Now in the last lesson, we created the skeleton of what will form the basis of our input page for our BMI calculator.

And we saw that when we created our reusable card and we extracted it as a separate stateless widget,that when we create a property if we omit the word "final", we get an error that tells us that the reusable card class or the class which inherits from, which is a stateless widget, is marked as immutable.

But one of its instance fields are not final and it points to our color field which obviously doesn't have the final keyword.

Now there's a lot of words that get thrown around in programming and very often they mean very similar things but people just choose to use different terms for it.

If you're learning programming for the first time, it actually makes sense to keep almost a dictionary or a glossary of terms just to be able to understand what each of the terms mean. And in the previous warning, it said of something called an instance field.

And it's important to note that when people talk about instance variables or instance fields or properties,they're actually talking about the same thing. And all it refers to is just the property that we create that we can change inside the class, or when we construct the class we can set it to have a different value.

Now the other term that we saw in that warning is immutability.

Well what does immutability mean?

Well mutable means it's changeable. So immutable means unchangeable and the things are unchangeable when it comes to Flutter development are pretty much all of our stateless widgets. The widgets are kind of like a Lego block and each of the blocks itself can't be changed.

You can't really break a Lego piece in half or try to saw it.

Well you could probably saw it in half, but then that's not really in the spirit of the game is it?

So even though each of the lego blocks are immutable, they can't be changed.

So how do we make changes to our app which is built up using a whole bunch of immutable or unchangeable blocks?

Well, we can simply take one of these immutable widgets that needs to be changed, destroy it and in its place, build a new one that has the changes that we want.

What that means is that when we update a particular widget on screen, what happens is the old widget because it's immutable, can't be changed. So it actually gets destroyed and a new one gets created in its place with the update.

So for example in this case, the widget used to have a white background.

And when that widget needs update to have a blue background, then it will get destroyed and a new one with a blue background will get created in its place.

So this is how immutability works.

And speaking of immutability, stateless widgets are immutable. They can't be changed. Their states don't change because they're stateless.

So all of its properties also can only be set once and then it's kind of set in stone. If you want to update it, you have to create a new one and pass in a new color for the new reusable card.

So that's why we have to add that keyword final in front of the property in order for this to only be set once and then it cannot be changed.

It's the final value of the color of the reusable card.

Now we're talking about immutable,there's actually two different types of properties that can both be immutable. And you'll commonly see these two different keywords used across your Flutter projects.

And it's very very easy to get confused between what they actually do.

So I wanted to spend this lesson talking quickly about what is the difference between the final keyword and the const keyword.

And when you mark something as final or const, what it actually means.

So round one, fight. In order to do that,let's head over to DartPad and let's create a const and a final.

So we know that we can create a new variable by writing something like int myNumber = 2.

Now in this case, the first keyword marks the data type that the variable can contain.

The second is the name of the variable.

And then after the equal sign is the value that's contained in the variable.

We've seen this lots of times. But if we wanted this variable to be not variable anymore, if we want it to never be able to change, then we can have a choice of using a constant or a final.

Either way, we would add those keywords at the very beginning.

So let's create a const, and let's rename this to myConst. And then let's create a final int, and let's name this myFinal and I'm going to make this equal to three.

Now the right hand side doesn't really matter other than to be able to show you that if at a later date,I decided that I actually want myConst to hold a different value,so I wanted to write myConst now equals four,then I will get an error because constants are immutable. And you can see down here, constant variables can't be assigned a value again. And it's because it already has a starting value, and that will never ever change.

It will always hold on to that value.

This particular myConst can only be two and it cannot be changed after it's created.

Now the same thing applies to final.

So if we want to change myFinal to let's say six, it also gives us an error.

Final is a final variable can only be set once. In both of these cases, they're immutable.

But if we had on the other hand, a variable a normal variable.

So let's bring out myNumber again, and we wanted to set myNumber to be a new value say four.

Well that's of course possible because this is how variables work.

They can be created and then at a later date, they can be changed.

They can be varied.

This is why they're called variables. But for constants and for final variables, they can't change in value.

So the immutability part is usually pretty easy to grasp.

You assign it a value to begin with when you create it and then it has to be that value from now on. You can't change it ever again.

But the hard part is knowing when do I use const and when do I use final?

Because at first glance, it seems like they pretty much do the same thing right?

Well there's some subtle differences between the two of them and that is what we're going to explore in this lesson. So if you read the Dart language tour, it will talk about final and const. And it will say that a final variable can be set only once whereas a const variable is a compile-time constant.

So what does this mean?

Well the moment when I click on the run button, my code that I've written here in the Dart language will be compiled to a format that the machine can understand,so something closer to the ones and zeros. And in that momentm it has to be able to work out the value that should be held inside the constant.

But for a final, this doesn't have to be the case.

It can be worked out at any point down the line.

Let me demonstrate this. Our constant could be for example, 2 + 5 \* 8.

This is perfectly valid as a const because it can be calculated the moment that I click run and the value of that right hand side will be set to my const. Now what we can't do though is at runtime or when our app is actually running on the phone, if I want to use something that is only available at that point, say a button that's on screen, and I want to try and get its dimensions then I can't do that with a const and I would have to use a final. So a const can't have access to anything at runtime,

so when your app is running. If there's something that's created after the code has been compiled, then it shouldn't be set to a const.

For example if I wanted to calculate the current time, then I wouldn't need to create a new object from the dateTime and then I could use the dateTime object and I could tap into something called now. And this should give me the current date and time. But it can't be assigned to a constant because this has to be worked out after my code has run.

And then we can get access to the dateTime. But if I had instead moved this into my final, then you'll see that this is perfectly valid code, other than the fact that it's not an integer that comes out.

If I print now myFinal, then you can see that it calculates the date and time after the code is compiled and it shows you what that time is.

If we head back to our code, then you can see that we can change our color to a final, but it can't be a const because the color comes from when we create a new reusable card and that could be created at any time not just at the time when the code is compiled, but it could be any time when our app is running. And it is in that moment when we get the value for color and so we can't actually use a const here.

It has to be a final.

Now similarly, it can't just be a simple variable because our reusable card is immutable.

So whenever we create a new reusable card, then it will create an immutable stateless widget.

And whenever the reusable card needs to change say its color or its size, then that reusable card, that specific one, actually gets destroyed. And a new one takes its place. So its properties can't be mutable.

It can't change.

Which is why we need to declare it as final. Now usually when we're writing our code, numbers and hardcoded things in our design are usually represented by a constant.

So let's say that if we were to create the bottom part of our design which is simply just a container that has a fixed height, then we might do that there at the very bottom of our column.

So just before the column ends probably right here and we can add our new container which is going to have a color of 0xFFEB1555. And this is a nice sort of pink color that we got from the design on the Dribbble side down here.

And then we're going to add a margin.

And in this case we only actually want the margin to be there for the top of this container. And all the other parts of the container, the side and the bottom,we actually want to have no margin so that it very much sticks to all sides.

So instead of saying EdgeInsets.all, we're going to say EdgeInsets.only.

And the one that we're going to set is the top which we're going to set to 10. And then we want to set the width of our container.

And in my case we want to have this container stretch out all the way across the width of the screen,no matter which screen is on.

So it could be on a small screen an iPhone 4, or a large iPad, doesn't matter.

We want it to stretch.

So the easiest way of doing this is simply writing double.infinity.

And this will give us a value that is going to be equal to the full width of the screen.

So now all we need to do is to simply give our container a fixed height of 80 pixels.

And if we hit save, we should be up to see it right here,and this is what it looks like, which is pretty neat. Now here where we have our height as a fixed 80 pixels,we might at a later stage decide that that's too high.

Maybe we want to make it smaller or maybe you want to make it taller, and it's quite hard digging through the actual code to be able to change this.

So very often, you see constants being used in this case where they're defined at the very top of a file.

And let's call it bottomContainerHeight and we're going to set it to 80.0.

And now inside our container, we can change the height to that constant bottomContainerHeight.

And now we don't have to dig through the code to try and find out where it is and try and fix all the places where that exists.

Now we can simply look at the top and look at our constants and change them if we need to.

So this can easily be a constant because we can work it out at the point when we click run.

So it's 80 pixels,even if it was 80\* 2, that's all things that we can work out at the time when we actually run our app.

But what we can't make a constant is something that requires our app to be currently running.

So we can't for example decide that we want to create a constant that's based on our color.

So for example, what if we want to know the string value of our color and we want to create a constant called colorString and we set it to equal the value of our colour property .toString which would normally create a string value from that color?

But this is completely illegal because const variables must be initialized with a constant value. It can't be something that has to be worked out at a later stage or when color actually receives a value.

So this is illegal.

Now it's your turn.

And the challenge is we have all of these colors which are created using the raw hex codes. And they're kind of inside our widgets in many places cluttering up our code.

So can you create a constant to collect them together at the very top?

And if we wanted to change the color of our reusable cards, then we only need to do it in one place instead of six.

So pause the video and try to complete this challenge.

All right.

So this is as easy as creating our bottomContainerHeight .

We're going to create it at the very top of the file and we're going to call this whatever you wanted to call, but I'm going to call it activeCardColour and we're going to set that to equal the color that we have over here.

So we're gonna copy that and paste it here. And then now every time we use that color, we can simply use the activeCardColor in its place.

And this again is something that can be worked out at the moment when we press the run button.

Now we have our active card color and you might want to go a step further and move our bottom container color up to the top as well.

So let's cut that and let's add another constant code bottomContainerColour, and this is going to be equal to that pinkish color up here. And now we can replace it down here with bottomContainerColour.

So now our code looks a lot cleaner and everything that is hardcoded now refers to a constant that lives at the top the screen which is very easily found and easily changed. And we know that all the changes that we make here will get reflected in all the places where they're used.

So this is a really common use for constants. Now in the coming lessons,we're going to be using constants and final in a lot of places.

So you'll have plenty of practice and plenty of exposure.

So if it doesn't make complete sense straight away, don't worry we're going to come back to it and we're going to revisit it in the coming lessons. In the next lesson though,we're going to add a child to our reusable card and make it even more customized with some custom content that's contained inside the code.

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