So let's open up screen 1 and you can see all that has is an app bar that's red, has the name screen 1 and a button which says go forwards to screen 2. So Screen 2 is pretty much the same story but it has a blue app bar and has a button that says go back to Screen 1.

So if we go ahead and run our app, you can see as promised Screen 1 is pretty simple. It's just got a app bar and a button.

Now at the moment when I click on this button to go to screen 2, nothing happens. And this is as expected because in the onPressed it's completely empty in that callback.

So let's see how we can create a route to Screen 2 so that we can see it when we press on that button.

Now Flutter has done a really good job of creating these cookbooks that we've been seeing along the course.

And one of these tells you exactly how to create a navigation towards a new screen and how to go back.

So we've already created our two routes, the screen 1 and screen 2.

Now all we need to do is use something called navigator.push to add a route to the stack of routes.

And what pushed us is very similar to when you push pancakes onto a stack right?

You would just add them one by one on top of each other.

It's a stack that goes on top and when you decide that you want to eat a pancake or you want to pop off a pancake, then it's the top one that comes off.

So this is very similar to how our screens are organized in our app.

They each go on top of each other and it's only when the top ones come off that we actually see the screen that's behind it.

So let's try and do this pancake maneuver in code. Now as it says, we have to use the navigator.push method and we have to pass in two things, a context and also build a material page route and tell it where we want to go. Now the context refers to the build context.

We saw this a little bit earlier on when we tried to copy the current slider theme. And to do that we said give me the slider theme of the current context.

So that's the current build context and every widget has one.

It's simply a way to figure out where in the widget tree this particular widget lives. And by passing in a context, it helps Flutter figure out where we are and where we need to go.

So let's try and go ahead and push screen 2 onto screen 1.

So inside the file called screen 1 inside the onPressed, we're going to call navigator.

And remember every single stateless widget has one.

As long as your scaffold is inside its own widget then it should be able to call navigator.push and the context is going to be the current build context of our screen 1 widget, which is the current location of our widget in the overall widget tree.

So this is a small widget tree which is going to be embedded in a large widget tree when the app runs and the location of it is determined by this context.

So that's the first thing it needs.

The second thing it needs is a route. And we're going to be using that material page route to build the second route.

So let's pass in the MaterialPageRoute and the builder is going to take a function.

So it's going to be a function that has a input called context and then it has a body which is going to return whatever it is that we want to navigate to.

So in this case, it's a new screen 2 object. And let's cap that off with a semicolon as well and add all of our commas so that Dart will be have to reformat our code.

Now at the moment it doesn't know what screen 2 is even though we can see clearly that we have a screen 2. And this is because we haven't yet imported the screen2.dart file.

So now if we hit save and we check out our app, when I click on this button it takes me to screen 2.

And it's even clever enough to know that for example on iOS when new screens come on top it comes usually from the right and it gets pushed on top in this kind of animation.

So by using this material page route, it actually helps us automatically determine what kind of animation we need.

So if you go into the Flutter inspector and you click on this button right here which toggles the platform.

So now I have a Android looking screen even though I'm running it on an iOS simulator. And you know this because our title is right at the left corner of our app bar.

And when I click on the button to go to screen 2, it actually pops it up from the bottom which is the difference between iOS and Android.

And that's all prebaked-in by the Flutter team and all you have to do is use this material page route builder.

So let's toggle our platform back to iOS or keep it on Android if you want.

And now we're going to try and figure out how can we go back to screen 1 by pressing this button? Just as I said before, we can push and we can also pop. So pop is much easier because all it needs to do is to destroy the top screen so that we go back to the one that's below. Inside our screen2.dart,when the go back to screen 1 button is pressed, all we need to do is write Navigator.pop and it'll pop the current screen and destroy it.

So now let's hit save and check it out. Go back to screen 1.

That's exactly what it does.

And we can now go between 1 and 2 as much as we wish. So that's pretty cool.

But what if we had a more complex app with lots of routes?

Say for example we had a screen 0 which had two buttons, one to go to screen 1 and the second button should take us to screen 2.

Well we can achieve that quite easily using named routes. Again in the cookbook,it describes how we can do this.

So let's go to the one which is navigate with named routes.

And we've already created our two screens and now we have to define the routes.

And we're going to do that where we create our material app.

So in our case that's going to be in our main.dart file.

So instead of having our home or our starting point as screen 1, let's change it to screen 0 so I can show you what it looks like.

Now of course we have to import screen0.dart in order to be able to use it.

So that's right screen0.dart and let's hit save.

And now because we want to go right back to the beginning, we're going to click on a hot restart.

So we go back into our main.dart and we go on to screen 0.

So screen 0 has two buttons, go to screen 1 or go to screen 2.

And when I click on them, it should take me to the correct route.

How do I implement this?

Well our material app also has a property called routes and this allows us to define what's called a map.

So a map is similar to a dictionary in other programming languages,and it's basically a collection type. Just as lists are collections of items,maps are also collections.

But the difference between maps and lists is that maps have a key and a value. Just as your dictionary has a term and the definition, maps also have the same thing.

So we can define our route using this map construct which we're going to talk a lot more about in the next lesson.

So we're going to see it in action but if you want to get a deeper understanding of Dart maps, keep going through this lesson.

And once you get to the next lesson we're going to explain it in greater detail. But it's super simple to use

and just looking at it, you'll probably already understand how it works.

Let's look at how routes are defined.

We can create a map which has a key and a value.

So it means that when the route that is / is requested, we should build the first screen or when the route that's called /second is requested then we should build the second screen.

So pretty simple so far.

Now let's define that for our app.

So let's go ahead and create a map first and we can do that just by opening a set of curly braces and hitting Enter. So inside here is going to be on a map and it's going to have some keys and some values and the keys and values will match up.

So the key is the name of the route.

So let's have a route that's just '/'.

And then we add a call on to specify the value for that route and you can see that all route expect a function as the value of the map.

So the string is going to be the ,in this case it's the name of the route. And the value is going to be a function that takes a context as an input and returns a widget.

So in our case it's going to take a context as the input and it's going to return a widget which is going to be the widget that we want displayed when this route is requested.

So the one that we want is going to be screen0 for when the route is /.

Now then we can go ahead and define some more routes . Let's have a /first.

And this is the key.

So the value for it is going to be another context that returns maybe screen 1 in this case.

So now we've associated the name of / with going to screen 0 and /first with going to screen one.

Now all we need to do is to add one last one which is /second.

And this should go to the second screen of course, so screen 2.

And we have to add this into our imports to tell it about this screen2.dart file.

Now that we've created all of our named routes.

Let's go ahead and use it.

Now you can see that we normally use a home property for our material app to define where the app should start, which screen it should display first.

But instead of this we could also use a property called initialRoute.

And we can simply just give it a string to tell it which route we want out of all the ones that we've defined.

So let's tell it to start off at screen 0 by giving it just the string that is /.

So now if I go ahead and hot restart my app, it should look exactly the same because it's still going to go to screen 0.

Now be careful though because if you have an initial route and you also have a home property defined,so let's add what we had before which is screen 0,now these two will conflict with each other.

And if I hit save right now, you can see my app will crash.

And it tells me that only one of them can exist.

So I can't define both a start and a beginning. That doesn't make any sense in code.

So let's delete the home whenever we're using an initial route.

And if you want you can just use home instead of initial route,but given that we've got all the routes define already it makes sense to do it this way.

It's much easier.

So now let's try and use these roots inside our screen0.dartat the place where we want to navigate to screen 1.

We can do this using unnamed roots by simply saying Navigator.push but instead of pushing and creating a route, we can push a named route instead.

So in this case we have our current context of where we are in the app but we also are going to define the route name which is just the string that we created here. So we can simply write /first if we want to go to the first screen and if we wanted to go to the second screen when we click on this second screen button, then we can say go to /second and this of course corresponds to the routes that we created in here.

So it's going to look at the key that matches /first to figure out what it should do from the value.

So now if we hit save and we check out our app, when I click on go to screen 1, it goes to screen 1.

When I click on go to screen 2, it goes to screen 2.

Now all of my buttons that use the navigator where I create the routes from scratch still work.

So I can still go to screen 2 from screen 1. But now in my stack I have three screens: 0, 1 and 2, all stacked on top of each other.

And if I keep clicking the back button, it'll take me all the way back to the beginning. So the navigator is a really easy way of going between screens and going between our routes. And as your app gets more complex you're probably going to have more and more routes and you're going to need complex ways of navigating between them.

So I would say that if you have a very simple app with only two or three screens, then probably creating the route at the point where you need to is probably good enough.

But if you have a complex app with four or more screens then it makes sense to list out all the routes in the beginning and make the navigation code much more expressive and easier to write in the actual screen.

So now that you've learned all about routes and navigation then guess what, we need to create a new page and we need to navigate to it.

So I'm going to right click on my lib folder and I'm going to create a new Dart file which is going to be called results\_page and I'm going to hit OK to create my new results\_page.dart I'm going to import my material package and we're going to create a very simple stateless widget in here.

So it's gonna be called ResultsPage and here we're going to simply return a scaffold that contains a app bar which is simply going to have the title of BMI CALCULATOR.

And it's also going to have a body which is just going to have a little bit of text that says hello,or whatever you want it to be.

It's just so that we can confirm when we've navigated to this new page that we've actually got the code set up correctly. Now that we have our results page,we can go back to our inputs page and at the very last part where we have a container,we're going to give it a little bit of text. So we'll add a child that is a text widget and it's going to say CALCULATE. Now our container in order for it to receive taps,we have to wrap it with a gesture detector.

And when it is tapped we're going to go ahead and create our route.

Now the challenge is in your court. Implement what you've learned about route to navigation to be able to take the user from the first screen where we have all our inputs to the second screen when they click on that calculate button. Pause the video and try to complete the challenge.

All right.

So in this case we're only navigating between two screens so we're probably not going to create named routes. But we can instead use our trusty Navigator.push method. And we've got our context as the current context and the route is going to be a material page route.

And in order to build it, we need to give it a function which takes a context as the input and it returns...

So you could either use a set of curly braces and write return or remember that the shorthand way of doing this if you have a single line of code is just a fat arrow. So an equal sign an angle bracket. And we can return our ResultsPage. And it's not showing up because we don't yet have it imported.

So let's import that at the top and we have our results\_page.dart imported.

So now if we hit save and we run our app, then you can see that when we click on the calculate button which is anywhere inside this pink area, it will take us to our second screen which has the word hello and the app bar of BMI CALCUATOR.

And this is also a good point to point out that all of the hard work that we've did in defining the theme for our app is now carried through to our second screen as well with the same primary color and the same scaffold background color and text colors. Now that we're able to navigate between our pages,the only thing left to do is to update the styling a little bit more and to actually calculate the user's BMI and display it on screen.

But as promised, in the next lesson we're going to do a deep dive on Dart maps so that you can learn a bit more about what all of these maps are that you're seeing in the namedRoute and how these key value pairs work in that.

Now if you're already quite familiar with maps or dictionaries and how they look and how they work in Dart, then feel free to skip the next lesson and go to the lesson afterwards where we continue building out our app.

But either way, I'll see you on the next lesson.

