Now these days it's hard to come by a good app that doesn't feature any animations.

It seems like every single app is taking advantage of the improved resources of mobile phones and they're all giving their user interfaces some sort of animation that makes it pop or jump or rotate and it looks so much flashier and so much more engaging when we add animations to our apps. So in the next few lessons I want to focus on how to create animations in Flutter so that you can also create delightful experiences for the user by incorporating these animations into your Flutter apps.

Now when we're building animations using native API such as Android and iOS, because a lot of the SDK were built in an age where animations were almost like an afterthought, the initial Android and iPhones didn't have the resources to be able to animate things with ease.

So these animation libraries actually came much later and if you look through a tutorial for how to build an animation in iOS or Android especially something a little bit more complex a little bit more involved with curves or different timings, you can see that it actually usually involves quite a lot of code. But because Flutter was built much more recently, it knows how important animations are to modern apps.

And so when implementing animations using the Flutter SDK, you'll find that it's actually incredibly simple to add animations to your Flutter apps. So there's no reason to hold you back.

Now in this lesson we're going to talk about one of the easiest and most commonly use animations in Flutter apps, and these are hero animations.

So what does a hero animation look like?

Well it usually takes place on a screen transition.

So when we go from page 1 of the screen to Page 2 of the screen if they have a shared element, then that element will transition between page 1 and page 2.

So that element, being an image or an icon or whatever it may be, has to be present on both pages and it will have a continuous transition as the user navigates from page 1 on to page 2 of the app. Now because they have that requirement of having a shared element, if you come from Android you might know this animation as a shared element transition, and it looks like this on a real Flutter app.

So we're transitioning from one screen to another screen and that shared element is being tweened or being animated steadily across the two screens. And in this lesson we're going to implement hero animations into our app.

This way when the user goes from the welcome screen to the log in screen, that little lightning icon will transition onto the second screen in a smooth transition like this, so this is what we're going to implement. And to do that we're going to need to first read up a little bit on hero animations. But as you would imagine, we're going to be using a widget and that widget is called the hero widget.

And to implement a hero widget animation, all you need are three ingredients.

The first ingredient is two hero widgets, one widget on the first screen and another on the second screen.

And then those hero widgets need to share a tag property, so they will both have a tag property and you need to set it to have the same value on both hero widgets.

And finally we're going to be navigating between the first and the second screen using a navigator-based screen transition such as push or pop. And that's all you need to implement the hero animation, dead simple.

So let's go ahead and do that for our app.

So in the welcome screen we know that we have this image which shows our logo, which is that lightning image.

Now here, instead of having our container just be a bog standard non-moving container, we're going to wrap it inside a hero widget.

So it's called Hero and this widget has only one required property and that's the tag. Without the tag the hero widget won't be able to figure out which other hero widget for it to animate to.

So let's go ahead and give it a tag and let's call this logo just so that we know what the hero widget contains, which is our logo image.

Now notice that in this first hero widget, our container has a height of 60 pixels.

Once we decide to click on the register button, we're going to be using the navigator to transition to the registration screen.

So if we head over to our registration\_screen.dart and locate where we have that shared element which is our image of the same logo.

Well in this case our container has a slightly different height.

It's much larger, it's 200 pixels instead of 60.

So let's wrap this container inside a hero widget also, and let's give it a tag and make sure that the tag matches precisely with the previous tag for our previous hero widget.

So this is the starting hero widget and this is the final ending hero widget.

And because they have the same tag, then Flutter knows that this is the end appearance and this is the beginning appearance.

So notice the only change is the height or the size of our image.

That's all we need to be able to implement a hero animation.

And if we go ahead and run our app, you'll see that we actually get a pretty sophisticated looking animation for pretty much very little work that we've done just now.

So because my hero widget lives in the registration screen, I'm going to go ahead and click on the register button and I want you to watch this little image, our logo image here.

Notice how it transitions from being a small image to a much larger image.

And that image is always on screen even as this first screen is being covered up.

So that image is almost hovering in midair when it animates and transitions to the next screen.

So now here's a challenge for you.

We've implemented the hero widget that goes from the welcome screen to the registration screen.

Try and figure out if you can get the same effect to happen when we click on the log in screen for that image to also perform that hero animation when we go to the log in screen.

So pause a video and try to complete this challenge.

Now we have our starting point for our hero animation in our welcome screen.

And as long as we create a hero widget with the same tag, then we'll be able to transition it to any screen.

So we could create a hero widget and wrap our container with the image on our log in screen inside that new hero widget and give it the same tag as the one that we had on the welcome screen.

So notice how this tag is now present in three places, log in screen, registration and welcome screen.

There's no limit to how many hero elements you can have with the same tag.

But this means that when we go from welcome screen to log in screen, Flutter can find a hero widget with the same tag, logo here and logo here.

And that means when we run our app again, we should now be able to implement the hero animation both when we click on log in as well as when we click on register. And all we needed was pretty much just two lines of code, super simple. Now in the next lesson we're going to look at some more custom animations and we're going to be using the animated controller to be able to define a more complex animation, such as how long do we want the animation to go on for or how do we want the animation to unfold.

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

