So in the last lesson, we managed to get our geolocator to work and grab the current position of the user's device to a low accuracy level.

So something like two or three blocks kind of range.

Now at the moment all that we're doing is using that one method,but we have to realize that there's a lot of cases where this might fail right?

So for example if the user didn't give us permission to get their current position, when that pop up came up to ask them for location permission they said no.

There's also other problems such as if the use'rs in a tunnel and it's physically impossible to actually get any signal from them to get their GPS address.

So in order to be able to handle all of these errors that might arise we have to learn about how Dart handles exceptions.

Now we know that when you write bad code, then our code usually won't even compile. Before it gets to run the errors get caught, they get underlined and it'll tell you there's a problem with your code don't run it and it'll refuse to run the app.

So for example let's say that we created a string called myString and we set it equaled this string,so inside single quotes the number 15.

Now at this stage if I tried to add the number five to my string, so I'll write myString plus five.

Let's try and do some basic arithmetic with strings and numbers.

Well it tells me it can't do that because this is an integer and this is a string.

It doesn't know how to combine different types.

This is what we would call a compile time error.

These errors are really easy for us developers to catch because our app won't even run and we'll definitely be alerted to them.

When something unexpected happens while your code is running well, then what happens? For example in Dart there is a method that is called parse and it allows us to take a string and turn it into a double.

So for example we can say double.parse and we can provide a string.

So I'm going to provide myString as a double.

And then this method will turn that string into a double.

So 15 as a string becomes 15 as a number.

Now let's store that inside a double type variable and we'll call it myStringAsADouble.

We'll set it to equal the output of that method.

Now we don't have any errors anymore because the job of this method is turn strings into doubles.

And in this case, it's pretty easy. 15 as a string if you remove the single quotes, it becomes a double right?

So we can even print this out and we can say myStringAsADouble and we can even add five to it if we wanted to.

Now at this stage there's no compile time errors because all the data types are what they should be.

And if I hit run, you can see that my string 15 gets turned into a double 15 and then it gets five added to it.

And that gets printed in the console.

So that's pretty good.

But what if I wrote some bad code and I gave it something that it can't actually work on?

So if I turn myString into 'abc' and I try to get my code to pass that string abc into a double and try to add five to it.

Well what happens in this case?

Now notice how we're not getting any compile time errors because all that it checks at compile time is that for the inputs that you're putting in, it matches the data type that it expects,so in this case parse requires a string data type as the source,and that's exactly what we're providing. We're providing a string.

Now it doesn't know that we're actually tricking it and trying to give it an actual piece of text to try and turn into a number, a job that a human can't even do. If you give me the words abc and told me,'give that back to me as a number', I wouldn't really know what to do either let alone our computer.

But notice how this error is going to happen in the future while our app is running.

So this is fine.

And this is fine.

But at the point when it tries to turn a piece of text, abc, into a number, that's where it's going to trip over and fail.

So that's going to happen while the app is running or during runtime.

So now if I hit run and start off our runtime, you can see I get an exception that's not been caught.

So an uncaught exception and it tells me that abc is not actually a double.

You can't turn this into a double or into a number.

So this method here parse, it's really relying on you being truthful and being good and always giving it a string that can be turned into a number.

But as our code grows and as it gets larger, we don't always know what's being passed around. And we might make an honest mistake just like this.

So we don't want our app to crash. But instead we want to make sure that we catch our exception. So in answer to that question, when something unexpected happens while your code is running, what should you do?

Well in that moment the program is going to throw out an error and it's your job as the programmer to catch that error and handle it. And to do that we can use try and catch.

So we're going to take the parts that we know can fail,so for example this part, and we're going to wrap it inside a try block. So we're going to say, 'try to do this'. But if it fails, then we're going to catch the error inside a catch block.

So we're going to provide catch and then we're going to give the exception a name, so you call it exception, or in most cases you're going to see this written as e short for exception or error.

Now inside this catch block we can actually print out the error. And now when I run my code, you'll see that instead of having a uncaught exception, my exception is now caught and it's now printed into the console.

At this point some of you might ask, well what's the difference?

Same thing happened right?

You had an exception and we know what the exception is when we look in the console,what's the point of adding in that try and catch block? Well,the difference is that if an exception occurs and it's not caught inside a catch block, your code will crash.

That means your app will crash and your users will see a black screen or your app will exit.

But if you do catch it, well you can respond to it and you can plan ahead.

So let's see try and catch in action in our Flutter app.

Let me quickly show you an example of how unhandled exceptions lead to a crash.

Also let me show you how we could recover from an exception and avoid a crash.

I'm going to delete everything that's currently inside my build method for the loading screen. And instead I'm going to return something based on a result, and that result is going to be based off a string called myMargin and it's going to be equal to 15, but as a string. And then I'm going to return a scaffold widget which is going to contain a body parameter and for the body of the scaffold, it's going to be a container that has a color of red and it's also going to have a margin property which is of course created using the EdgeInsets.

And I'm going to have a margin on all four sides.

Now the value that I'm going to use in here is going to be a double because it expects a double.

And I'm going to use that parse method to turn my string, myMargin, into a double so that it can be used as the margin for my container.

So now at this stage if I go ahead and run my app, let's see what happens.

You can see that once the app is up and running,I have this red container and it's been given 15 pixels ofmargin from all four edges.based on this line of code.

Now let's break our code.

Let's change this margin to an actual piece of text which we know can't be parsed into a double.

Now at this stage as soon as I hit save, you'll see that hot reload happens and this line of code gets carried out and my app crashes right?

I see nothing onscreen and I see a lot of errors inside my console.

Now had I been prepared for this and I knew that some at some stage maybe it was possible that something that couldn't be passed into a double, was put into here as the input.

Well then I could've caught that in an exception.

So let's move this line of code out and let's create a double which is going to be called myMargin AsADouble.

And we're going to set it equal to double.parse(myMargin)

Now we know that this is the offending line of code,so this is something that could break. Well so in this case we're going to say try and do this, but if you fail then we're going to do something else.

So we're going to catch the exception and we're going to print the exception. Also we're going to provide an alternative widget to render.

So let me copy that and paste it in here.

It's also going to be a scaffold and it's going to have a container.

And in this case, myMargin is simply going to have a default value of 30. So in this case we're saying try and turn this margin into a double.

And if you can, then use myMarginasADouble inside this container that you're going to render but if there were any errors with this, then try and catch that inside a catch block and not only print the exception into the console but also provide an alternative, so provide a different container.

Now at this stage if I hit save and let's reload our app, you can see that even though this code is still failing and I'm getting my exception printed into my console, 'Invalid double' That's abc. I'm getting an alternative reality being rendered on screen.

Now if we look at our code, it's actually doing something really simple.

It's simply providing a default value for our margin when parsing myMarginasADouble has an error and myMargin doesn't actually have a value.

Now we can make this much simpler using another construct from Dart that's really useful. Instead of creating two alternative scaffolds, let's take this out of all try catch block and keep it really simple.

So all it does is it tries to turn our margin into a double and then if there were any errors, it'll catch it.

And our build method is simply going to try and return a scaffold

using this myMargin as double.

So let's move that out of the try block, up here, and it's going to start out with no value.

So it's going to start out with null, but it's gonna try and turn myMargin into myMarginasADouble by parsing this value myMargin.

Now if there were any exceptions it's going to print the exception but we're still going to return a widget nonetheless.

Now in this case it's going to use myMarginAsADouble. But if we run our code right now, margin as a double tries to be turned into a double but because it's a piece of text, this fails and the catch block gets triggered.

Now in this case we can either say if there were any errors turning into a double, let's change myMarginAsADouble into a default value, let's say a 30 right? Now in this case, you can see that the margin of 30 gets given to our red container and it's using it from here because we know that we definitely can't convert abc into 30 right?

So it's actually the catch block that's being triggered and we see the exception in the console as well.

But there's actually a much easier way of doing this checking to see if a particular variable is equal to null,in that case providing a alternative value.

And we can do it right here in line with our margin property.

So instead of trying to provide a value here, we can say myMarginAsADouble and then we add two question marks.

And what this syntax says is that if this has a value i.e. it's not null, then use it.

But if it is null, then use my alternative instead.

So use 30 in its place.

And this line of code works exactly the same as before.

You can see that the margin of 30 is being applied here because myMarginAsADouble at the moment is equal to null.

And we're also getting some exceptions thrown which we could handle in our catch block.

Now let's say that I change this to an actual number that can be parsed so that this line of code actually succeeds.

Then when I run the code then you'll see that the margin changes to 15 instead. It's preferentially using this value but if it actually is equal to null,then this operator makes sure that we have a default value in its place so that we don't end up having a blank screen or problems rendering our margins. So in this lesson, we saw the try catch block where we can enclose a line of code that might fail and might throw an error, inside a try block. And if it does throw the error then we'll catch it inside our catch block. And we can either handle the exception by printing it or by putting it onto the screen to alert the user or we can provide an alternative reality in the catch block.

We also saw the null aware operator and in this case what happens is that we have some sort of variable that we know could be null at some point, it might not contain a value.

So in that case we add two question marks saying that if in the future when we try to use this variable and it's equal to null, then use this default value instead.

But if it does in fact have a value then use that value, so that we're aware that a variable can be null,and if it is, we provide an alternative.

So let's apply what we've just learned to our code inside getLocation. So we know that this is the line that probably could fail in certain cases, say if we didn't get permission to get the location or if the user is unreachable and we can't figure out their GPS and even the device doesn't know their GPS address.

So let's wrap this line of code.

So Android Studio is putting it onto two lines just to make it easier to read.

But this is in fact all one line of code and we're going to wrap it inside a try block.

So I'm going to add the try keyword, add a set of parentheses and then I'm going to wrap everything that depends on that position inside the try block. And then I'm going to follow that with a catch block and inside a set of parentheses, I'm going to put the exception or simply e for short. And inside the catch block we can print the error or exception if it does occur, or if we're being more sophisticated we can actually try and figure out what the error is and decide on an alternative route based on that.

So as Benjamin Franklin says by failing to prepare, you're preparing to fail.

So we have to make sure that we always prepare for the worst case scenario when something doesn't work or something might throw an exception and we have to try and catch it so that we can handle it or deal with it or provide an alternative path.

Now let's talk about the last piece of the puzzle of the try catch block, namely what actually triggers an exception?

Where do these exceptions actually get thrown from? The key to understanding this is the throw keyword.

A method can throw an exception and by throwing an exception, we can handle the exception in a try catch block.

Let me show you how to create a method that throws an exception.

So here I'm going to create a new method called somethingThatExpectsLessThan10.

So this is really long.

And it needs to take an input which is an integer and we'll call that

n. Now inside the body of the method, we have to do something with this integer.

And but we should expect that in all circumstances, n should always be less than 10. And if n was greater than 10, then something terrible has happened.

And in that case we have to throw an exception.

We can't work with something larger than 10.

So in this case, we would use the through keyword to throw out an error and our exception is going to read, 'n is greater than 10, n should always be less than 10'.

So here's our exception.

And now let's say inside our getLocation instead of trying to print the position and get the position from our geolocator, we instead use our method somethingThatExpectsLessThan10 and for n, we're going to put in 12 which clearly is going to violate our rules and it's not going to work.

So now if we hit run and we check out the console then you can see that we get, 'n is greater than 10, n should always be less than 10' thrown as the exception.

But if we didn't handle the exception if we didn't have this try and catch block and we tried to run this code that throws an exception under certain conditions, then our app will actually crash. So let's get rid of that method here and here.

And let's bring our code back with the try catch block. Now going back to our getCurrentPosition method,let me show you where this method actually throws an exception. For this,we'll need to peek inside the geo location package.

If we take this method and we hold down COMMAND and we click on it, it takes us to where it was declared or where it was created. And you can see here in this method that's created from the geolocator library,you can see that it checks to see if permission is permission status granted, so the user actually allowed you to access their position,and then we try and get the location. But if they didn't let you have their permission, then it would handle the invalid permissions.

And here we see how the exceptions are thrown.

There are two cases when an exception is thrown by our geo location package, permission denied and permission disabled.

This is the throw keyword which triggers the exception inaction.

This is why we wrapped the getCurrentPosition method in a try catch block.

