In previous lessons, we saw how we could incorporate a drop down button with a bunch of dropdown items.

And this is of course the Android appearance,so the material design widget. And in the last lesson we looked at how we can use a Cupertino picker which is the iOS appearance for the iOS picker.

Now let's create two methods where we can either create a Cupertino picker or we can create a dropdown button.

So I'm gonna cut this code for my dropdown button and I'm going to incorporate it into a separate method.

So we're gonna call this get dropdown button. And inside this method is where I'm going to paste what I copied over, so everything that's the code for the dropdown button. And then I'm going to uncomment it and return this as the output.

So I'm going to add that semicolon at the end and also change the return type here to a dropdown button that contains string items.

So now let's hit save for Dart to reformat our code and add some comments to make that reformatting look a bit prettier.

So now we have all get dropdown button which is going to use get dropdown items to fill its items.

Well it seems a bit extra to have two separate methods just to get a dropdown button.

So let's refactor this and combine them into one.

So I'm going to cut all of this code inside my get dropdown items.

I'm going to paste it at the very top of the get dropdown button and instead of returning the dropdown items, I'm simply going to put that inside here.

So now we've combined all of this into one method which grabs the items and puts them in a dropdown button and returns it as the output.

So now it's time to do the same for our Cupertino picker.

So I'm going to cut it out of the code in my build method and I'm going to create a new method that will output a Cupertino picker and I'm going to call this simply iOS picker. And then inside this method I'm going to paste what I copied over and use this as the output, so returning a Cupertino picker. And then in the same way I'm going to take all of this and I'm going to paste it into my iOS picker and instead of returning the picker items, I'm going to use the picker items as the children of my Cupertino picker.

So now we can also delete this get picker items method as well.

So now I have a way of getting my iOS picker through my iOS picker method or I could get a Android picker maybe or Android dropdown. Maybe we should change this to a Android dropdown and because we deleted that method get dropdown items, then we have to delete it from our build method here as well to get rid of that error we're seeing in the right hand side here.

So now inside my container I can either choose to display a Android dropdown,so if I hit save you'll see that I'm getting my dropdown list being rendered here, or if I choose to I could change to an iOS picker.

So let's hit save and we see now a iOS style picker.

So wouldn't it be nice if I could simply just create a method here, let's call it get picker which simply returns a widget and the get picker is simply going to check to see if I'm running my app on iOS.

Well then we're going to return the iOS picker. But if I was running it on Android, well then I'm going to return the Android dropdown.

Wouldn't that be cool?

Well with Flutter you can do exactly that.

We can actually check in our code to see which platform our app is currently being run on.

So we can plan ahead to see that if our code is being run on an iOS device, then we could return an iOS picker.

But if our code is being run on an Android device, we can return the Android dropdown. And to check to see which platform our code is being run on we need a Dart library called dart:io and you don't have to add this to your pubspec.yaml. All you need to do is just to import it like how we imported the Dart maths or the Dart convert libraries.

Now I don't actually want everything that's inside the dart:io package to be available inside my price screen.dart. Because if we take a look at our Dart SDK and we actually go into the IO package, then you can see it's got a lot of Dart files right.

It's got things such as how to handle reading from file or how to handle the directory or how to do various other io functionality. All that I want is actually this platform.dart file because it has these boolean properties that tell me whether if I'm running my app on iOS on Android, on Windows, or Mac OS, whole bunch of things. And this is what I need.

So instead of incorporating the entire io library, I'm going to say only show me the platform class. And that platform class of course lives inside our platform.dart file and it's right here,this platform class.

That's where that capital P comes from.

But say if I only wanted a different class say the ProcessInfo class, well I could also say that we'll only show me the process info class. So this way, we're able to narrow down in a library and only incorporate into our file the relevant part, which is just the platform class.

Now there's another keyword that's quite helpful which is called hide.

What this line is saying import everything that's inside dart:io, so everything inside here, all the classes I want to be able to use it in this file but hide this thing called platform. I don't want to see this platform class.

So you're gonna be using show more than hide and you've also seen as which simply just renames this entire package to give it a name, for example when we did it for the http package. So we have a name for the http package which we can specify after that keyword as.

So there's as, there's show and there's hide.

So we're going to be using show to just expose the platform class from dart:io. And then down here in our code, we can literally use our IF and ELSE IF statement to check if the platform.isiOS. Well then we're going to return the iOS picker, but else if the platform is Android well then we're going to return the Android dropdown.

So now inside here instead of choosing iOS picker or Android picker, well we can choose both.

We can say get picker and now if I hit save, because I'm running this on a iOS, device I will get a picker shown in here.

So if we have the same app, the same code running on both Android and iOS, you can see that it detects that this is on an Android emulator and this is on a iOS simulator and exactly the same thing happens on physical devices too.

As long as the operating system is Android or iOS, then we will get the Android dropdown being rendered here while the iOS picker is being shown right here.

Isn't that cool? Now we can make our code even easier without having the need for this get picker method. If we simply just go into here where the child property expects a value and we use that ternary operator we say if the platform.isiOS, well if that is true, well then we're going to render the iOS picker. But if that's not true, so for Android or Windows phone or wherever the app might be run on, we're going to render the Android dropdown instead.

So in a single line of code, we've checked the platform and it's only pretty much the iOS platform that actually needs that unique sort of iOS style picker. And for everything else, we can just put a dropdown list into this container instead.

So that's how you can check to see what platform your code is being run on and to respond to that check

by providing a different user interface element depending on that platform.