Well you saw previously that we could parse data forwards through our routes by adding it as a property to the widget. So we could add a property here,we could call it final and we call that variable locationWeather. Because a little bit later on we're going to get the weather from a different source as well.

Now when this stateful widget is initialized, we can happen to the location screen.

And here we can add our this.locationWeather.

So now when we initialize our location screen, we have a property that is the locationWeather and we can set it to a new variable.

And this is where I'm going to parse over that weather data.

Now I can get that location data over as soon as I initialize a new location screen.

But how do I access that from my location state? This is the central problem and this is fundamental to understanding how Flutter works.

We need to display the weather information received from the loading screen inside the text widget of the location screen.

And this means getting the data from the location screen object into the location screen's state object.

Remember the location screen's stateful widget is actually a separate object from its state. And the text widgets don't live inside our location screen.

Instead the text widgets live in the build method of the state object.

However, the location screen object and the location screen's state object are linked.

In other words, the state object knows which stateful widget it belongs to. The state object has a property called widget which will point to its parent stateful widget.

Let me show you how this works in our code.

I'm going to print our weather data to the console via the widget property of the location screen's state.

If we go ahead and add a init state, so that we tap into the moment where the state object is initialized, here we can print out a widget object.

Now you get access to this widget object inside every single state object. And what it is, is the current configuration.

So a state object's configuration is the corresponding stateful widget instance.

So basically if we tap in to widget, we'll get access to the location screen stateful widget which has a property of locationWeather. So we can say print widget.locationWeather. And now we're able to pass this hot potato from our loading screen over to our location screen. And from our location screen state, we can tap into the location screen through widget and tap into that property through location Weather right here.

So two parses in order to get it here.

So let's just print it to confirm that we've actually got that weather passed over from here.

So notice there's no print statements in our loading screen and when I take a look at my console, as soon as it's moved over to a location screen, I've got that location weather printed in the console here.

So it's definitely working. I'm definitely accessing that location weather here.

So now all we have to do is to use that location weather and get the temperature and get the condition code and get the city name from it.

So I'm going to copy these three lines of code that I had from before, and I'm going to paste it into a new method called updateUI.

So I'm going to use it to update the user interface and this updateUI is going to have a single input which is going to be something called weatherData. And you can add the data type of that input as dynamic because remember that locationWeather is a var that comes from this weather data var.

So it's dynamic here, it's dynamic here and it's also going to be dynamic here when we pass it in. So now we can paste in those three lines of code and we can update it so that instead of using decoded data, we're using weatherData in order to tap into that weather data object.

So now if we wanted to use these properties inside our widget, we're going to have to create some properties inside our state object.

So we're going to have a double that's going to be called temperature and a int that's going to be the condition and a string that's going to be the cityName.

And now we can set temperature to equal weatherData, conditioned equalweatherData and city name.

So now let's reformat our code with a CONTROL + save and we now have a updateUI. At this stage instead of printing the widget.locationWeather, we can pass it over to our updateUI method and provide that widget.locationWeather as the input to that method.

So when our location screen gets initialized, we pass over the locationWeather which we can tap in to through widget.locationWeather and we call updateUI, again passing over that weather data. And then we pass it to get the temperature condition and city name out of it. Now at this point, if we try and print out temperature value, so let's hit run again and hot restart our app, you can see that the temperature that gets printed out is in Kelvin,so it's not Celsius and it's not Fahrenheit.

Now we could either convert this temperature from Kelvin into Celsius by subtracting two hundred and seventy three point one.

So let's take our temperature here, 285.61 and if we subtract 273, we get 12 degrees Celsius or 54 degrees Fahrenheit.

But this is such a common functionality that Open Weather Map actually already knows how to handle it.

So if we go into the API docs and we take a look at the unit format, you can see that it can return the temperature as a metric value or as an imperial value simply by adding a unit equals metric to the end or units equals imperial to the end.

So I want my unit in metric,so I want it as Celsius.

So I'm going to copy that and I'm going to paste that to add to our URL.

So at the very end after the API key, I'm going to paste a ampersand and I'm gonna say and units equals metric.

So now if we hot restart again, you can see that the temperature I'm gonna get printed out is now in Celsius.

And you can change that to imperial if you need the temperature in Fahrenheit. So now that we've sorted out that little niggling thing, we can now use our temperature inside our widget. And the place where we display it is right here and instead of 32 degrees, I'm gonna delete the 32 and I'm going to add my temperature variable here with a dollar sign. And I'm going to keep that little degree symbol as a string.

So now if we hit save and we take a look at our app, you can see that we're getting twelve point four degrees and the rest of the content is being pushed off of the screen.

Now nobody needs to know the temperature of where they are to that degree of accuracy right?

All we want is just the first number without any decimal places.

So how can we do this? Well instead of storing temperature as a double,we can change it to an integer, a whole number with no decimal places. And now we can create a local variable that stores the double temperature, so we'll just call that temp. And then we can store the actual value in the temperature variable as the temp.toInt and this will convert our double to an integer.

And now if we hit hot restart, you can see that when our temperature displays, we only get a whole number.

So we get 12 degrees without the rest of the decimal places.

So now we successfully parsed the weather data for our current location into our location screen and we're able to pass it to get the temperature, the condition and the city name and we already started using the temperature in our widgets. The very last thing I want to do is just to refactor this loading screen a little bit.

Instead of having to have the latitude and longitude stored up here as doubles and instead of having to create so many variables, I can simply use the location.latitude right here and location.longitude in here.

Now the only problem is that you see when we use a single dollar sign to insert a variable value, it can only take one step. So we can only take that first variable that it identifies.

If we wanted to tap into location.latitude or location plus three or do something with it, we have to add a set of curly braces around it.

So we're going to add some curly braces around location.latitude and also some around location. longitude.So this way, we can further simplify our code by deleting those two lines and also we got rid of those variables right up there. And now our code is a lot shorter and it's still just as expressive. We're passing in a latitude that's equal to the location we got back.latitude and longitude is location.longitude. And our loading screen is now pretty much complete.

So in the next lesson, we're going to explore how we can get the other part of our screen updated using the data that we got back from Open Weather Map.

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

