All right.

So once you've got the skeleton project checked out from GitHub and you've done all the usual steps of getting the packages and opening it in Android Studio, you should see something that looks like this.

And we currently only have a single file main.dart, and there's only a little bit of starting code here,this time. We have our BMI calculator as a stateless widget which will return a Material app which has the input page as the homepage. And the input page is a stateful widget,so things in it can change and can update. And inside the state of the input page,we have a scaffold with an app bar that just says BMI calculator, some text in the center which just says 'body text' and also just a dummy floatingActionButton.

And the reason why these things are here is so that I can demonstrate something to you. And the thing I want to talk about in this first lesson is Flutter themes.

It's simply a way for us to be able to define our custom color palette and our custom styling for the look and feel of our app.

So this means that if you're creating app for a brand, or your company or for your startup, you can keep it consistent across the different screens and be able to define your own custom styling and your own custom colors and keep it consistent across different pages.

Now if you're interested in first reading a little bit of background about Flutter themes, then I'll link to this page in the course resources.

And this comes from a section on the Flutter docs called cookbook. And cookbooks are basically what they sound like. They're essentially a book full of recipes that tell you how to achieve a certain thing.

So for example, if you want to play sound, you first have to do this, then step 2 you have to do that.

And then finally at the very end once you've completed everything written all the code that is necessary,then you should be able to achieve the end goal which is similar to our potato chip recipe. Now the Flutter team have created a whole bunch of recipes in their cookbook, and all you have to do is head over to flutter.dev/docs/cookbook and you can see a whole bunch of things that you can do.

And there's constantly new additions and updates in here.

So it's a really really cool place to check out when you want to do something specific with your custom app.

So let's head back to themes and what we're gonna do is we're gonna create an app theme.

We're going to be able to share some styling and color across our entire app.

So to do this, we have to provide a ThemeData widget to our Material app.

So when we construct our Material app, in addition to be able to have a title or home, we can also specify a theme.

And this is a ThemeData widget, which has a whole bunch of properties.

And if you check it out, you can see just what those properties are.

And as I said, there's loads of them for you to go through. Things such as the app bar theme or the bottom app bar color or the brightness button colors, whole bunch of things that you can set so that you will be able to have some default colors whenever you create a new button or whenever you create an app bar.

And this is shared across all the pages in your app.

Now this is super relevant because in the app that we're making in this module actually has more than one page for the first time.

It's got multiple screens.

So when we do the hard work in our theme, we will be able to use that across our entire app. If you go ahead and run the skeleton project as it is as soon as you've checked it out from GitHub,you'll be able to see that there is no styling that we've implemented as of yet.

So everything that you see on screen including the app bar and the text and also the floatingAction Button, they are styled using the default Flutter theme. And that means the app bar has the material blue color and the text for the app bar is white, but the text in the body of the scaffold is black.

And then our floatingActionButton is also the same blue color. And anything that is inside the floating action button, any of its children, is usually white.

This is the default Flutter light theme.

And if you want to use this theme, then you don't have to do anything at all.

You don't have to specify anything for the theme property of the material app. But if you wanted to customize the theme for your app, then you can simply add a new property called theme. And as you can see this expects a widget that is a theme data widget. And theme data will be able to define visual properties such as colors, fonts and shapes for everything that exists inside this app.

So let's add a theme and if you start typing theme, you can see that there's a couple of default ones that come prepackaged. The light one you won't usually ever have to select because that's the default value anyways for all of your material apps. But there's also one that's dark.

So if we hit ENTER, and we add the dark theme to our material app, and if I hit save we can see what that looks like.

So the background becomes dark, the app bar gets very dark and the floating action button has this nice accent color of sort of turquoise green color.

If you wanted this particular theme, then it's as easy as changing the theme to the dark theme.

But what if you wanted it to be more custom?

What if you wanted to be able to define what the background color should be, what the app colors should be what the accent tones are?

Well then you would have to create your very own theme data widget.

So that's what we're going to do right now. As I mentioned before, the theme data widget has loads of properties that you can change.

And it's a good idea to actually have a read of this document just to be able to see what are all the things that you can change and what each of these properties actually do.

So the accent color for example would change the foreground color for widgets such as knobs, text, over-scroll edge effects and also of course our floating action button.

And then there's others such as the button color, which is the default full color of the material used in raised buttons.

So there's detailed descriptions of what each of these properties will do when you change them.

So have a read of that before you get started and once you familiar with what are all the different knobs that you can turn, then it's much easier to know what you can change and how you can customize your app. Now in our case, the property that I'm going to change and this is probably the most frequently changed property, is going to be the primary color.

Now the primary color is the color that will be given to the background for major parts of the app such as the toolbars or the tab bars.

But notice this doesn't include the body of the scaffold for example.

If we go ahead and change our primary color to a, I don't know, something different from the default say a red color, then let's hit save.

And you can see that firstly our theme defaults to the Flutter light theme, and the only change that we've made is actually changing the top up bar to have a red background color as the primary color.

Now if we had other things in our app such as a tab bar or a bottom app bar, then those things will also change to have the same primary color.

Now what if we wanted to change the accent tone here?

Well we can of course just go in here and change the accent color.

Let's change that to a purple for example.

See what that looks like.

So now all floating action button has changed its fill color to the accent color that we've chosen here.

And you can mess around with this and all of the properties that you see here until you achieve the desired look and feel for your brand or your company or your app. And the degree of customization that you can do is really very extensive.

So far we've always been using the default material colors which come from the material palette. And to use it,we've always tapped into colors and then we've simply selected a particular shade or a particular tone of colors in the palette. But what if you wanted to use your very own custom colors because there's a lot more colors than what's in the material palette?

Well in that case you might want to start off by getting a hex code.

So for example, the design that we're replicating here has this nice sort of dark blackish purply color as the background. It's not exactly black and it's not exactly purple, but it's a nice shade in between.

Now one of the cool tools that I use very frequently is a free tool called ColorZilla,

and it works on both Chrome and Firefox. And what it does is it adds a little color dropper into your browser.

So that means when you click on it, you can pick color from the page.

So for example if I wanted to get this shade of yellow from this page and I want to use it in my app or on my website, then I can simply click on it and it will copy the hex code of that color to my clipboard where I can then implement it.

So if you want to install this, it's as simple as just clicking on this and installing it to your Chrome or to your Firefox.

And it's pretty easy to use.

Let's put that into action.

Let's use our color dropper and we're going to pick the color from the page. And I'm going to try and pick the color that's the background of this app.

And that comes out a 0A something something. And I'm going to paste it into here and change my primary color.

Now in order to use something that is not a default material color palette color which we get from here, we have to use something else.

So if we delete that and we start writing color, then you can see there's a whole bunch of different ways that we can create a color to use as our primary color.

And the most common one that you'll be using if you're using hex codes, is the first one where we simply write the word color and inside the parentheses, we add the value.

And I'm simply going to paste that hex code that I copied over from earlier on.

Now if you have a look at the color class in the Flutter docs, you can see that it's expecting a 32 bit color value in the ARGB format. Well what does that mean?

So A is the alpha or the degree of transparency of the color.

So we know that if we were to select a color, say this green color, then we could go from fully opaque to fully transparent to have as much or as little of the color show up.

That's also something we can specify when we're using the color class. In most cases, you're going to be using a fully opaque color.

And the value for fully opaque is 0xFF, and that is the value that you're going to be seeing in a lot of our colors.

So the trick is whenever you copy over a hex code, it only includes the code for the R, so the red, and then the G which is the green and then the last two values stand for the B which is the blue.

So in all three of these channels, red, green and blue, If we specify a different amount for each of those colors we end up with a specific color.

So for example if we had a lot of red but not so much green and a lot of blue, we get a purple sort of color. And usually colors are represented using the hash notation, and we have our hex code here. Now if you've never worked with hexadecimal color codes before, then I'm going to include a link to this Stack

Overflow page which does a really good job of explaining how you can use hexadecimal to represent different values.

And it's a good place to start to understand where all these letters and numbers come from. But essentially,all that it does is it encodes a certain amount of red, green and blue, and mixing those together creates a specific color. Whenever we copy over a hex code,it usually looks like this.

It has six values and it's preceded by a hash tag or a pound sign.

Now in order to use it in Dart, the easiest way is to simply paste in the hex code,delete the pund sign and add the 0xFF and this will make our color be a opaque color and the remaining six characters we're gonna leave as it is, because they encode the actual hues of the color.

Now if we go ahead and hit save, let's check out what our app looks like.

So you can see the app bar which is determined by the primary color, has now changed to the exact same color as what we saw in the dribbled design.

So what if we wanted to change the background color of the body of our scaffold as well?

Well here's a challenge.

Try and see if you can work it out using the documentation on the theme data class looking through the properties that we can change and trying to figure out which one might affect the background here.

One of the easiest ways to troll through documentation is sometimes just to use COMMAND + F or CONTROL +

F to find relevant keywords.

So in our case we want to change the background of our scaffold body.

And if I start searching for background, it highlights all the places where it appears and you can see that there's the background color property, and then there's the dialog background color but there's also the scaffold background color.

And this as it explains, is the background color for a typical Material app.

And this is exactly what we want to change.

Let's add that property in here.

So in addition to the primary color, we're going to change the scaffold background color and it's gonna be exactly the same as the previous value.

So let's hit save and now, voila! We have a scaffold background changed to that same purplish blackish color. But now we have a bit of a problem because our text in the middle, which used to be in the middle of the page, has now completely disappeared.

And the reason is because the text has a default color of black, and when black is next to black,it's kind of hard to see.

So we need to change it to a white color. In order to do that,we can't simply just change the text color in our theme data because, there's a lot of different types of text that could change say the body text or the heading text.

So the property that we have access to though is something called text theme. And this takes a specific widget which is called a text theme widget, and we can specify in here the part of the text that we want to change.

Is it the caption? Is it the button?

Is it the headline?

Well in our case it's actually body1.

So it's the default text style for our Material app body.

So let's pick the body1 property and this is going to expect a text style. Similar to how we've been changing the style of our text inside our app,we can also do the same inside our theme. So we're gonna change the color to a white color.

I'm just gonna use the material colors palette here because it's much easier, and then I'm going to hit save.

And we should now be able to see are text show up because it's now white and it has contrast.

Now the other option if you didn't want to use the colors palette, you can also use what we've been doing before which is color and it's a good thing to remember that in order to have a white color, all you have to do is just add eight Fs. So one two three four five six seven eight.

And if you hit save, it's exactly the same thing as using the colors.white.

So it's your choice whichever you find it easier to read.

Now previously, we saw that the dark theme pretty much has everything that we would want in our theme because we're kind of also going for a dark background and a dark app.

So how can we use everything that's already specified in the dark theme but then update certain parts to our liking, such as the specific background color or the specific app bar color?

Well there's a really neat trick that you can do which is called copyWith.

Let me show you what we can do.

I'm gonna go ahead and cut out everything that's in our theme data widget. And as the theme, I'm going to go back to that dark theme.

And after we specify the dark theme, I'm going to write a dot and I'm going to choose the copyWith method.

And inside here, I can specify all the things I actually want to change from the dark theme.

So that's going to include our primary color, our scaffold background color.

I'm going to delete the accent color because I actually quite like the one that you get from the dark theme.

And I'm also going to delete everything in our text theme because we know that the dark theme already changes all of the text to white.

So this vastly simplifies the code that's in our theme data.

And if we hit save, you can see that the result looks pretty much the same other than the fact that we now have the default dark theme accent tone. Now even though we can create our theme that is app wide at the Material app level by changing the theme property, we can also actually have more granular control over the theme of particular widgets.

For example, if you wanted to change the floating action button's theme on this particular page,then we can actually specify a custom theme just for this floating action button. So we can say this floating action button can be embedded.

So we're gonna use our intention actions, and we're going to wrap our floating action button inside a new widget and that widget is going to be a theme widget which always has a property of data. And this expects some theme data.

And here, we can change our theme. So maybe I want to change it to the light mode where our action button is going to become blue. Or maybe I want some custom theming, where I want to change my accent color to a custom color, maybe we can go back to that purple color again. So you can see that in this case even though the theme that is app wide is a dark theme with a custom primary color and a custom background color, we can actually change any widget's theme just by wrapping it inside a theme widget.

And this gives us more granular control on a particular page or for customizing a particular widget.

But we actually don't need a floating action button in our app.

So I'm simply going to delete everything that is in our floating action button property including our theme widget. And now we're ready to move on to the next step which is actually creating our design for our calculator.

But right before I do that, I want to go ahead and separate out my main and my input page.

Previously we've always pretty much created everything on one page and it's sort of a mumbo jumbo.

But now, I want to be able to show you how we can create multiple routes, multiple screens and we also want to keep our code a bit more tidy. So as our code grows, it's going to take up a lot of space in the page.

So it's usually a good idea to actually separate out individual components into separate Dart files.

The way that we would do that is we would right click on the lib folder and we will create a new Dart file. And I'm gonna call this input\_page. And then we're going to click OK, and that creates our input\_page.dart. And if it asked you whether if you want to add it to Git or not, you can click yes or no.

In this case it doesn't actually matter.

So here's our input\_page.dart and I'm going to cut my stateful widget and the input page state class out of our main.dart, and I'm going to paste it in here.

So the first thing you see is a whole bunch of errors and this might be a little bit familiar because it's very similar to when we open up our brand new skeleton project, when it doesn't know what each of these things are. In order to solve that,all we have to do is import our material.dart package. So that it knows what a stateful widget is, what a scaffold is and all of that.

But now we don't know what the input page is.

Well how can we reach this particular page? Again,the secret is in the imports.

So we're going to type a new line of import and we're going to import the input\_page.dart into this page so that we can access the input page state for widget. A and we can create it as the home of our Material app.

So very often in the wild when you come across Flutter apps, you'll see it's structured like this.

The first main.dart file usually contains a lot of theming.

So what are the custom fonts?

What are the custom colors?

What is the main app going to be named?

But then it's going to direct towards the first screen and then for every subsequent screen, there should be a separate Dart file. And we're going to see that as we create more screens in this current module.

But now we've separated our app into two distinct pages and we're already done with our theme for our app.

So we can leave this page alone and we can focus on a much shorter and more concise input\_page .dart file. That's where we're going to continue in the next lesson.

So for all of that and more, I'll see you on the next lesson.