Now it's all very well and good using a container to lay out a single widget, such as a piece of text or a image. And we can define it padding, its margin, its size, its background color etc. But what if you had lots of different things that you want to lay out relative to each other?

Well, in that case, you need something that can take lots of children, and not just a single one.

And the easiest way to do this is using columns and rows, so that we can lay out our widgets in a vertical direction or a horizontal direction, and determine how they should be sized.

Going back to our widgets catalog and looking specifically at the layout widgets, we're gonna go down to the multi-child layout widget section. And you can see the most commonly used ones to layout multiple child widgets, are rows and columns. Rows go horizontal, columns will go vertical. Let's go ahead and create a column of our own.

What if instead of only having a single container, I had lots of different containers? So I'm going to go ahead and get rid of the margin and padding, so we don't confuse ourselves, and I'm gonna name it

Container 1.

So here's Container 1, and it's white.

And I'm gonna collapse my container. So you can see that all the code that's related to setting the properties of our container, is all enclosed in these parentheses. When I click on it, I can expand it. But I can also compress it like this. Instead of having the child of our safe area,

So what's inside our safe area being just a container, I'm going to cut that and paste it down here, for a little bit later on. And instead the child is going to be a column. Now a column crucially, doesn't have a child.

It has children, because we can put lots of things inside our column. And it's got this angle bracket to tell you that this is actually a list of items that we can put in here.

We can put lots of widgets in between these square brackets. And what we're going to start with is that container that we moved out just now.

So I'm gonna hit COMMAND + X to cut and paste it in here, and then I'm gonna collapse it down.

And at this point when I hit save, you'll see that nothing much really happens on screen because we have a column that only really comes into action when you have more than one widget.

Let's leave our first container and let's build another container as the second child of our column.

Now this container is again going to have a width of 100 and a height of 100. And notice that when you add a comma to the end of say you know every line, and you hit save, it makes it really easy for Dart to know how to lay out your code.

Let us also give our container a color so that we can actually see it.

So let's change this to a blue color.

And hit save. And you can see we now have two containers onscreen, and I'm going to give this a child which is a piece of text, and I'm going to tell it to write the words 'Container 2'. So we have container 1 and container 2.

And I'm simply going to copy this container to create a third one.

And our third one is going to be red. And it's going to be container 3.

So now we have three containers all a 100 by 100.

All with different colors, and it's being laid out in a vertical direction. Which is because, we have all the containers inside a column. Now our columns will automatically try and take up as much of the vertical space as possible.

So if we go into Flutter Inspector and we check out what the column actually looks like, you can see it's this box right here.

It's trying to take up all of the space that is available vertically. But horizontally, it's limiting itself to the size of its children.

We can change this however.

We can change its mainAxisSize, and main axis for a column, is of course vertical.

So let's change this to the minimum size.

And you can see now, when I hit save, my column actually shrinks to be the size that's only required to fit my three containers, all of its children.

Currently, our column is laying out its children from top to bottom, so container 1, 2 and 3.

Now what if we wanted to go the other direction instead?

Well, we can change its vertical direction.

We can either change it to go up, which means it goes from the bottom to the top.

So here's 1, 2 and 3 . Or we can change it as down, which is the default, where it goes from top to bottom.

Now what if you didn't want to change the direction of it, but instead you wanted to change the spacing between our containers.

Well then you would use something called mainAxisAlignment, and by default all of our containers are aligned to the start.

So it's mainAlignment.start. And this will try and place the children as close to the start of the main axis as possible.

So this is a vertical axis, and this is the start of the vertical axis.

However we can also change it to end.

So if we had MainAxisAlignment.end then our containers will all be bunched down here, towards the end of our main axis.

Now you can also move it all to the center. And you can use a variety of these to determine how you want your screen to be laid out.

Now what if we didn't actually want all of our children to be bunched together like this, at the center at the start or at the end. What if we wanted them to be spread out a bit more. Well we can use something that's called 'spaceEvenly.'

And this will calculate the amount of free space that the column has, and it'll space all of our children evenly throughout the column.

But what if we wanted something to be at the start and something to be at the bottom, and then another one in the center?

Well we would then use spaceBetween instead.

Now what if I wanted to change the way that my containers are aligned on the horizontal axis?

Well in this case, instead of using the main axis alignment property, I would be using something called the cross axis alignment.

And this is a cross or perpendicular to the main axis.

So for columns, they'll be the horizontal.

Now here, I also have access to the start CrossAxisAlignment, and also end CrossAxisAlignment.

But you'll notice that if I just go ahead and put this in, and I hit save to hot reload, nothing really happens on the horizontal axis.

And the reason for this is that my axis alignment is actually trying to align to my children.

So that means that say, if I had a child that was much wider, let's say 300 across, then all of my children will align themselves to the right edge of each other.

But if they're all the same size, then they're already aligned on the end, so they won't pop over to the right side.

In this case.

Now if you wanted all your three containers to go over to the right, one way of doing this might be creating simply a invisible container, where you give it a width of infinity, and to give it a width of infinity, we write double.infinity

And what this will do, is it will create a container down here.

You can see it has no height.

So it's just got a width, and it's as wide as the screen will allow it to be.

So if I give it a small height so that you can see it more easily, in this case, it's just this last bit.

It doesn't have a color.

So it's completely invisible but it pushes all of our children to the right, because we're now all aligning to the end of our column on the horizontal axis and one of our containers is as wide as the screen allows now.

What if instead of doing this, you wanted actually to stretch all your containers, so that they fill the entire width?

Now you could provide a width for each of the containers, to be double.infinity

So let's change the width to

So now, all of my containers will stretch to the entire width that's possible. But there's a much easier way of doing this actually, by using a property that comes with our column. And instead of using Cross AxisAlignment.end, we can use CrossAxisAlignment.stretch. And this will stretch our children so that they can be the best versions of themselves, or in this case, actually stretch them across the screen.

So you can see that even though I've reloaded, everything looks exactly the same.

And I no longer need to provide a width property for my containers because it's being stretched by its parent, the column.

Now what if you wanted a little bit of spacing between your containers. Because at the moment, they're all very much in each other's business and right next to each other with no space in between.

Well if you knew exactly how much space you want to have between your containers, you can use a Sized box. And a sized box can have a height and width of course.

But in the case of where it's being put inside a column, which is aligned vertically, then we only really need to provide a height. So we can give it a height of 20, and leave it width property as zero.

And you can see, this goes in here between container 1 and container 2, and gives us a little bit of spacing between those two things.

And it helps us when we want a little bit of space between our children. Now, everything that I've spoken about columns also apply to rows.

So in fact, I can straight up just change our column to a row, and you'll see it does exactly the same thing on the horizontal axis.

So in this case though, we're saying that the cross axis alignment.

So for a row, the main axis of is of course horizontal but the cross axis alignment.

So the vertical alignment has to be stretched so all our containers are stretched vertically.

And for all the children, they each have a height which doesn't really get taken into account anymore because it's being stretched, but we can also give it a width for example.

So let's say we give it a width of 30 for the first one.

You can see that gets affected, because my containers are being aligned on the horizontal axis.

Now you can see that my sized box disappeared, because I'm giving it a height but that doesn't really matter when it doesn't have a width.

So in a row, in order to give it space, we're gonna be using a sized box with a width in order to give it a little bit of spacing.

So the best way of getting to know columns and rows and containers is really to try it out for yourself.

So in the next lesson, I've got a challenge for you.

I've got a layout that I want you to replicate using containers, columns and rows and everything that we've learned about so far.

Now while you're playing around with this, as humans, we tend to be quite visual. And it's usually easier for us to see what something will do before actually writing out the code.

So I recommend pulling up this Flutter layout cheat sheet that's created by Tomek, and all of course linked to it in the course resources. And what he's done, is he's taken things such as the column and row, and he's tried to show you what it looks like before you write the code.

So in this case, when you use main Alignment.start on a column, this is what it looks like. When you use main alignment don't start on a row, this is what it looks like. And you can see that he's put in all of the things that we've talked about, at least for columns and rows, and you can identify it visually before you commit it into writing the code. In order to complete the challenge, head over to the next lesson and check out the layout

I want you to achieve. And then go ahead and write the code yourself, and see if you can make it work and make it look exactly the same.

So, I'll see you on the next lesson.