Now in the last lesson we learned about the gesture detector and we added it as a parent to our reusable card, at the top. The ones with our male and female icons.

And we created a method called updateColor that allows us to select one of these genders and be able to activate these cards by changing their color.

Now at the moment our code works just fine.

But it's a little bit wordy and also there's a few things that are not so great.

And the first thing that comes to mind is the fact that we're passing in a integer to represent a gender.

So we're saying that one is going to equal male and two is going to equal female.

So that's why we're checking to see if the gender is one then that means the male card got pressed.

But if the gender was two then that means the female card got pressed.

Now as you can imagine, if you came back to this code in six months time or if somebody else was looking at it, they might get a little bit confused because it doesn't make a lot of sense.

And even though we have a lot of comments here it's not great.

So how can we solve this? In this lesson,we're going to address all of this by learning a concept in programming that is super useful which are Enums. Enums stand for enumeration.

And if you look it up in a dictionary, it means the action of establishing the number of something.

And that's kind of what we've been doing here right?

We've been establishing that the male gender is the number one and the female gender is the number two.

But this is super painful and quite confusing.

But in programming, this need crops up very frequently.

Say if you're trying to encode different types of cars in your app and you say that we have this thing called carType and if it's equal to one, then it's convertible, if it's equal to two then it's an SUV,if it's three then it's a hatchback.

Now these numbers don't actually inherently have any meaning.

And we're actually assigning it a meaning and trying to remember it down the line. And trying to force other people who we work with also to follow this convention, and that's difficult.

But imagine if instead, we can actually give it a word which already has meaning.

So if we said that the first one has a carType of .convertible and the second one was .SUV or third one was .hatchback.

That makes a lot more sense and is a lot easier to work with and you can see at a glance in the code what it is that we're talking about when we're coding up these different types of cars.

If you imagine that we already have booleans which we know to be either on or off or true or false, and is a bit like a light bulb. It's very binary right?

It has two states, on and off.

But what if we wanted more states, then a light bulb can hold.

Well then that's where we kind of start using Enums. And I would say that you names are almost a bit like a blender where you have multiple options right?

You might have a high setting, you might have a low setting, you might have a medium or a juice or a blend or alter blend, whatever it may be. And it's more than just on or off.

There's many different options and that's exactly what we use enums for, when there's a lot of different options that we want to be able to encode and not just have to rely on numbers.

And we can actually give each of the options a name that has meaning. An enum looks like this in Dart.

We first start out with the keyword enum, to show that we're creating an enum. And then we give the enum a name.

So it could be carType or it could be gender. But the naming convention for enum names are the same as class names.

So there are camel cased but they start off with a capital letter. And then we have a set of curly braces that describes each of the different types.

So say typeA, typeB, typeC or hatchback, coupé, convertible.

And then when we actually need to use the enum, then we can simply writeEnumName. typeA or .

typeB. Let's head over to a Dartpad to see this in a little bit more detail.

Now let's say that we had a class of Car and previously we only had things such as number of seats or number of doors, which makes a lot of sense to use an integer for right? Five doors is literally an integer.

But if we wanted to encode a carStyle then we might have to use an integer and we have to make a note to ourselves that remember that you know if this is one then that means it's a hatchback.

And if it's a two then it means it's an SUV etc. which is quite painful. But it does mean that if we decided to initialize our car,say we create our constructor and we set our this.carStyle when we create the car.

Then we could go into our main function and we could create a new car.

So let's it's a data type of car.

Let's call it myCar and it's going to be equal to a car that's constructed.

And we have to give it a carStyle and let's say it's equal to an SUV.

My car's an SUV.

Now if you read this line of code, it makes absolutely no sense whatsoever to a human unless we add a whole bunch of comments to say that, oh actually this car style 2 means that I'm creating a new SUV.

Now it's good practice to have comments that explain the logic of your thinking of what your program does.

But it's not great when you need comments to be able to understand what's actually going on in a particular line.

And this is a classic case where enums can come in and save the day, because if we lost all of these comments and we came along to this code two months later, then this makes no sense whatsoever. So what

if we created an enum?

So let's do this here outside of any other classes and we're going to use the enum keyword to say that we're creating a new enum. And the enum is going to have a name which is going to be a capitalized camel case.

And we're going to call it CarType.

And then we open up a set of curly braces and we add all the different car types that we might want in our program.

So we might have hatchback, we might have a SUV and maybe a convertible. And let's just add a coupé just for fun.

So we've now got four types of cars and they're enclosed inside an enum and now we can use these car types.

So instead of having a car style that's represented by an integer, we can change the data type to a enum type,so our carType. And now when we initialize our carStyle, instead of writing 1 for coupé or 2 for hatchback, we can actually provide the enum as a named value.

So we could write carType. and you can see that we're already getting all of our values in our dropdown list.

So let's say that we're creating a new convertible.

Then we've got myCar, which is created from the car constructor, and the property for its carStyle is a convertible carType.

Now look at how much better that reads.

Even if I came back to this code a million years later, well maybe I'd already be enslaved to the computers.

But either way if I came back to this after a long time, this line of code still makes a lot of sense.

So let's head back to our code and here's the challenge for you. Create a gender enum at the top of file.

You can't create enums inside classes. So make sure it's outside of any of the other classes, maybe somewhere below the constants. And then use that gender enum inside our update color and inside our gesture detector to be able to say that, in this case it was the male card that was pressed and in this case it was the female card that was pressed so that we can process it differently inside this updateColor method.

So pause the video and try to upgrade your code by using Dart enums.

All right.

So first things first, we're going to create our enum using the enum keyword. And we're just gonna call it Gender but we're going to make sure it's capitalized at the beginning to show that it's not the same as a variable or a constant.

Now gender is going to have only two different values and it's going to be male and it's going to be female.

So now we can use this enum gender.male or gender.female in our code. For example our updateColor can now instead of taking a int type input, we can have it as a gender type input so we can either leave that name as gender or if you find that a bit confusing, you can change it to selected gender, just to know that it's the name that we're giving the input. And now we can check to see that if the selected gender is equal to, not one, but gender.male, then we should do all of this but if the selected gender was instead equal to gender.female then we would do all of this.

And now in our code here where we update color, instead of passing in one, we can pass in gender.male and over here we can pass in gender.female.

So now look at how much better our code reads. When this card has a tap detected, we update the state by updating and update the color and we tell it that the female card was pressed. And then it goes into our updateColor method and it checks to see if the gender was male that was selected or if the selected gender was female that was selected and then we carry out all of the rest of this code.

So enums are really useful when you have more than one option for a property.

So things such as car types or high-low medium settings or in our case genders male female etc. It's really really useful to clean up our code and make it more expressive. But still our problem is that our code is so wordy that it's actually painful to look at.

So in the next lesson, we're going to learn another thing that we get access to in Dart which are ternary operators. And we're going to clean up all of this to reduce it down to maybe one or two lines of code.

So if you're ready for all of that magic, head over to the next lesson.