Now in the last few lessons we've been refactoring our code and making it simpler, easier to read and more succinct. And in the process we've learned a lot of the fundamentals of Dart that enables us to do this such as enums or const, final parsing function and a whole lot of programming knowledge.

But in this lesson we're going to get back to our design and we're going to build out the next card.

Now the next card is going to be a slider and we're gonna be using a Flutter slider but we're going to customize it to our liking. And in the process we're going to learn a lot about customizing Flutter widgets.

So first things first, let's head over to our next reusable card, which is right here. And at the moment it doesn't really do anything. It's completely empty.

So let's give it a cardChild. The cardChild property is going to hold a number of things. Because we have quite a few things that are going to be stacked vertically,it makes sense to create a column widget. And it's going to have a bunch of children, the first of which is going to be a text widget. And our text is simply going to say the word height because this is going to be the height input. And I would really love for that text to have the same style as the male and female text here.

So the same font size, the same color and the same font. And at the moment that style is coming from a constant that's defined in our icon\_content.dart file. And it doesn't really make sense to reach into the icon content file to be able to define a label that's on my input page.

It seems like that would get really messy.

So this is a good time to talk about creating a dedicated constants file. As we create our design and our user interface, it's very frequent that you might want to share designs or share settings across different screens.

So instead of having it live inside a particular file or a particular component, it makes sense to create a new file that is going to be a Dart file and it's going to be called constants.

And this is going to hold all of the things that are constant across our app.

So that is going to include this labelTextStyle for example. So I'm going to cut it out of here and paste it into here instead.

And of course whenever we're using material components we have to import our material library.

Now in addition to these text styles, I can also add in all of my color constants from over here. And we can paste that into this file instead.

And now I have all of my constants together in one place and I can refer to it from any of my files simply by importing the constants.

So at the moment you can see that we have some errors here because it doesn't know about labelTextStyle anymore since we moved it out.

But as soon as I hit import and I import my constants.dart file, then that error goes away and it's able to see into here and see what text style it should apply to that label.

Now similarly on my input page, I've got errors all over the place because it doesn't know about this activeCardcolor and inactiveCardColor.

So let's go ahead and import our constants into this file as well.

So now it's a good time to mention that the naming convention for these constants usually start out with a K.

So we can refactor our code to meet that style. So we can right click on each of our constants and we can go to refactor and rename which is our friend that is going to search for all the places where we use this constant and rename it all in one go.

So instead of calling it bottomContainerHeight, the convention is to start with a K, K for constant because C is used for a lot of other things.

So K for constant and then we're going to start off with a capital B.

So K constant, bottomContainerHeight we're going to refactor that and now you can see everywhere where we use that, it's now been refactored automatically.

So I'm just gonna go ahead and do the same for all my other constant. And from now on when I'm creating new constants in my constants.dart, I'll be using that naming convention.

All right.

So here are all my constants renamed and they're going to be renamed everywhere where they're used.

Now why did I go through all that trouble of renaming all my constants?

Well whenever you're trying to pick out a color or a text style from your constants file, you start out by typing K. And now Android Studio is intelligent enough to show you all of the relevant constants that you might want to use at this point. So you can see that there's kBottomContainerColor, kBottom ContainerHeight and there's also constants that Flutter has defined such as the kMaterialEdges, kMaterialListPadding,so this way all the constants can be searched easily in one go.

So let's change that back to kActiveCardColor. And every time you're looking for a constant you get that dropdown list to select from, which is much easier than trying to remember what the name of your constants at the top of the file is especially once this list gets a bit longer and your app gets more complex.

All right.

So back to our reusableCard which now has a text widget and I'm now able to give it a additional style which is going to be the kLabelTextStyle,so I can simply select that.

And now you should be able to see in the app we've got the same styling for that height text. But our card has now resized to the size of its child.

So remember that the way that containers work is that if they have no children, they'll try to size themselves to be as large as the parent allows,and in this case it was stretched to the full width of the column. But once it receives the child, it will try to resize itself to fit the size of the child.

So that's why my card has now shrunken in width.

But this is relatively easy to fix.

We can simply go to our main column at the very top and we can change it's crossAxisAlignment to our friend which is crossAxisAlignment.stretch. And now it will stretch all of its children to take up the full width of the column no matter the size of the child. In addition to the height label, I also want to add a value for the selected height so that when the user toggles the slider it will change as they move it around.

So I now need to add a number and a bit of text that says cm or centimeter. In order to do that, I'm going to add a row under my text widget right here. So my row widget is going to have just two children and both of them are going to be text widgets because one of them is going to display the height that's selected and the second one is just going to display the word cm. My text widget is going to contain some numbers.

So let's say it starts out with maybe the string 180. And then it's going to have some styling. And I want this to be super large so my textile is going to have a font size of maybe 50 and it's also going to have a super super heavy font weight. So I'm going to set it as probably w900 which is the heaviest weight possible.

The heavyweight champion of my font.

So now if we take a look at our design, you can see that the number is pretty much as thick as my icons and I quite like that consistency. So I'm probably going to be using this number style across the other numbers that are on the screen,so for example when I create the weight card or the age card. So it makes sense to turn this into a constant.

Pause the video and create a constant from this text style and replace all of this code here.

All right.

So of course we're going to create our constants in our brand new constants.dart file.

We're going to start out with the keyword const and we're going to name it using the K convention.

So this is going to be k and I'm going to call it numberTextStyle.

You can call yours anything you wish.

And I'm going to replace it with what I have here which is my text style.

And over here I'm going to call it NumberTextStyle.

And now that should refer to our file over here.

And I'm going to paste in that text style I had from before.

So now I can reuse this one and this one across a lot of my design components.

And it keeps my code relatively simple and straightforward on my main page. Now that I've created my number,let's go ahead and add the second text widget in my row which is going to simply just say cm.

Now if you're from the States and you want to have imperial as your unit then feel free to change the numbers or the units as you wish.

Now, that cm is going to have the same text style as my male, female or height labels.

Simple enough, I can simply tap into the style property, add my k for constant and select my label text style.

So now I have 180 and I have my cm.

Now you'll notice that everything is kind of all over the place at the moment. My height label is all squished right at the top and my number row is all the way to the left.

And it doesn't look very good.

So let's bring all of our components into the center of our column by changing the mainAxisAlignment of our column to center.

And this should bring everything down into the middle of the column.

Now let's also make our row centered along the main axis so it brings it right into the middle, by going over to our row and changing our mainAxisAlignment to center as well.

And now everything's at least in the middle.

But there's one other thing that I would really like to have, which is for the cm or the centimeter or whatever units you chose, to be on the same baseline as my number here.

So that way it looks like it's actually a 180 cm rather than sort of having the unit floating around somewhere in the middle.

And you can see in the final design that's exactly what I have and it looks a lot nicer when it's aligned all to the same baseline.

Now we can do this quite easily by simply adding a crossAxisAlignment to our row.

So remember that the main axis is along the length of the row and the cross axis is along the short side,so the width of the row.

And we're going to set our crossAxisAlignment to one of the options which sounds very very much like what we want, which is .baseline.

And in this case, it's going to place the children along the cross axis such that their base lines match.

Let's go ahead and select that.

Now at this point if we hit save and we tried to look at our design, you'll see that it will break.

And the reason it tells us seems kind of cryptic. It's saying if we are going to change our cross axis alignment to baseline, then we actually have to have a text baseline property set to something and it can't be null.

How do I know this?

Well you can simply take this error message from our run panel and pass it through our friend which has Google of course. And we can search to see if this has already occurred for somebody else.

So very often when you get error codes with Flutter, you can search for it in Google and it'll come up in Stack Overflow or in the actual Flutter GitHub repository issue area. And here somebody actually from the Flutter team has provided the solution here saying that you have to specify the textBaseline argument if you want the baseline to be aligned, so that they know which baseline to line against and maybe in the future, they're going to plan on making that message better.

And this is really one of the great things about the Flutter community. It's in development but your input and also all the other developers experiences are contributing to the final product.

And day by day fixes will be made and changes will happen and it's just going to get better.

So let's try and implement that.

So they're talking about something called a textBaseline argument.

So where we have our crossAxisAlignment for our row, if we go ahead we can see that if we click on CONTROL + J or CONTROL + Q on Windows, you can see that one of its properties is exactly that text base line.

So let's try and set it so that our crossAxisAlignment.baseline will work.

So let's add textBaseline and then we can start typing TextBaseline with a capital T. And we can now see that we have two options either alphabetic which is for alphabetic characters or ideographic which is for basically graphical characters.

So we want alphabetic and now if we hit save, you can see that our error goes away and our row is now beautifully aligned across the baseline. Pretty neat.

Now if you're working on any sort of development be it web or mobile or Flutter, you'll inevitably encounter errors and warnings and problems. But just know that there's a strong community behind you who will be able to help you out and you are not alone.

And in most cases the things that you've experienced, other people have experienced too and they've probably come across a solution already. All right.

So we've solved our labels and our numbers, and it's now time to add a slider. Now Flutter has a widget called a slider and it will implement a material design slider. And if we search for it, you can see that this is roughly what it looks like. It's got a small round thumb which you can move around and it's got one active side and one inactive side.

And when you click on it, it's got this sort of outline that pops up. And that looks very similar to what we want to implement in our design.

So let's go ahead and implement it so that we can create this design. Now right below where I have my row ending here,

I'm going to add a slider widget. And the slider widget has a number of properties such as a onChanged callback.

So it's expecting a method that takes a double as an input and outputs void.

So whenever you see an arrow as one of the properties, you always know that this is expecting a function.

And it can also detect when a change starts or when the change ends.

So that means when the user starts dragging and ends dragging, you can also set the minimum position and the maximum position.

So maybe if you're setting a temperature, you'll go from 0 to 30, but maybe for other users your scale will be different so you can determine that.

And then there's the number of divisions you might want, a label for the slider and then there's the active color and the inactive color.

So the active color, the documentation says, is the color to use for the proportion of the slider track that's active.

And if you click on it, it gives you a little bit more detail.

And it shows that the active side is actually the side between the thumb,so the circle in the middle, and the minimum value.

So basically that's going to be the color on the left here.

And then the inactive color is the one on the right.

So let's go ahead and implement some of these properties so that we can bring our slider to life. So first things first, we're going to provide a value. Now,the value of our slider is going to be the height that the user selected.

And we also want it to show up inside here, in this number.

So let's go ahead and create another property for our class that's going to be a integer and it's going to be called height. And we'll start it off with a value of 180.

Now this can't be final or constant because it's going to change all the time and every time the user slides on that slider. Instead of hardcoding 180 in here, we're going to be putting the value of the height in here.

But because the text widget needs a string as the data, we can't simply just put a number in here. And it tells us that type it can't be assigned to type string.

So let's go ahead and convert it by writing height.toString.

So now we have a string that's converted from our number and you can always convert numbers to strings or ints to doubles. But if you want to do the opposite way then it's a little bit more tricky because A B and C don't map directly to a number right?

But this is easy to do.

So now for our slider, the value is going to be our height. And we're going to get another incompatible issue here.

It's going to tell us that the argument type int can't be assigned to the parameter type double, and that's because the value is expecting a double to go in there as the value.

So what we can do is we can use again our trusty .notation to change it to a double.

So again super easy conversion and easily doable.

So now our height is going to be displayed as the value of the slider.

So that's the starting value basically. Now,next we're going to define the scale for our height and most people are not zero centimetres tall hopefully.

I guess unless you're like a single celled organism or something, but they're probably not going to be using our app.

So let's start off with a minimum value of maybe let's say 120, and maybe a max value of say, I don't know, 220 right?

So because we're in centimeters, we're talking about 1 meter 20 and 2 meter 20. You can obviously change your scale if this doesn't make sense at all.

If all the people that you live around are like three meters tall, then feel free to change the scale.

Again these are things that work really well as constants.

So it might make sense to add it into our constants over here.

But for now I'm just going to leave it in here to make it easy to see how our slider is being built out.

But feel free to create a constant and you'll be able to easily hit up one file and update those constants if needed.

So you can see that our slider is still being highlighted as yellow,and whenever that happens you also get a line in the right telling you what the warning is about.

You also get a little yellow box telling you there's problems or there's warnings found.

So in this case if we hover over it, it tells us that the parameter or the property onChanged is required.

So at the moment, we're still missing one more property which is the onChanged.

And you can see that just as I was typing over here, that it's telling us that it expect a function something that takes a double as an input and returns void.

So we have to pass in a function into the slider just as we did for our reusableCard. In our onChanged, instead of just having an anonymous function with no inputs, it actually expects a double value in there.

And that value is going to be the new value of the slider.

So you can see that in the docs it tells us that the onChanged is called during a drag when the user is selecting a new value by dragging.

And if we click on it to see more information, you can see that the slider will pass the new value to the callback and we will be able to tap into that new value inside the anonymous function.

So this is how they're recommending us to use the onChanged property.

So let's go ahead and do that. In our onChanged property, it's going to take a double.

And we're going to call it newValue.

And this is going to be the value that will be passed to us from the slider when the slider gets changed by the user.

And then inside the curly braces here, we'll be able to tap into that newValue and we'll be able to use it for our purposes of changing the height.

Now if we go ahead and hit save and when we run it, we can see it vaguely on screen but it's so dark that we actually can't differentiate it from the background.

So let's go ahead and add some color to it.

Let's add an activeColor which is going to be a custom color as usual.

And in this case it's going to be 0xFFEB1555, and this will turn the left side pink and let's

go ahead and add a color for the right side as well.

So the inactiveColor and this is going to be 0xFF8D8E98.

So now we have a gray on the right and a pink on the left.

Now at this stage something very curious happens. When I click onto the handle of my slider and I drag it around, because we're going to print the new value that the slider is at, we're going to see it inside our console.

So you'll see the value change and I can go all the way down to the very bottom which is 120 and go back up to 220.

But the slider itself is not really doing anything on screen right? It's not animating, it's not moving at all. It looks like it's completely stuck. If it wasn't for all of these readings that I'm getting by printing the new value, I would have thought that my slider is completely not working.

Why is this?

Well if we head into the documentation and we look at the onChanged property, you can see that it says that the slider passes the new value to the callback.

So when I move my toggle around, it will pass the new value into my callback through this input here.

So I get access to that new value which is being printed inside my console. But it does not actually change the state of my slider until the parent widget rebuilds the slider with the new value.

So how do we rebuild the parents slider with the new value?

Well the callback,so everything in between these curly braces, should update the status of the parent using set state.

So the parent gets rebuilt in the way that they've shown here.

So for example here in their onChanged, they have a new value being passed into this callback here and then that new value is used to update the value of the slider which is right here.

So this is how we're able to update the appearance of the slider.

Let's give it a go in our code as well and actually get our slider moving and animating. Instead of printing the new value,we're going to set the height property which is a mutable or a variable property that we have in here.

And it's also an integer which starts out at 180.

Once the slider is changed, we're going to set the height to equal the new value.

But the problem is that new value is a double but height is an integer.

So let's go ahead and round it up to an integer so that can be turned into the nearest whole number.

And then all we have to do is call set state and make sure that the state of our slider gets updated based on this height.

Now if we hit save and we check out our handiwork, then you can see our slider's moving and our text which displays the height also, is also updating.

Perfect.

We've got all of our functionality of our slider card now enabled.

