way that the elevated ones look with the shadows.

So don't ever feel like you're limited to these default Flutter widgets.

The Flutter world really is your oyster and you can build anything you want onto the screen. So I'm going

to change this back to my circle because I quite like it as a circle. And I'm also going to change the

elevation actually to zero because I actually prefer it completely flat.

It looks a lot better to my eye but again it's up to you.

You can change it any way you wish.

But it's time to actually give our rounded icon button a little bit of functionality.

We need to be of to pass in something to the onPressed so that it performs the action of updating the

weight.

Now we've done this before with the height.

So as a challenge I want you to update the weight card here so that when I click on the plus button,

it will increase that number.

And when I click on the minus button it will decrease the number.

So remember set state is your friend. Go ahead and try and complete the challenge.

OK.

So just as what we've done before with our slider, we need to use set state to update that number here.

So we know that the number is currently stored in a variable called weight.

And we just have to update it whenever the user clicks on one of these buttons.

So inside our round icon button, we need to be able to pass it another property something like onPressed

right?

But at the moment it obviously doesn't have that property yet.

So let's add a final function data type as a onPressed and we're going to add it into our constructor

as well.

So it'll be this.onPressed.

Now it doesn't really matter what you name it. At the moment,

I've kept it the same as the name of the raw material button onPressed and I can simply just pass it

in here.

But the naming is up to you.

And I'm going to also mark both the icon and the onPressed as required using my required annotation

because our widget's called round icon button so it probably needs to be able to be pressed and it probably

needs an icon.

So that's my logic for marking these two as required.

Now that doesn't really change anything other than the fact that when we create a round icon button,

we'll get a warning if we don't have one of those properties added.

So for my onPressed on the left which is the first one in the row, I want this to update my weight

by decreasing it, because this is the minus symbol.

So I'm going to call set state and inside set state I'm going to take my weight property and I'm going

to subtract it by 1.

And the shorthand for that is --.

And in my round icon button that has the plus sign, the onPressed method should increase my weight.

So I'm going to call that state and I'm going to increase the weight by adding ++.

So it adds it by one every time my round icon button is pressed.

So now all we have to do is hit save and check it out.

So now if I hit minus, goes down and plus it goes up.

So perfect.

Now all that's left is to implement the very last card, the age card.

So knowing what you know and using what you've learnt, try and complete this as a challenge.

And I'll wait for you right here with the solution.

All right.

So the age card looks pretty much the same as the weight card.

So we already know how to create the weight card.

And we're going to be reusing this round icon button inside that weight card as well.

So right here is our very last card and we're going to give it a card child.

And this is going to contain a column. Our column is going to have its main axis centered

and it's also going to have some children including a text widget that has the word 'AGE' in it and it's

going to have the style that is the constant for the label text style.

And then after that text widget, we're going to have another text widget which is going to contain the

actual number for the age.

So we're going to need to create a variable up at the top.

And it's also going to be an integer which is going to start out let's say maybe 20, doesn't really matter,

your choice.

Now down here we're going to include that age as the text, but of course we have to convert it from an

integer to a string. And the style for the age is going to be the KNumberTextStyle which is that large

font size.

So let's check on our design make sure that everything looks as we expect it to.

And now all we need is a row that contains our two buttons.

So let's add a row and let's center its content along the main axis,

exactly the same way as we did before. And it's going to have two buttons.

So we'll add two children and it's going to have our round icon button and the round icon button requires

two properties.

So those get added in as soon as we hit enter

which is kind of neat.

And the icon for the left one is going to be FontAwesomeIcons.minus. And once it's pressed what

we want to happen is we want to set state and we want to reduce the age by 1.

Now let's add a sized box so that we can space out the two buttons from each other.

And I'm going to add a width of 10 and then we can simply add another round icon button. And this one

the icon is going to be the plus sign,

so .plus and when it's pressed, we're going to set state and we're going to add to the age,

so age++. Now all we have to do is check to see if we have any errors and we have one here, it's

because I forgot a comma.

Now if I hit save, Dart is reformatting my code and I can now check out my app.

So I've got my weight, my age, my male female height and our entire interface is pretty much complete.

So I can click on a card to choose my gender.

I can move around the slider to choose my height.

So let's put that on my height and then my weight is about, yeah but 63 and my age is a secret.

So now we're ready to go ahead and compute our BMI from all the data that is received in this user interface.

And in order to do that we need to be able to create a second screen and navigate to it.

So this is the first time that we're creating a multi-screen app and we're going to be learning how

to do exactly that in the next lesson.

So I'll see you there.