Now in the last lesson, we vastly simplified and refactored our code.

And now it looks a lot simpler and a lot easier to read.

But there's just one thing. We know that we're currently using a reusableCard to be able to create a card that has a particular margin or a particular look or a particular rounded border.

But we're also adding it as a child to a gesture detector to be able to detect when the user taps on that card, and when that happens we set the selectedGender.

Now it's really likely that for other reusable cards, we might also want this functionality, being able to detect touch.

And even if we don't need to, there's still two areas where we are repeating the same code right?

Adding a gesture detector to the reusableCard.

So why don't we just bite the bullet and upgrade our reusableCard to also be able to detect touch?

And that's what we're going to do in this lesson. And in the process,

I'm going to show you something really cool that you can do with Dart which is passing around functions as if they were just any other object.

And I've just noticed that I've actually got a typo in the name of my file.

Over here you can see that I've written imput instead of input, but this is really easy to fix because it's not affecting any of our code.

So if you ever get a typo or a spelling mistake in your file name, simply just right click on it and click on refactor, rename and it will change all the places where this is used. So let's change that input it will search for references,search for comments and strings and it'll refactor everything to the right spelling.

In Dart,functions are first class citizens.

This means that they have a type and they can be passed around just like any other type for example a string or an int. And they can also be set as the value of a variable or a constant.

So let's take a look at this now.

So here in my DartPad, I've got two functions.

One is a function called add which simply as the two numbers that it's given.

And the second one is called multiply and it multiplies the two numbers that it's given.

Now if I wanted to perform a calculation, I could say add and then provide my numbers 3 and 5 for example, and then I would end up with a result.

So let's call it a int result equals the result of the add and then we'll print the result to the console.

And you can see that I will be able to run it as it is.

But what if I wanted to build a calculator where the user is able to tap on the plus button or the minus button and perform the calculation that they need?

Well what if instead of having to call a different function each time instead,what if I create a new function that returns an int and it's called calculator and it not only takes two numbers int n1 and int n2, but it can also take a function as an argument?

So we would be able to add it as a type.

So for a string it would be string for an int it would be an int. But for function we would just write the name Function with a capital F. And we could give it a name let's call it calculation.

And then inside the body of the function, I will perform the calculation on those two numbers n1 and n2. And then I would return the result of this calculation as the output of my calculator.

Now what I'm able to do is instead of having to call add then multiply then whatever, then in this case I could simply write the result is going to be calculator performing a function on two numbers, let's call it 5 and 8, and then the third input here is a function.

It's expecting an actual function as an input.

So I'm simply going to add the name.

So for example, add. Well in this case, it's clear that I want my calculator to perform some sort of function on 5 and 8 and then add them together.

So now if I hit run you can see that it's going to try and add 5 and 8 together.

But if I simply switch out the parameter here into multiply, as long as I spell it right, then it will multiply 5 and 8 together.

So functions are being passed around just as any other types, so if it was an int or if it was a string.

And this gives us a lot of flexibility and it's really powerful as we'll come to see. Now the other thing that we can do with functions is we can even assign a function as the value of a variable.

So for example instead of saying int calculator, I could simply create a new variable that is of type Function and I could call it calculator and I will set it to equal my function here. And at the very end, I just have to cap it off with a semicolon.

So now it works exactly the same way as before.

If I click run and I change it to add or multiply, doesn't really matter, it still works as is. My function is now stored inside a variable called calculator and I could even make this a final variable if I wished to.

Which means that I'm no longer able to change the assignment of my calculator so I can't say calculator now equals a different function for example. It'll give me an error telling me that calculator is a final variable and can only be set once.

Now notice how at the moment, I'm creating functions completely outside of any classes and even outside of the main function.

And this is perfectly legal in Dart.

You can have top level functions just like this and they are not associated with any class or object.

But let's see how it works inside a class.

So I'm going to delete all the code so far and I'm going to create a new class called Car. And car is going to have a property that is a function and it's going to be called drive.

Now this property will be initialized when car gets constructed.

So I'm going to create my car constructor and I'm going to write this.drive to say that when I construct my car, I have to provide a drive method to associate it with my car object.

Let's say that we have two modes of driving right? Let's create one which is called slowDrive, and in this case it's simply going to print 'driving slowly'.

And then let's say we have another one which is called fast drive and in this case it will print 'driving super fast.'

All right.

So now I have two functions which are top level functions. They are not associated with any class. But when I initialize my car in my main function, for example I could say myCar is equal to a new car which is constructed from the Car class.

And I now have to provide a value for the drive property.

So the drive property is going to be set equal a function right?

So it's expecting a function.

I can't just put in a number, say 2 or I can't put in a string because it needs to be of the type function.

So let's give it a function because we have to. We can either say slowDrive or fastDrive.

Well let's say that my car starts out being a slowDrive car.

And notice how I'm not actually adding any parentheses after the function.

I'm simply passing over the name of that function.

So now at this stage myCar.drive is associated with the slowDrive function.

So I could show you this by simply running print myCar.drive.

And notice again I'm not putting a set of parentheses after drive because that is the thing that will activate the function.

So right now if I print, you'll notice it'll print Closure 'slowDive'.

So that's pointing to this particular function.

It's saying that in myCar the drive property is equal to slowDrive.

But if I actually wanted to call my drive method, I would say myCar.drive.

And I would add the parentheses and any inputs if necessary.

And it's this line that actually triggers the drive method that is associated with my car.

And we're getting driving slowly printed to the console.

Now let's say that I wanted to upgrade my car right?

I will say myCar.drive, the drive property is now going to be equal to fastDrive.

And again no parentheses after the method.

But I'm simply changing the value of that drive property on myCar.

So now I've taken my car to the garage.

It's being souped up. Now it's going to turbo and it's supercharged and it's blazing.

So now when I write myCar.drive and we print out the result.

So I'm going to comment out that line.

And now you can see that myCar's been upgraded and now it's driving super fast. When I'm passing functions around, I'm just using its name.

I'm not adding the parentheses.

And when I want to call the method and actually trigger the functionality that's when I add the parentheses and any inputs if necessary. Now that we've seen how flexible functions can be and how they can be passed around just like any other object or any other type,it's time for your challenge to apply what you've learned to our code.

The goal of the challenge is to do the gesturedetector on the first icon card and also the second one and refactor it into the reusableCard.dart file. So that way we can actually pass a function into our reusableCard just as we've passed a color property or a cardChild and our function is also going to be passed as a value for one of these properties.

So if you're successful by the end nothing about your app will have changed other than the fact that the gesture detector should now live in the reusableCard and inside this build method.

Pause the video and see if you can solve this challenge. All right.

So first things first. I'm going to simply delete my gesture detector from this page. So I'm going to hold down OPTION or ALT, hit ENTERand I'm going to remove my widget from both places.

So both in the first card and the second card.

Now I just have a pure reusableCard and I'm going to pass over the functionality where I set state and change the selectedGender when the user clicks on this card or this card into my reusableCard.

So inside here in the build method, instead of just returning a container, I'm actually going to wrap the container inside a gesture detector.

And here of course I will now again have access to the onTap. And I'm going to pass over a function into my reusableCard widget.

So in exactly the same way that we did with our previous two properties, color and cardChild, I'm going to add another final property which is going to have a function as the type.

And I'm going to call it onPress. And this is also going to be initialized when I create a new reusable Card.

So I'm going to add the this.onPress over here so that I can pass over whichever function I want to be used inside my reusableCard.

And then it's going to be set as the value of the onTap.

So this means that when I create a reusableCard, I can specify a function that is going to be the function that will be called when the gesture detector detects a tap on the reusableCard.

So now I can go back to my input page and add some properties to my reusableCard. So I can add my on Press which I just defined just now,and it's going to be the same as what we would have done with the gesture detector.

We're going to add an anonymous function and inside the anonymous function, we have our set state and inside the set state,we change the selectedGender to gender.male in the first reusableCard and in our second reusable card,we have our onPress and in this case the set state is going to change the selectedGender to female. Now when I hit save on my code, nothing about my app will change. You can see that it still works exactly the same way as it did before, but now we've simply refactored the ability for our reusableCard,this custom widget that we've built based on gesture detector, based on containers, based on box decorations etc. and it's now also able to take a onPress. So it's able to respond to taps.

So we've essentially upgraded our reusableCard to also be able to detect touch simply by passing over a function as the value of the onPress property. And the result of this is cleaner code with less things being embedded and all the functionality and the design is now together in one widget.

It's always good practice to refactor your code as you go along.

This way you don't end up with spaghetti code at the end and lose all motivation to tidy it up.

You know that when your room gets to a certain state where it's just no longer possible to recover it and then you give up.

That's not what we want with our code.

We want it to be neat and tidy and performance.

Now in the next lesson, we're going to continue working on the other cards that we have in our user interface.

And we're going to implement a Flutter slider to be able to select the user's height on a scale.

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

