Now that we've learned all about the provider package, it's time to actually put it into action in our　app so that we can read the tasks that are stored,

we can add new tasks,　we can check tasks off and we can delete tasks when the user desires it.

So in this first part of putting provider into practice, we're going to tackle the first goal which is　to be able to get the tasks and to be able to use it both on the task screen as well as the task list.

Essentially we're going to keep the state of our task provider high so that we can use it all the way　down the tree and we're going to be providing that data both to the TaskScreen as well as the TasksList and they're going to listen for changes in the tasks data. On the task screen　the reason why we need it is because we have this little part where it says 12 tasks.

That number has to be dynamic and reflect all the tasks that are currently inside　our To Do list.

And　for the task list　well that's pretty self-explanatory.

We need the task data available there so that we can actually render this list of tasks.

Now in the coming lessons where we're implementing provider, I'm going to explain exactly what behavior　we need first upfront and then you have the choice of either completing it as a challenge or simply　following along with me.

But I recommend that you really give it a go yourself　and even if you get stuck and come back, you will learn a lot more than simply just following along.

So in this case, what we need to be to do is to use our change notifier provider to provide our tasks data　right at the top of our app at the very root of our widget tree.

And then we can listen for it inside the task screen to update that number as well as inside our tasks　list to update the list of task tiles.

So once you already, open up your current Todoey app project and try and achieve this goal.

So I hope you've given that a go and you're here just to check the solution.

Now remember there's a lot of ways of doing the same thing. You can achieve exactly the same behavior　using all sorts of different ways　and my solution is just one way of doing that.

So the first thing we need to do is to of course go into our pubspec.yaml and to add our provider　as a dependency.

I'm gonna do that right here and then I'm gonna get all my provider related files and now I can use　it.

So the first thing that I'm gonna do is I'm going to create a new Dart file inside my models folder　and I'm going to call this task\_data.

So this is going to be the object that's going to be provided throughout my tree so that they can tap　into the data of our tasks.

This class is simply going to be called TaskData and it's again going to extend our change notifier　class.

Now the reason why it's erroring out at the moment is because we need to be able to import the foundation.dart package or we could import the material.dart package.

But the material.dart package actually contain the foundation.dart package　so if you wanted to keep it light then you can simply just use foundation.dart.

And now we will be able to find the change notifier inside foundation.dart and we can use it to extend　our task data class so that we can turn this class into something that other objects can listen to for　changes and update when we tell it to.

So the first thing I'm gonna do is I'm going to import my other models class which is my task.dart　and that just a quick reminder is basically the structure of a task object. And now if I go into my tasks　screen.dart, I can find where currently my task data exists. It's inside this list of task inside my　TasksScreenState.

So I'm going to lift that right out, so I'm going to cut it out of here, and I'm going to move it over　to my task data object.

So it's going to be pasted inside here.

It's going to be created exactly the same way. Tasks have a name　and by default they're set to 'not done'. So　now I'm ready to provide this task property across my app. And I'm going to do that at the very top of　my widget tree.

So I'm going to do it inside the MyApp class and I'm going to wrap everything that's inside my material　app with this change notifier provider class. And of course to be able to use it　I need to import my provider.dart package.

So remember that provider has to be at a place in the tree that's high enough to be seen by the rest　of the widgets that branch off later on.

So in our case, that means putting it right at the top of MyApp class.

Now if you only needed the data say inside the task tile and nowhere else, then you might be able to　go up one level to its parent and then wrap its parent which is this task list which of course lives　right here,　then you could wrap that with a provider widget to be able to provide to all of its children.

But if you get this wrong and you put this too low in the tree then the children might not have access　to it then you'll get errors when you run the app and try to do things telling you that it's not placed　at the correct position in the widget tree.

So take care where this goes.

In our case it needs to be above our material app.

Now I'm getting a little warning here because there's a parameter called builder for the change notifier　provider that's needed.

So we saw this before. The builder takes the current context,so the current position inside the widget tree, and it returns the object that needs be provided to all　of the children in the tree. And that object is of course on newly created task data object.

So let's go ahead and provide the task data object and also tell it about this task data object which　comes from the models folder and it's called task\_data.dart.

So now we have our change notifier provider at the very top of our widget tree providing the task data　object to all children below the tree that wants to listen for the data in there. Now we can actually read　the data from here namely the tasks list.

So the first place where we might want to do that is actually on the task screen. Here where we have　all tasks.length which shows the user how many tasks they have remaining in their list,　well this needs to be based on our task list.

We need to count how many there are in here and provide it as a interpolated string right here.

And we're getting an error right now because we removed tasks from our task screen state.

Let's go ahead and remove tasks.length and let's use what we learned previously where we tap into　to provider and we call the .of method.

Now I'm going to get Android Studio to help me out with typing out the code.

But first that means I have to import the provider.dart.

And now I can tap into the .of method of provider　and of course always specify the date type in the of method and it's going to be my task data object　which again we need to import.

So we're going to tap into the models and we're going to tap into the task\_data.dart.

Eventually we're going to remove the need for the task.dart and to manage all of that related stuff　inside task data.

But for now let's go ahead and tap in to this object which is going to be equivalent to our task data　object now and let's go into it to get hold of the tasks property. So let's write a .tasks and then　we're going to get the count or the length of that object which currently should be equal to 3.

So let's go ahead and hit save and try to get rid of the part where we have a Dart analysis error.

So the first one is this part where we're adding tasks to our previous task list which now doesn't exist　so of course it's undefined.

So I'm actually going to go ahead and just comment out our set state and in the next lesson when we tackle　how to add new tasks to our tasks data provider, then we'll figure this part out.

So the other part is of course down here where we have our tasks being passed to our task list.

Now this is of course the second place where we need to read from our task data

.task property but instead of passing it over and drilling through our tree, instead I'm simply going　to go into our task list and get a hold of a reference to that task data object over here instead.

So we actually don't need to pass anything over anymore so we don't need these two things and we don't　need this reference to our task.dart either.

But instead I'm going to be importing our provider.dart and also our model's folder　the task\_data.dart as well. And we're going to be using it inside our task list to build our list　view.

Now right here instead of using the widget.tasks which taps into those properties passed into the　task list stateful widget, instead I'm simply just going to tap into the provider.of method. So　instead of the widget.tasks.[index].name, I'm going to replace widget.tasks with provider.of and specifying the data type which is our task data. And then it's going to be provider.of.tasks[index].name. So I can copy this over and replace our widget in all of these　places.

And for now I'm simply actually going to comment out our set state to handle what should happen　when our checkbox gets checked.

We're going to deal with that in a later lesson.

But right now all we want is just to be able to read the data that's inside the tasks inside our tasks　data which is being provided using our provider package.

So now notice how we're not using that widget anymore which comes from the stateful widget.

So that's our way of accessing the properties that we deleted.

And so we actually don't need this to be stateful　even.

We can actually turn it into a simpler stateless widget and we can still get hold up with the　provider and get the data that we need because we're no longer passing it over from our task screen　via the constructor.

So now notice how we also don't need states inside our task screen so we can convert this to a state　less widget as well by deleting all of these lines and changing this to a stateless widget and we can　delete the unused imort right up here as well.

Now our app is actually a lot simpler because we have these memoryless stateless widgets which simply　get constructed when they're needed and destroyed and replaced with new ones when they update.

So now we're tapping into the provider of our task data and getting hold of that list right here in order　to populate our task titles　and also we're using it inside our task screen to show how many tasks we have remaining.

And this is just simply reading the tasks.

Let's go ahead and hit save and run our app from scratch. And you can see that hopefully everything going　well and all your code in the right places　we should have exactly the same user interface as we had before. But this time this data is no longer　being passed around,　drilled down through the tree and instead it's coming from our provider.

So did you manage to get to this point when you tried to tackle it yourself?

Well there's a couple of things that we did including creating our task data class that extends change　notifier and wrapping our material app within a change notifier provider which provides that task data object　and then finally using it by using the provider.of task data to tap into it both inside our task　screen as well as our task list. That works right now but we can actually make it a lot simpler. And we　can do this through the use of something called a consumer widget that comes from our provider.dart　as well. Because notice how we're using our provider.of task data.tasks in three places here.

It's getting a little bit repetitive　right?

So instead of calling provider.of in three places, we can simply just wrap all of the downstream widgets　that need to be updated when this tasks property changes inside what's called a consumer widget.

So notice how we're using our provider.of inside our tasks tile but we're also using it inside the　item count.

So if we go one level higher than that well then that's going to be our list view widget.

So let's go ahead and wrap our list view widget inside a new widget which is going to be our consumer　widget.

And this of course comes from the provider package.

And in order to use it we simply wrap any downstream widgets that need to be updated when a particular　piece of data that we're listening to changes and then we specify the data type which is also going　to be our task data.

That's what we're listening for.

And then it simply requires a builder and the builder takes the current context,

so where we are in the tree,　in addition it will be able to provide the current data and we can give that object a name.

So that is going to be our task data object.

So I'm just simply going to call it taskData and that is going to be equivalent or basically the same　as calling provider.of task data context.

And finally it also takes a property called a child.

And now our builder is constructed and we can simply return any widgets that need to be built using　this data. Instead of having a child property　I'm going to take this entire list view builder all the way down to the closing list view builder comment,　I'm going to cut it and then make sure I return it inside my consumer. Of course because we're returning　this widget we have to add a semicolon here instead of a comma. So now we've gotten rid of all our errors　and instead of using provider.of blah blah blah we can replace all of that with just our taskData.

Let's go ahead and replace taskData in all of these places where we have provider.of　and now what happens is that this consumer widget is going to be the one that's listening for changes　in task data.

So when task data updates its state then it's going to rebuild this entire list view and update all　the task titles that have changed.

So now our code is a lot more succinct and we can even go one step further because notice how right　here we're using our taskData.tasks.length　so we're getting the length of our list of tasks here,　but we're also doing that right here　right?

So usually when we're doing the same thing over and over again then we should probably set up something　to make it a little bit easier a little bit more convenient　the next time.

So if we go into our task data we can actually create a new integer which has a getter.

So we use the get to keyword to define this being a getter and the getter is going to be something that　returns a value.

So this means that whenever somebody taps into the name of this property, this taskCount, then we're　going to return a value that we calculate. And we're going to return this value based on our task list.

So we're going to say tasks.length　and now what happens is every time we try to tap into this task count property, well it's going to compute　the value of it by looking at our tasks which is this list and getting the length of that list.

So now we can go into our task screen and instead of calling provider of task.tasks.length,　we can simply just simplify that to the task count and we can of course do that inside our task list　as well　so it will be taskData.taskCount.

So we've now dramatically refactored our code both through the use of a consumer widget which can save　us a lot of provider.ofs.

And also we've managed to create a getter inside our task data to simplify the process of getting the　length of our tasks list.

Now of course you didn't need to do all of this to complete this challenge.

All I wanted was to see that you could use the provider package to get hold of the data that were providing　through the use of a new object that uses and extends the change notifier as well as wrapping our　entire tree inside a change notifier provider to be able to provide this task data object both inside　the task screen either using providr.of or, as we've seen now, we can also use the consumer widget as　well.

So that completes reading and exposing our provider.

But in the next lesson, we're going to look at how we can add new tasks into our provider through the　use of our change notifier.

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