Now in the last lesson we came across a new keyword, this thing called static.

Now we've seen quite a few keywords already such as class or extends or return.

And if you take a look at this URL you can see that there's actually quite a few keywords that Dart holds to be special.

And we've talked about a lot of them already such as enum or else or in or show or hide.

But the one that we're going to dive deeper into right now in this lesson is static. And static allows you to implement class wide variables and methods.

Now what does this mean?

Well previously we saw that we could create classes by using the class keyword followed by some definition of a property.

And also we could create methods inside our classes.

Now later on we could construct a object from this class such as a car from a car blueprint.

And remember we construct a new object from a class with this syntax the name of the class followed by a set of parentheses.

And it's this set of parentheses that cause the constructor of the class which by default is just an empty constructor that we get for free.

But if we wanted to create our own constructor of course we can.

And we could say something like car with this.numberOfDoors to say that when we construct a new car out of this car class we create the object out of the blueprint.

Then we have to specify a value for the number of doors property.

So let's say that my car is a 3-door car.

Well in this case it modifies how my object gets constructed from this class.

But either way in order to construct a new object, we always put a set of parentheses there to show that we're actually creating a whole new object and initializing it with properties and giving it capabilities.

But what if we had a different class?

Let's say we had a class called Square and this class has only two properties, the numberOfSides which is by default set to 4, and a color property.

Now when I create mySquare object out of the square blueprint and then I create yourSquare object out of the square blueprint, it makes sense for mySquare and yourSquare to probably have different color properties right?

So it makes sense that I should be able to change the color of my square to red and your square to blue.

But what about a property like the number of sides of a square?

Well that's gonna be common across all squares right?

For it to be a square, it kind of by definition needs four sides.

It's not really unique to the square object.

It's not like this square can have three sides but this other square can have four.

So if in my main function I wanted to know the number of sides a square has, well at the moment I would have to first initialize a brand new object out of my square just to be able to tap into that property called number of sides.

Now that seems awfully wasteful doesn't it?

It takes a lot of resources to create a new object from a class. We have to give it properties not just the number of sides, but also a color property and maybe as is in most classes it might have lots and lots of methods and lots of things going on.

So it seems like a lot of effort to create a brand new square object just to be able to get a property on it such as the numberOfSides. Now you might say that why would you ever want to know the number of sides of a square?

It's pretty obvious isn't it?

It's gonna be 4. Well let's create a different class to make the use case a bit clearer.

So let's say that I had a class called a Icosagon, just make sure that I spell that right.

And let me ask you, how many sides does an Icosagon have?

Well according to Wikipedia, it has has 20.

Now you might not be able to remember that, you know just off the top of your head. So you might actually legitimately have a real need to be able to use or print out the number of sides of an Icosagon. So you might need to initialize it and then we tap in to number of sides and then we print that out and we get, oh yeah it was 20.

Of course now I remember.

But again, same situation. We're having to initialize a whole Icosagon object where it's never gonna be used again. We're not even bothering to say this object into a variable so that we can tap into it at a later point.

And you've seen code like this before where we've needed to create an object just to be able to get the value of one of its properties.

So how can we be more efficient with our code and with the resources that our code is going to run on?

Well in this case, we can change number of sides into a static property.

So to do that we just add the word static in front.

And this is of course a modifier which now modifies this variable from just a bog standard, what we would call an instance variable, because it's associated with an instance of this class, so the Icosagon object that gets created. Well now by adding that static keyword in front, well we've now turned it into a class-wide variable. And to use a class variable, we don't actually need to create an object to be able to do that.

We could just say, "Well what is the property of this class?"

And if you think about it, whenever we create a new square it's always going to have 4 as the number of sides.

Whenever we create a new Icosagon, it's always going to have 20 as number of sides.

This is not something that will change from object to object.

It's going to be universal for all objects created from that class.

So numberOfSides makes the perfect static variable.

Now let's create another class.

So I'm going to create a class called Circle.

And if you remember anything from high school maths, to do anything with circles, it's pretty essential that you use the constant pi right?

So whenever we need to work with circles or anything that's in that category, we probably want to have pi as a constant handy. Well let's create a constant that is of type double called pi in our Circle class and let's set it to 3.141592.

I think this is as many decimal places I can remember for pi.

Now as soon as we do this, we get an error telling us that only static fields can be declared as a const inside a class.

Previously when we've been using consts, we've always been creating it outside of classes just as some sort of free floating constant, usually in a constants file.

But what if we actually wanted to associate a constant value with a class?

Well in order to do that you have to turn it into a static const.

Why is that? Why does it have to be static?

Well the reason is because by having a const that already has a default value, that means that every object that gets created from this circle class will have the same value for this property called Pi and it will never be changed.

So if you think about that, that sounds an awful lot like how static variables work right?

So it doesn't make sense to create a constant that is a property on a circle if it's not going to be static.

So whenever you want a constant value to be associated with a class, you have to make it into a static const.

Now we've only been looking at applying the static modifier to properties concepts and variables but you can also apply it to methods.

So let's say that we had a method called workOutCircumference and we had to provide an input in the form of a radius of the circumference of the circle's circumference that we want to work out. And inside the body of the method we're going to work out the circumference

By using that age old formula of two pi r, so 2 \* the value of pi times the radius. And that for every circle will equal the circumference.

Finally our method is going to print out the circumference into the console.

So now if we go back up to our main and we comment out the previous print function, we go ahead and create a new circle.

Then we can call the workOutCircumference method that comes from that class and let's provide a radius.

So let's say that the radius is equal to 15.6, why not?

So now if we hit run then you can see that we get 98 point something printed into the console as the circumference of our circle with a radius of 15.6.

Now in this case again I'm creating a circle just to be able to use that method.

That's associated with that class.

I'm not going to need that circle class later on because if I did I would have created a variable that would have held on to that circle object.

And so that later on I could use that circle object again in another scenario or be able to access one of its properties and that's how proper objects are used right?

We store an object inside a variable and then we use it again and again for various tasks. But very often you find yourself just needing a method or a property that comes from the class.

And in this case whenever you're writing code like this whenever you're constructing an object in line just to be able to immediately use that object's method or property, then you would probably benefit from adding a static keyword to whatever it may be.

And in this case we can add that static keyword to our method as well.

So now it becomes a static void and so this method is now a method that's associated with the class rather than with the object that's constructed from the class.

So instead of creating a brand new Circle object just to be able to use that method,

I can just use the class method like so and I don't have to construct anything.

And it's far more efficient this way if I just wanted to be able to use that method that's associated with the circle class.

Now coming back to our example in our welcome screen, if I want to be able to tap into that welcome screen id in my main.dart, then unless this was a static variable then in order to just do something very simple like that I would have to instantiate an entire welcome screen object and that involves creating a welcome screen state and of course that means we have to build our widget which has scaffolds, padding, whole bunch of things that's getting created just so that I could access that one measly string. By changing it into a static variable, I can now bypass all of that heavy lifting and instead just hop into that property that's associated with my class, so far more efficient. And even better, I can turn this into a static const so that I can't accidentally change it somewhere else in another class such as my main or my registration screen so that that stays always as this value.

So we only have that string as a string in one place and everywhere else that we refer to it, we use that static property.