Now in previous lessons we talked about lifting state up and the reason why we needed to lift state　up　in that case was because here we have a child widget, a task checkbox, which is lower down in the tree　than the task tile.

And we wanted to be able to use the user interaction that happened in the taskCheckbox to determine　and set the state in the parent widget so that we could use that state to both update the checkbox as　well as the text widget which lived higher up in the tree. By lifting the state up in the widget tree　it's able to be passed downstream to many more child widgets.

So now we're in another situation where we actually need the state to be higher up in the tree.

Let me illustrate the problem.

So when we pressed the 'Add' button and the 'Add Task' screen pops up,　well if we think about how our widget tree looks at the moment, this is kind of a simplified version.

We have our TasksScreen at the top and then that in turn contains a AddTaskScreen screen and a TaskList and　the AddTaskScreen has a text field.

The TaskList has a whole bunch of task tiles and some text widgets.

Now the task list is built using the state of something called tasks.

So that's a list of task objects.

And using the list view builder, we're able to render a dynamic list of task titles.

But imagine if the user types in a new task to the AddTaskScreen and then they press the Add button,　how do we get the data inside the text field all the way up and all the way down to the text widgets?

Well we would need to increase the scope of the tasks list. We again need to lift the state up. We need　to be able to bring the tasks list higher up so that we can add to it and use it across the child widgets　so that we can render the latest item that's added inside our list view.

Now it's time for the challenge. Using what you've learned about lifting state up and callbacks as well　as all of the stuff that you've learned before,　can you make the to do list app incorporate the data that was typed into the text field and when the　user presses the Add button　the latest to do item gets added into the list and we retain the capability of being able to check it　off while keeping our　task title as a simple stateless widget?

So in order to achieve this, you'll have to think about what you've learned about lifting state up, callbacks　and use everything you know to tackle this problem.

So give it a good go and pause the video to try and complete this challenge. So I hope you gave that　a try.

I know it's one of the harder challenges but I think you can do it. If you haven't tried, again　pause the video and make sure you've given it a go before you continue　because I'm now going to reveal the solution or at least one of the solutions because as always there's　many ways of doing the same thing.

So the first thing I want to do is currently I have my tasks which contains the names of the tasks and　whether if the tasks are done inside my task list stateful widget,　so it's at this level of the tree. But I need to lift the state of this task list up to be able to use　it both inside the AddTaskScreen as well as other places inside my task screen, for example the number　of tasks that are remaining or the task list which needs to render the names of the tasks.

So I'm going to go into my task list and I'm going to cut my tasks and I'm going to move it up the tree　into the task screen,　so right here.

And as soon as I do that because this is a modifiable property, I get a warning　and that means I have to switch to a stateful widget.

And we also don't know anything about tasks over here　so we have to import my models folder and my tasks.dart. So now we have our list of tasks inside our　task screen widget.

It's now accessible downstream such as for the Add task screen as well as for the task list.

So let's pass it down to the task list first.

So when I create a new task list inside my container over here, I want to be able to pass over those　tasks like so.

So to be able to do this, my task list will need a property called tasks which is going to be a list　of task objects.

So let's create it as tasks,　make sure there's an 's' at the end.

And then when we construct our task list, we're going to make sure that the tasks gets passed over and　initializes as this property.

So now inside the task list state, we can tap into this item by using the widget object. So we can use　widget.tasks and we can add that to all the places we currently have a red squiggly line.

So it passes over the object into the state.

So widgets refers to the stateful widget which of course gets that property　when we create a new task list. And now over here our errors go away because this task list now knows　how to construct itself using this property.

So now when we go ahead and runner up again　you'll notice that pretty much nothing has changed and our tasks are now coming from a higher point　in our widget tree, namely the task screen.

Now we're ready to go ahead and make the AddTasksScreen work.

So if we go into here, then we can first make sure that our text field is able to pass the data over　into the onPressed,so remember the data lives in different places. Our text field as here but we only want to pass that　data when the user presses the Add button. Our text field of course has a property called onChanged　and when it gets changed, it takes a callback to pass over the new text into a callback.

And when that happens we're going to save that string inside the new task title and we're going to set　it inside this callback.

So the new task title is gonna be equal to the new text that's added into the text box right here.

And when the onPressed gets pressed, let's first try printing the value of the new task title.

Now remember that the onChanged gets triggered every time the text field gets changed.

So now if I go ahead and add a new task and remember that if you want the keyboard to pop up on iOS anyways　it's just a COMMAND + K to do that.

So let's add a new task. Let's say, I don't know,

'Go to the gym' and I press　Add, then you'll see that string being printed down here because it's being passed over through this　new text and that new text gets assigned to the new task title which is right here and then it gets　printed when I press the flat button.

Now of course this new text actually updates every time the text field is changed. So it's not going　to just update when you're done with writing because it doesn't know when you're done with typing.

So for example if I go ahead and print the new text right here, then you can see that every time I add　a new letter it's going to call that print again.

Again coming back to this idea, we don't know when the user's done.

We don't know if they're just having a moment to pause to think, 'Go to the gym and eat something.'

This is what I do.

Uhm no, just kidding.

So we don't know when they're done.

So we have to wait until they actually signal that by pressing the Add button.

So we're going to just keep track of everything they type　and only in the moment where they press the add button do we take that as the final task title and we　do something with it. Now that we know how we can get hold of this new task that they typed into the　Add task screen,　well how do we actually go ahead and take it back to the task screen to add it into our list of tasks?

Well it's same as before right?

We need a callback.

So let's go ahead and create our callback right here.

Let's create it so that the add task screen actually takes the callback, so our callback is going to be

a set of parentheses and then some curly braces.

And then inside here we're going to try and print whatever it is that the user typed in.

So let's put that as the newTaskTitle and that we're going to try and print the new task title　but inside our task screen instead of our add task screen to demonstrate that we've actually managed　to get access to it in this top level widget.

So now when we construct our add task screen, well it has to be able to take a function　right?

So our stateless widget is going to take a final function called addTaskCallback and then it's going　to be added when we construct our add task screen,　so this.addTaskCallback,　and now if I hit save and we go back to our our add task screen then you can see the error goes away. So　now we have this call back which we passed over to the add task screen and it gets put into this property　called add task callback,　we now have to use it within our onPressed. In here instead of printing the value of the new task title,　I'm simply going to call that callback which is the add task callback and then I'm going to pass in the　new task title right here.

Now this should be passed back to the task screen and I'm going to see if I'm able to print it right　here.

So let's run our app again and go ahead and click the Add button and then I'm going to add 'Go to the　gym' and then press add and you can see it gets printed.

So I now have access to whatever the user typed in inside my task screen through the use of this callback.

So over here I can instead of printing it, I can actually just add it to my list of tasks　right?

I can tap into tasks.add and I'm going to add a new task item which is going to have a name based　on the new task title.

And then of course the isDone property is　going to be set to false by default.

So now I should have a new item in my task list and if I wrap that inside a set state, then I should　be able to see my task list update as soon as I hit the Add button.

So now if I go ahead and hit save and I press the Add button, then if we dismiss this bottom sheet then　you can see that the latest task has now been added into my to do list.

But we don't want to have to manually dismiss the bottom sheet after every time　right?

The natural thing should be when we press the Add button, it also dismisses the bottom sheet revealing　the latest item we've added. In order to do that we have to add a call to the navigator to pop off the　current context which is going to be the bottom sheet.

So now let's go ahead and run our app from scratch and let's add a new item　so 'Go to the gym' and then go ahead and press add and you'll see the bottom sheet disappears　and our latest task gets added to the to do list.

So my solution might be different from yours but as long as you've managed to lift the state of our　tasks object up to the task screen state so that it can be used both inside the callback to the add task screen　to get hold of the task that gets created inside the text field and also use it inside the task list　to populate it with the latest tasks when it gets updated, then you've pretty much achieved success in　this challenge.

Now there's one last thing that we could add here which also uses the tasks list which is this hard　coded thing here where it tells us how many tasks are in our task list.

So instead of writing 12 every single time, we can go ahead and add some string interpolation to　tap in to the task list and tap into the length property to get the actual number of tasks showing up　inside here. So how is that? Were you able to complete the challenge by lifting the state up and using　our callbacks?

Did it make sense? If it didn't then be sure to review the lesson on callbacks and try using some of　the things you've learned about it to practice it inside this app.

Now in the next lesson, we're going to take state management a little bit further and talk about something　called the provider package.

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