Now in the last lesson we used our scratch.dart file to learn more about futures and async and await.

But we don't need this file anymore so you can either delete it or you can keep it around for reference in the future.

It won't affect your project.

Now let's head back into our loading screen and if we take a look at the code now, it should make a bit more sense to you.

You can see that we're creating a new geolocator object that comes from geolocator.dart. And then we're using that object to get the current position. We're calling getCurrentPosition and this is a method that's inside geolocator. And we provide the amount of accuracy we want for this location.

And we're going to choose low so that we don't use up all the battery. And because it takes time to get the current position this method is asynchronous and it will happen independently of whatever else you decide to try and do. But because in most cases we can only proceed forwards with for example printing the position we got back or with using that position in getting some weather data,then we're adding the async keyword to modify our function. And that gives us access to the await keyword which basically says wait until this completes before you continue doing anything. If we didn't have this await keyword, then we could still have a position but it won't be an actual position.

It will be a future position.

It's just like that order number.

It will have a value in the future once this process completes. But at the point where this code is called or when it's triggered, this is just a receipt.

It's not an actual coffee.

And right now if I was to print out this position variable, it won't be an actual position that actually comes out. And we can show that by changing this from running scratch.dart to main.dart.

And now if I press run and I go ahead and click on get location, then you can see that what's being printed in my console on the main.dart, so we can close down scratch.dart if you want, is an instance of future position. This is kind of like you promise your friends to go and grab them a coffee and all you did is come back with the receipt.

Now they're not going to want the receipt.

They're going to want the actual coffee.

So that's why we're adding that await keyword in front of the method call to say that, wait for this to finish before you assign this value to the position.

So that way we actually get an actual position or an actual coffee, rather than just the promise of a coffee that's going to be there in the future.

And now if I click on get location, you can see the actual location gets printed rather than just a future.

So it's all very well and good that we can get the location when we press on this button.

But what if we wanted the location as soon as our screen loads up? Because in our weather app, as soon as we open up the app, it's going to trying and detect our location and based off that location, it's gonna get the weather.

It doesn't make sense to force the user to press a button to say get weather or get location,that seems a bit extra. So how would I call get location if I wanted it to happen as soon as my screen loads up?

Well in order to do this we have to learn about widget lifecycles.

We know that stateless widgets are basically just like very simple Lego blocks right?

You can't saw them in half,you can't change them.

You can't do anything with them unless you decide to destroy them and create a new one.

And you have to keep destroying and creating new ones every time you want a change in a stateless widget.

So for these widgets, their lifecycle methods are very simple. There's only one that you should be concerned about and that's the build method. When the widget gets built this method will be called and inside here,you will create whatever appearance or widget you want to show up on screen.

Now on the other hand, we also have our stateful widgets and we know that these stateful widgets can be combined and we can track the state using a state object.

Now that state object is there keep track of variables such as what is the configuration of my widgets,what are the properties of my widgets.

And I can change all of those variables by using a set state and it will update my app. Now in this case, the state object actually lives a lot longer and so it's got more lifecycle methods.

There is an init state, which gets triggered when that state initially gets initialized.

There's of course the build method which gets triggered when the widgets are actually built and will show up on screen. And then there's a deactivate method which gets called when that stateful widget gets destroyed.

So just as we humans are born and we grow up we go through different life stages and then we also die,so do our stateful widgets. But we can tap into each of those stages in the lifecycle if we wanted different things to happen at various times.

Now I just want to show you when these lifecycle methods will get triggered. And to do so, I'm pulling up that previous file that we had, the navigation demo that we used earlier on.

Now you don't have to write any of the code,

I just want to show you what actually happens.

It's enough to just follow along. Here we still got our screen 1 which is a stateless widget.

We've got our screen 2 which is currently a stateful widget. And I've made our app start out at a screen 1 which is the first route.

So when I run my app, I head over to the screen 1 which only has a single button that pushes me onto screen 2.

Now once I'm on screen 2, I want to be able to show you when these lifecycle methods actually get called.

Inside our state object, I'm going to tap into a number of methods that come from the parent class, the state class. One of those is the initState method.

And if we click on this initState, hold down CONTROL + J or CONTROL + Q, you can see that this is called when an object is inserted into the tree.

And this means that when we create our stateful widget as soon as it's inserted into the tree, it's going to call initState.

So this is the first thing that happens.

So inside here I'm going to add a print statement saying 'initState called'. And that will print whenever this method gets triggered. And it will get triggered automatically at a particular stage in the life of this state object.

Now the second one that gets triggered,so the second point in the lifecycle, is going to be the build method.

Now inside here, I'm also going to add a print statement saying 'build called'.

And finally there's also another method that comes from the parent class that I want to show you which is called deactivate. And deactivate will get triggered when this stateful widget gets destroyed.

So let's add a print statement in here as well.

Now let's go ahead and click run for our changes to go through. And in our console you can see that as soon as I click on this button, I should head over to screen 2.

Now as soon as that stateful widget is created and inserted into the widget tree, initState gets called.

So that gets printed in here.

And then as soon as it builds all of the widgets inside the screen which is just my button and my scaffold etc. then we get billed called. Now every time I make a change in the screen or I change one of the properties that the widgets depend on, then build will get called again and again and again.

And this is one of the most frequently used lifecycle methods.

But at the very end when I click on go back to screen 1, screen 2 is going to pop off,that means it's going to be destroyed.

It will no longer exist.

And that is the time when deactivate will get called.

So there are other lifecycle methods but these are probably the most useful and the ones that you'll actually come across and need when you're creating Flutter apps.

So we know that if we want something to happen the moment that our stateful widget is created and add into the tree, then we're going to put our code inside initState. If we want something to happen every single time our stateful widget gets rebuilt, then we'll put it into the build method.

And finally if we want something to happen when our stateful widget gets destroyed, then we would put the code inside the deactivate method.

So now using what we've learned on lifecycle methods, here's a challenge for you.

I want you to change our code so that the position gets printed into the console without tapping on any buttons.

In fact you can go ahead and delete the entire contents of the scaffold.

What should happen is as soon as our app runs and as soon as this loading screen appears inside the phone then we should see our position being printed in the console.

And this will rely on what you've learned from this lesson.

Pause the video and try to complete that challenge all right.

So currently when we run our app all we get is just the blank screen.

There's nothing inside build, just a blank scaffold.

So we're not gonna use any buttons to trigger getting the location.

Instead we're going to add a initState to our loading screen. And because we know that this is going to be triggered as soon as our stateful widget gets created,so it's the moment that this appears on screen.

Now it's important to know that initState only gets called once and it's only the moment when that state gets initialized and gets created.

But something like build even though it does get called when our widget gets built onto the screen, but it gets caught every single time that our widgets rebuild.

So every time a piece of text changes or an image changes or this animation, then build will be called again and again and again. And very often you don't want to put code in there that will get called repeatedly because it's very expensive.

Instead we're going to put it inside initState.

And so inside initState, we're going to call get location.

And now if I go ahead and hot restart my app to make it reinitialize this particular stateful widget, then you can see as soon as it goes on the screen, I get my position printed in the console.

I didn't have to press anything at all. So our lifecycle method are really useful if we want to tap into a particular moment in the life of our stateful widgets. If we wanted to save a piece of data just

before the stateful widget gets destroyed or if we want to deallocate something from memory or if we want to create a new object as soon as the stateful widget gets initialized, these are the methods that we can tap into to make our code run at a particular time. And in our case we're making our get location method run in the moment as soon as our loading screen state gets initialized which is going to be at the very start and we only make it run once. Now at the moment, we're getting our location and we're importing our geolocator all inside our loading screen which is kind of not its job right?

So in the next lesson we're going to learn more about checking to see if the user has given permission to get the location.

And we're also going to refactor all of the location related work into our location.dart file.

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

