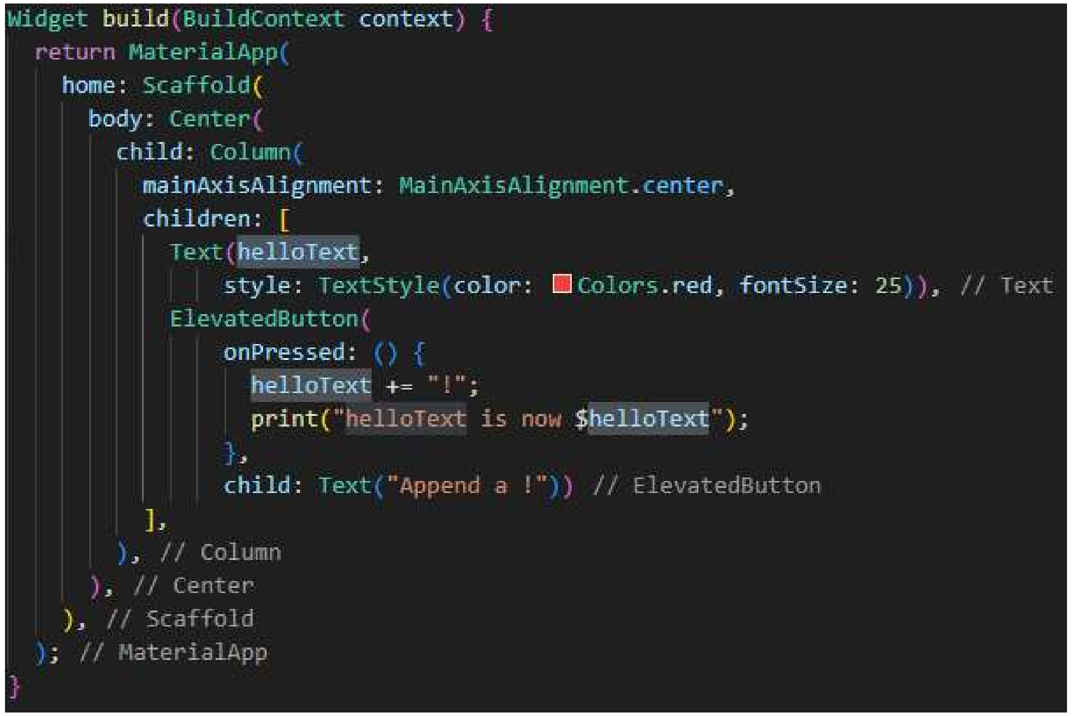
Stateful widgets and dynamic UI elements

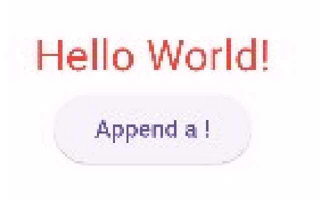
- Try to change texts in your app when user taps a button.
- Learn that stateless widgets are not updated.
- Add a StatefulWidget to your app and learn to use setState.
- Use Sliders, TextFields, Checkboxes and Switches in your app.
- Use ListTiles and CheckboxListTiles as part of your app.
- Learn how to define an AppBar and a Floating Action Button

Goal: make text ”Hello World!” in our app dynamic

Idea: Change the text when user taps a button:



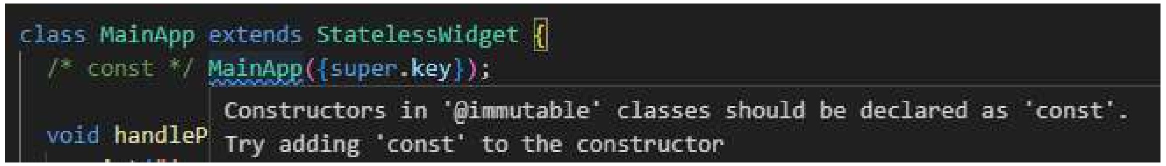
```
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(helloText,
                style: TextStyle(color: 🟥Colors.red, fontSize: 25)), // Text
            ElevatedButton(
                onPressed: () {
                  helloText += "!";
                  print("helloText is now $helloText");
                },
                child: Text("Append a !")) // ElevatedButton
          ],
        ), // Column
      ), // Center
    ), // Scaffold
  ); // MaterialApp
}
```



Where to define the variable helloText ?

Stateless Widgets need a const constructor

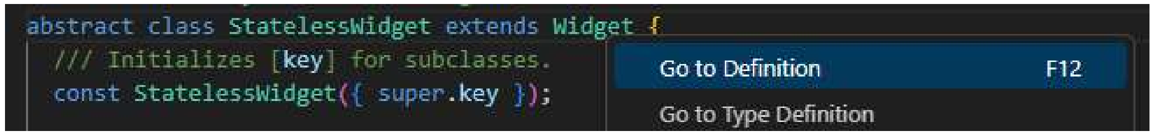
Our MainApp is derived from StatelessWidget:



```
class MainApp extends StatelessWidget {
  /* const */ MainApp({super.key});

  void handleP
```

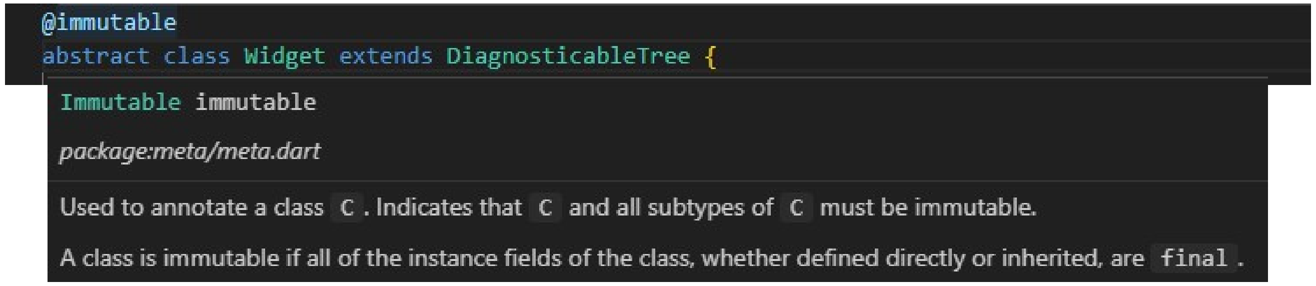
```
Constructors in '@immutable' classes should be declared as 'const'.
Try adding 'const' to the constructor
```

StatelessWidget is derived from Widget:



```
abstract class StatelessWidget extends Widget {
  /// Initializes [key] for subclasses.
  const StatelessWidget({ super.key });
```

Widget is declared as “immutable”, thus the derived MainApp is immutable too:



```
@immutable
abstract class Widget extends DiagnosticableTree {
```

```
Immutable immutable
```

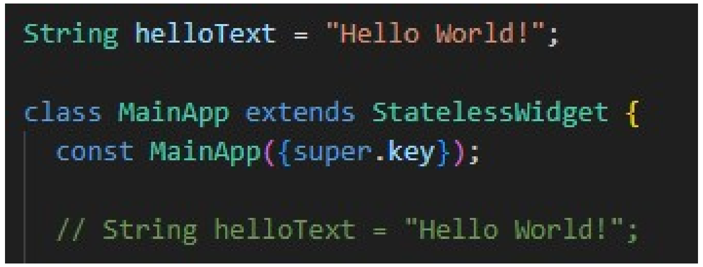
package:meta/meta.dart

Used to annotate a class `C`. Indicates that `C` and all subtypes of `C` must be immutable.

A class is immutable if all of the instance fields of the class, whether defined directly or inherited, are `final`.

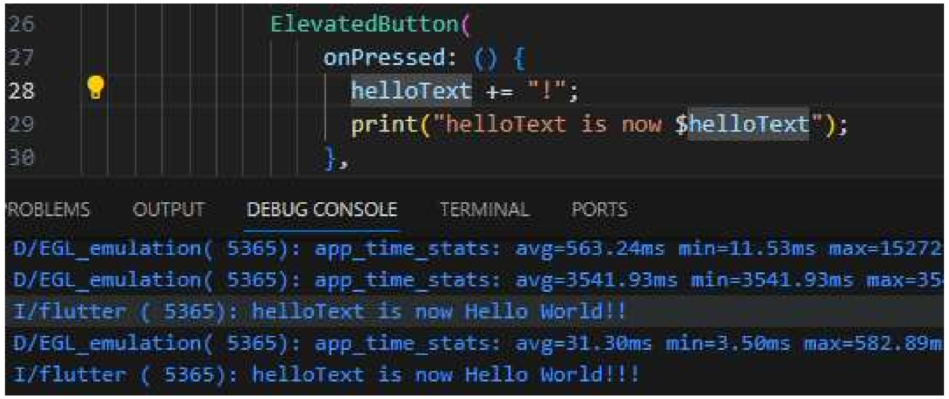
The UI of Stateless Widgets is not updated

Idea: we define helloText as a global variable:



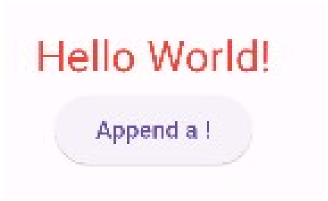
```
String helloText = "Hello World!";

class MainApp extends StatelessWidget {
  const MainApp({super.key});

  // String helloText = "Hello World!";
```

helloText is changed, but the UI is not updated:



```
                    ElevatedButton(
                      onPressed: () {
                        helloText += "!";
                        print("helloText is now $helloText");
                      },
```

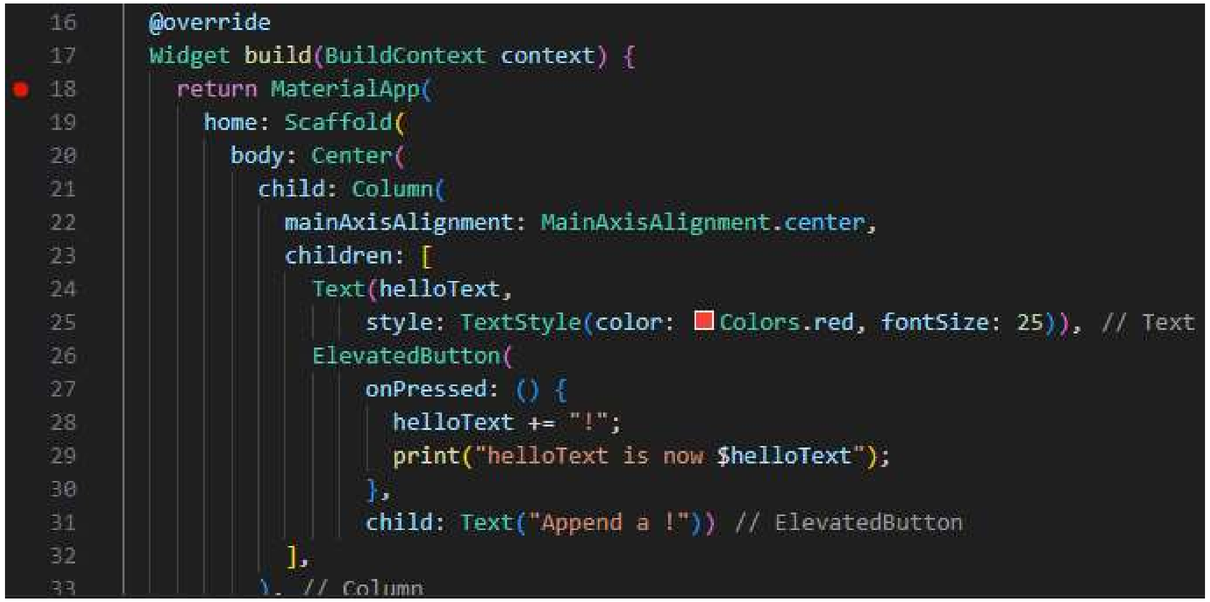
```
D/EGL_emulation( 5365): app_time_stats: avg=563.24ms min=11.53ms max=15272
D/EGL_emulation( 5365): app_time_stats: avg=3541.93ms min=3541.93ms max=35
I/flutter ( 5365): helloText is now Hello World!!
D/EGL_emulation( 5365): app_time_stats: avg=31.30ms min=3.50ms max=582.89m
I/flutter ( 5365): helloText is now Hello World!!!
```



Hello World!

Debugging build method of Stateless Widgets

Set breakpoint on build method:

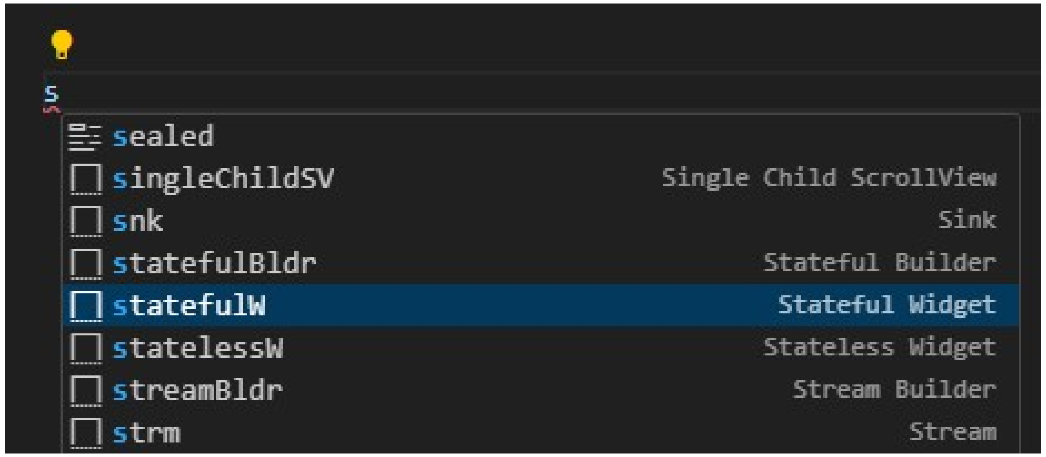


```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text(helloText,
                style: TextStyle(color: 🟥Colors.red, fontSize: 25)), // Text
            ElevatedButton(
                onPressed: () {
                  helloText += "!";
                  print("helloText is now $helloText");
                },
                child: Text("Append a !")) // ElevatedButton
          ],
        ), // Column
```

The build method of a StatelessWidget is **only called once** by the Flutter framework during program start !

Create a Stateful Widget

IntelliSense of VS Code helps us:

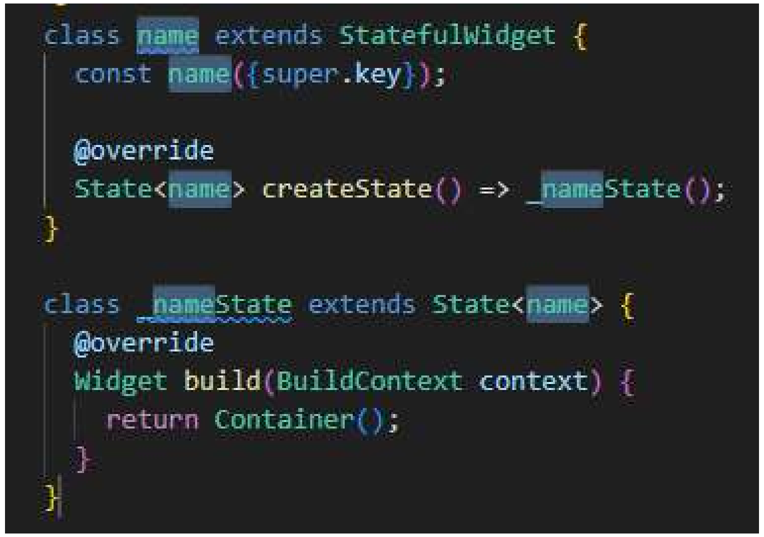


```
s
sealed
singleChildSV                  Single Child ScrollView
snk                                               Sink
statefulBldr                          Stateful Builder
statefulW                              Stateful Widget
statelessW                            Stateless Widget
streamBldr                              Stream Builder
strm                                            Stream
```

Enter “s” and select “Flutter Stateful Widget”.

Create a Stateful Widget

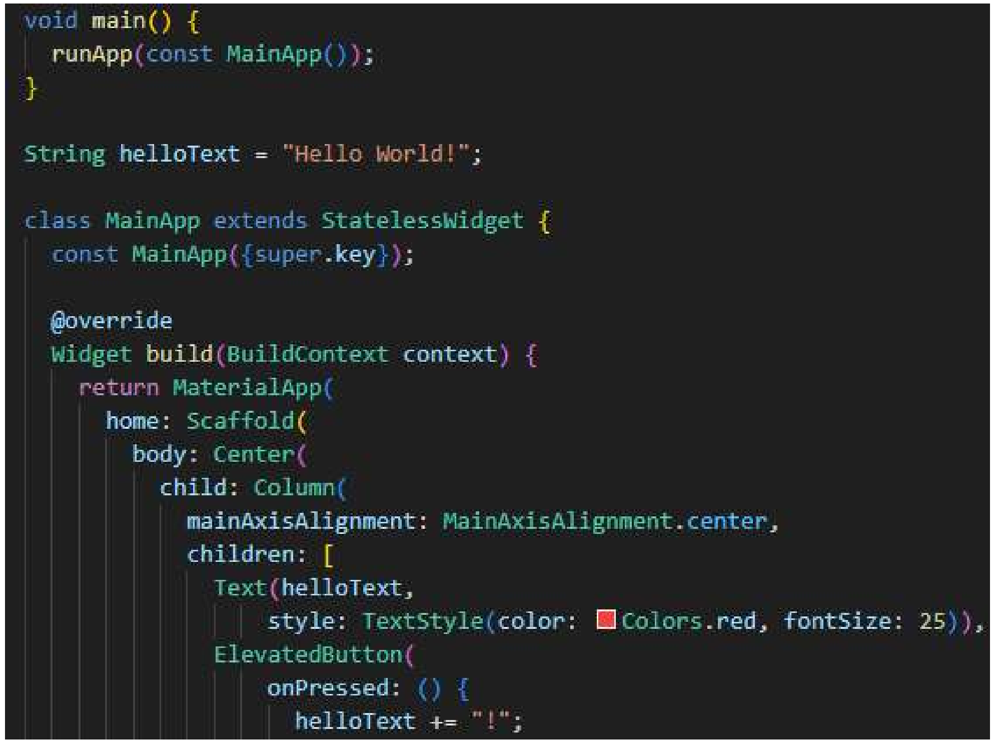
Stateful Widgets have a State, and this State has a build method:



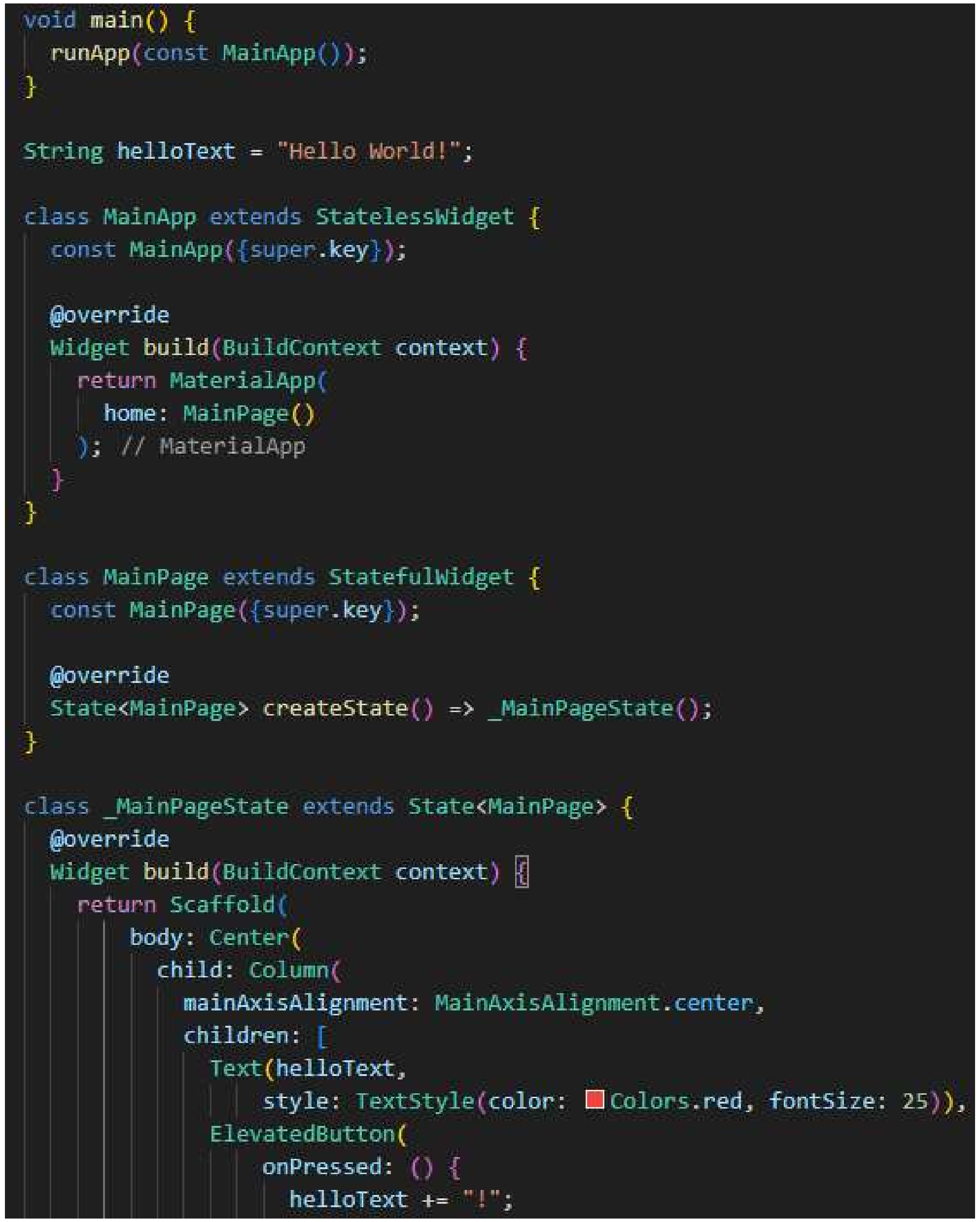
```
class name extends StatefulWidget {
  const name({super.key});

  @override
  State<name> createState() => _nameState();
}

class _nameState extends State<name> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

State is not immutable, it can have non final members (it can change “it’s state”).

Compare old and new code:

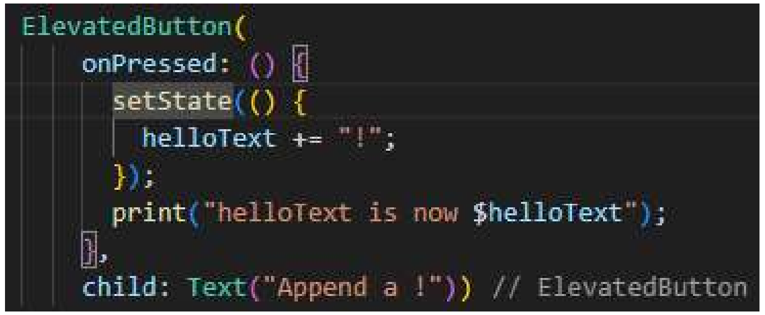


```dart
void main() {
  runApp(const MainApp());
}

String helloText = "Hello World!";

class MainApp extends StatelessWidget {
  const MainApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(helloText,
                  style: TextStyle(color: 🟥Colors.red, fontSize: 25)),
              ElevatedButton(
                  onPressed: () {
                    helloText += "!";
```

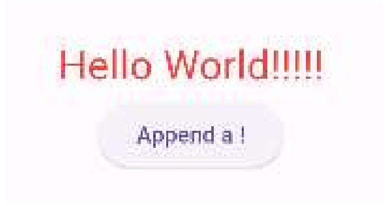


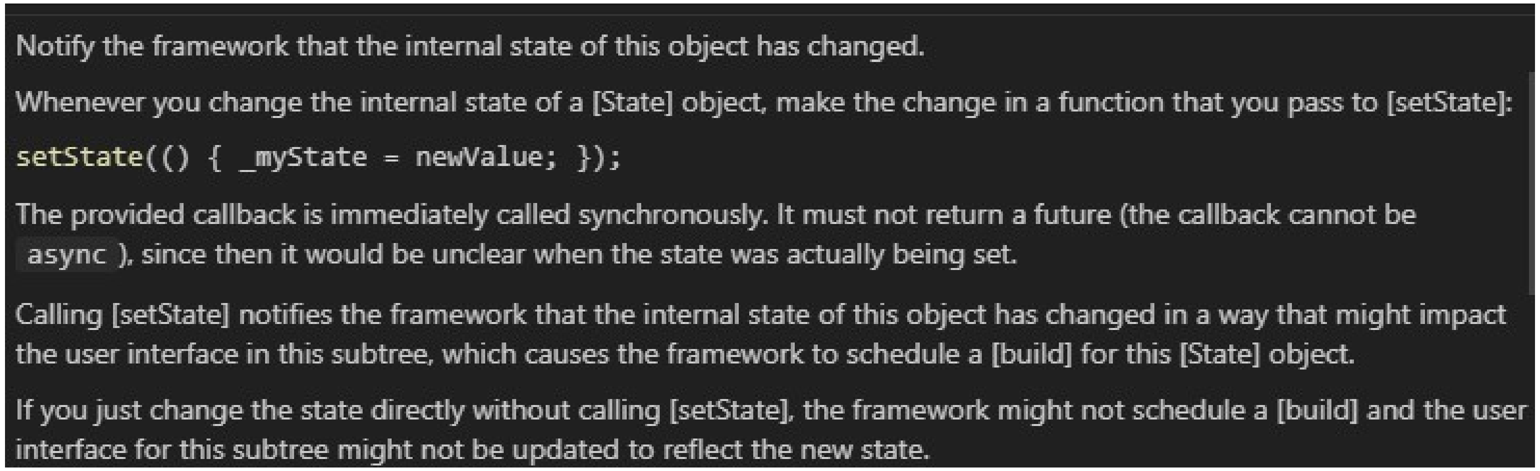
```dart
void main() {
  runApp(const MainApp());
}

String helloText = "Hello World!";

class MainApp extends StatelessWidget {
  const MainApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MainPage()
    ); // MaterialApp
  }
}

class MainPage extends StatefulWidget {
  const MainPage({super.key});

  @override
  State<MainPage> createState() => _MainPageState();
}

class _MainPageState extends State<MainPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(helloText,
                  style: TextStyle(color: 🟥Colors.red, fontSize: 25)),
              ElevatedButton(
                  onPressed: () {
                    helloText += "!";
```

“setState” triggers a redraw of the widget



```
ElevatedButton(
    onPressed: () {
      setState(() {
        helloText += "!";
      });
      print("helloText is now $helloText");
    },
    child: Text("Append a !")) // ElevatedButton
```





Notify the framework that the internal state of this object has changed.

Whenever you change the internal state of a [State] object, make the change in a function that you pass to [setState]:

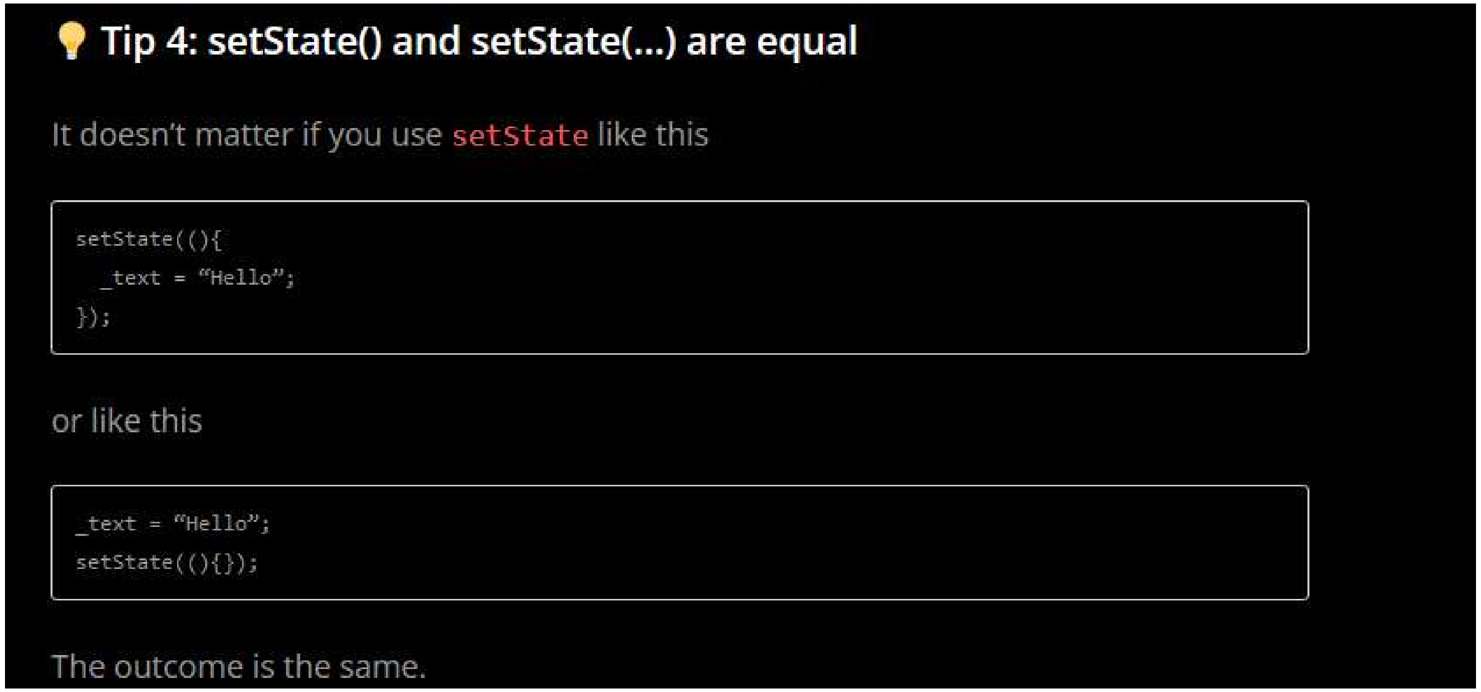
```
setState(() { _myState = newValue; });
```

The provided callback is immediately called synchronously. It must not return a future (the callback cannot be `async`), since then it would be unclear when the state was actually being set.

Calling [setState] notifies the framework that the internal state of this object has changed in a way that might impact the user interface in this subtree, which causes the framework to schedule a [build] for this [State] object.

If you just change the state directly without calling [setState], the framework might not schedule a [build] and the user interface for this subtree might not be updated to reflect the new state.

Alternatives how to call “setState”



💡 Tip 4: setState() and setState(…) are equal

It doesn’t matter if you use `setState` like this

```
setState((){
  _text = “Hello”;
});
```

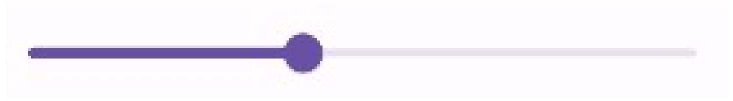
or like this

```
_text = “Hello”;
setState((){});
```

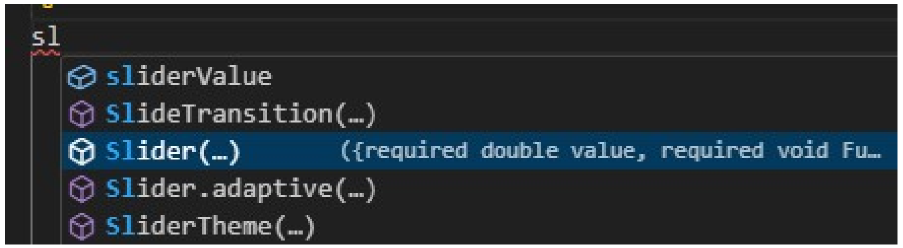
The outcome is the same.

Copied from https://quickcoder.org/flutter-set-state/

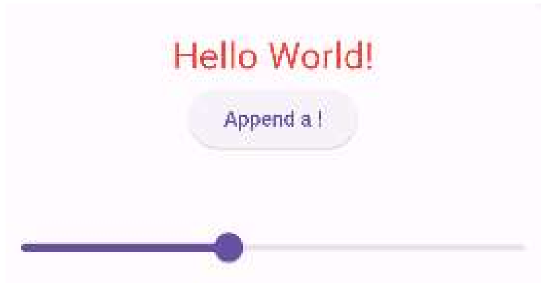
Define a Slider

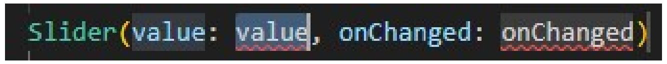


IntelliSense of VS Code helps us:

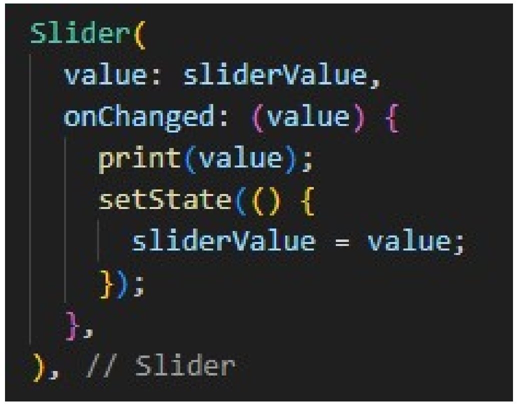


```
sl
  sliderValue
  SlideTransition(…)
  Slider(…)          ({required double value, required void Fu…
  Slider.adaptive(…)
  SliderTheme(…)
```



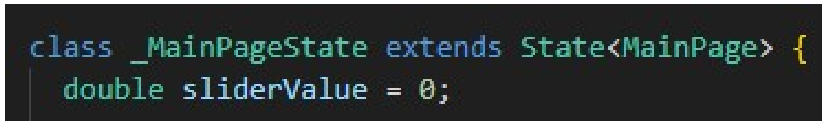


```
Slider(value: value, onChanged: onChanged)
```



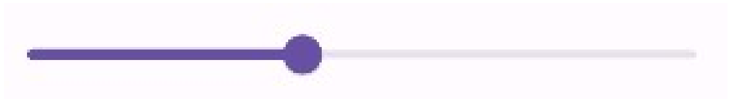
```
Slider(
  value: sliderValue,
  onChanged: (value) {
    print(value);
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

sliderValue can be a member of our State class, because the State class is not immutable.

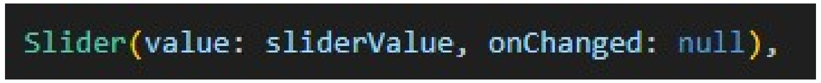


```
class _MainPageState extends State<MainPage> {
  double sliderValue = 0;
```

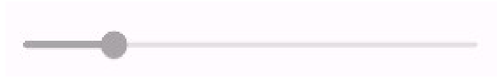
Define a Slider (continued)



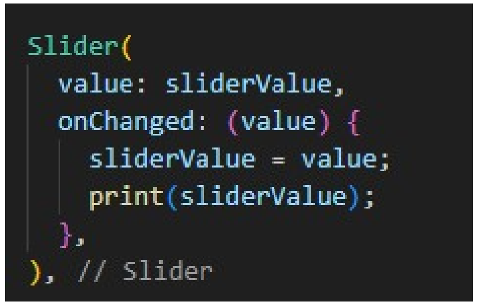
When onChanged is null, the slider is disabled:



```
Slider(value: sliderValue, onChanged: null),
```

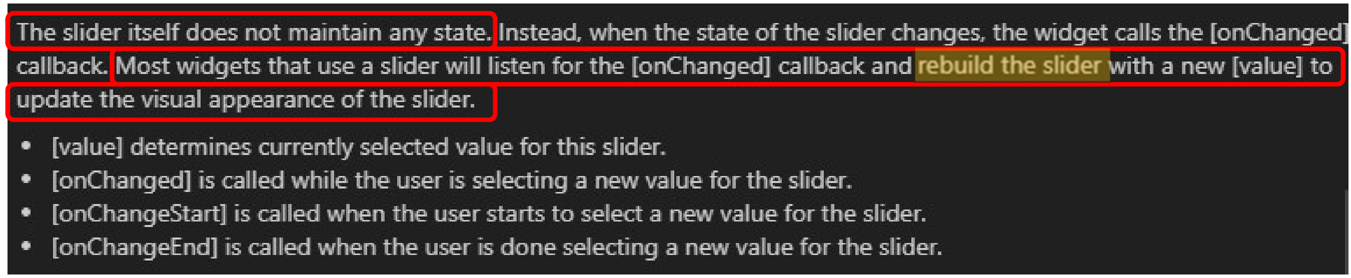


Without calling setState, the slider does not move:



```
Slider(
  value: sliderValue,
  onChanged: (value) {
    sliderValue = value;
    print(sliderValue);
  },
), // Slider
```

From Intellisense (scroll down there):

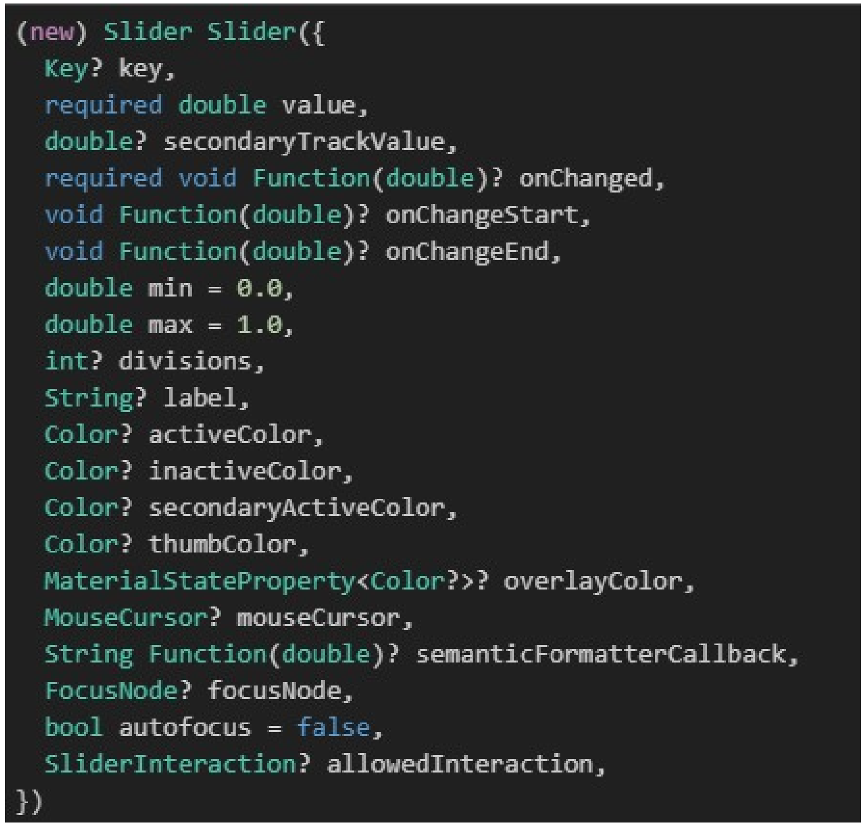


The slider itself does not maintain any state. Instead, when the state of the slider changes, the widget calls the [onChanged] callback. Most widgets that use a slider will listen for the [onChanged] callback and rebuild the slider with a new [value] to update the visual appearance of the slider.

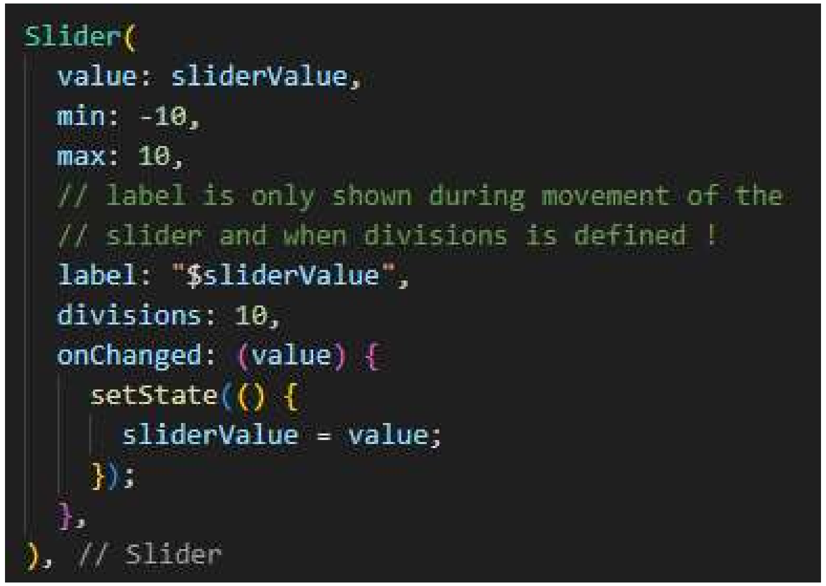
- [value] determines currently selected value for this slider.
- [onChanged] is called while the user is selecting a new value for the slider.
- [onChangeStart] is called when the user starts to select a new value for the slider.
- [onChangeEnd] is called when the user is done selecting a new value for the slider.

Slider with more properties

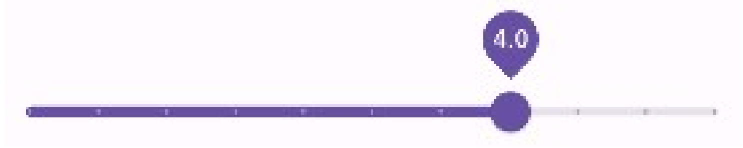
IntelliSense shows the c-tor parameters -> try some of them that seem to be interesting:



```
(new) Slider Slider({
  Key? key,
  required double value,
  double? secondaryTrackValue,
  required void Function(double)? onChanged,
  void Function(double)? onChangeStart,
  void Function(double)? onChangeEnd,
  double min = 0.0,
  double max = 1.0,
  int? divisions,
  String? label,
  Color? activeColor,
  Color? inactiveColor,
  Color? secondaryActiveColor,
  Color? thumbColor,
  MaterialStateProperty<Color?>? overlayColor,
  MouseCursor? mouseCursor,
  String Function(double)? semanticFormatterCallback,
  FocusNode? focusNode,
  bool autofocus = false,
  SliderInteraction? allowedInteraction,
})
```

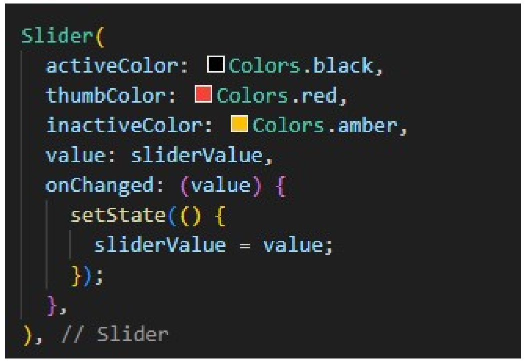


```
Slider(
  value: sliderValue,
  min: -10,
  max: 10,
  // label is only shown during movement of the
  // slider and when divisions is defined !
  label: "$sliderValue",
  divisions: 10,
  onChanged: (value) {
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

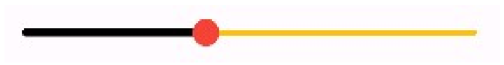


Test more slider properties

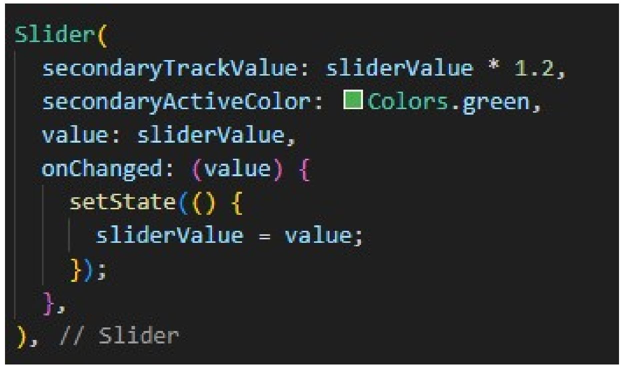
A “German flag slider” (but take care: too many colors might confuse the user):



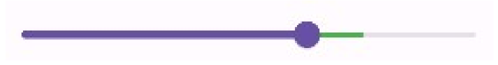
```
Slider(
  activeColor: Colors.black,
  thumbColor: Colors.red,
  inactiveColor: Colors.amber,
  value: sliderValue,
  onChanged: (value) {
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```

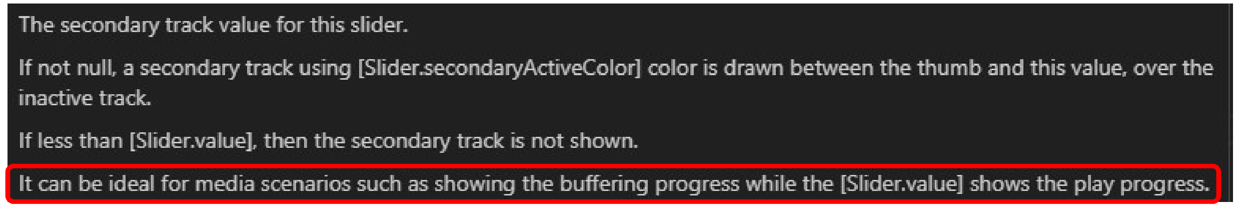


A slider with a secondary tag:



```
Slider(
  secondaryTrackValue: sliderValue * 1.2,
  secondaryActiveColor: Colors.green,
  value: sliderValue,
  onChanged: (value) {
    setState(() {
      sliderValue = value;
    });
  },
), // Slider
```





The secondary track value for this slider.

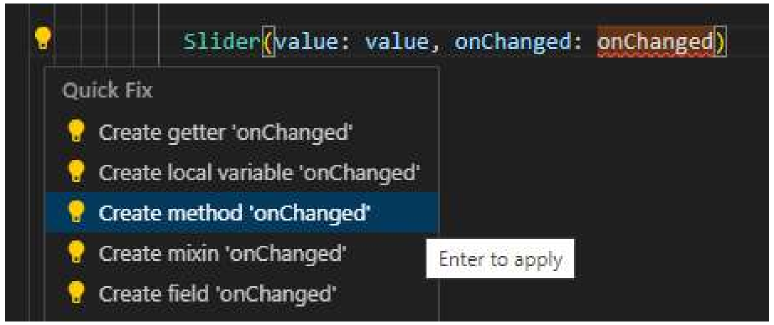
If not null, a secondary track using [Slider.secondaryActiveColor] color is drawn between the thumb and this value, over the inactive track.

If less than [Slider.value], then the secondary track is not shown.

It can be ideal for media scenarios such as showing the buffering progress while the [Slider.value] shows the play progress.

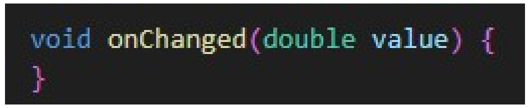
Alternative to anonymous onChanged method

Use Intellisense to create a method:



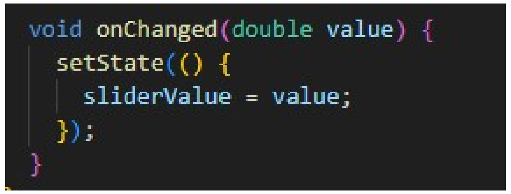
```
Slider(value: value, onChanged: onChanged)
```

Quick Fix
Create getter 'onChanged'
Create local variable 'onChanged'
Create method 'onChanged'
Create mixin 'onChanged'
Create field 'onChanged'
Enter to apply



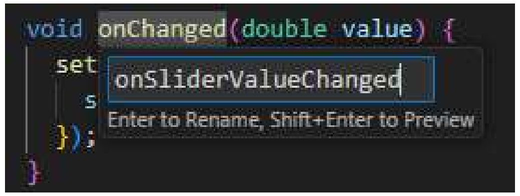
```
void onChanged(double value) {
}
```

Fill it:

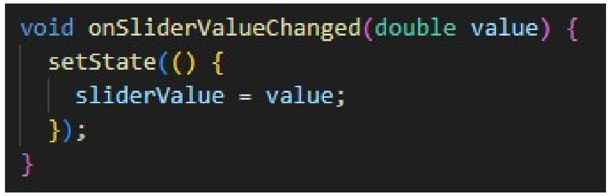


```
void onChanged(double value) {
  setState(() {
    sliderValue = value;
  });
}
```

and then rename it with F2:



```
void onChanged(double value) {
  set onSliderValueChanged
    s
  });
}
```

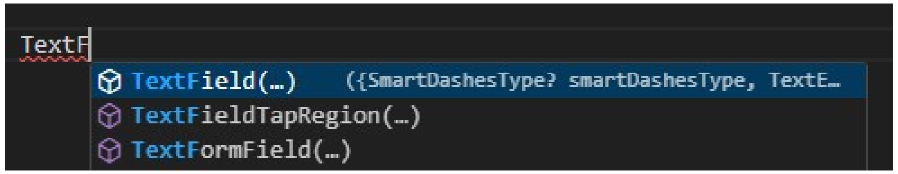


```
void onSliderValueChanged(double value) {
  setState(() {
    sliderValue = value;
  });
}
```

“**Golden rule**”: use an own method when quite a lot of code is needed in onChanged.

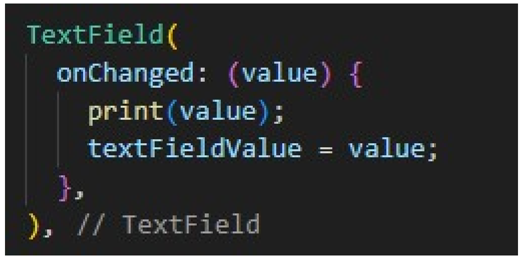
Define a TextField

Again you can start with IntelliSense of VS Code:

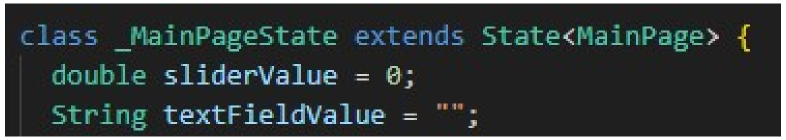


```
TextF
  TextField(…)      ({SmartDashesType? smartDashesType, TextE…
  TextFieldTapRegion(…)
  TextFormField(…)
```

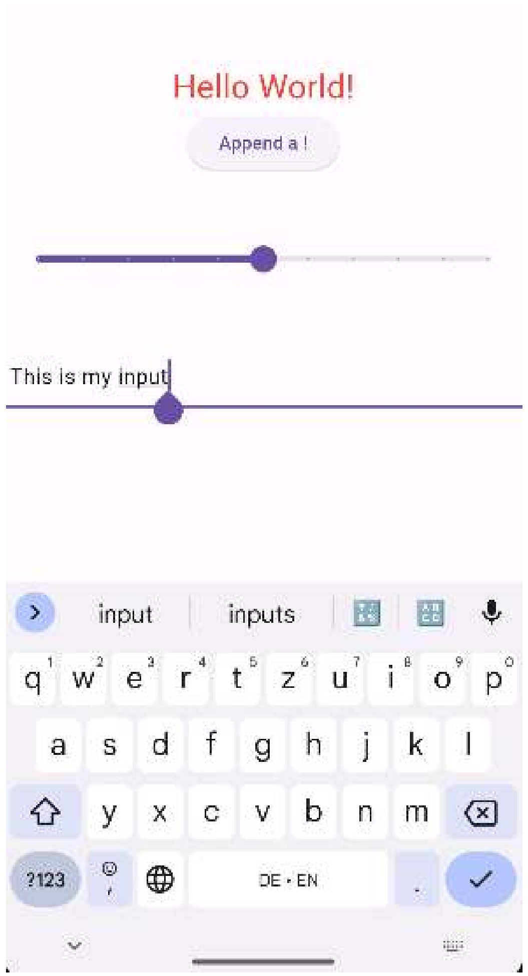
TextField has no required parameters, but we define an **onChanged** method to keep track of the entered text:



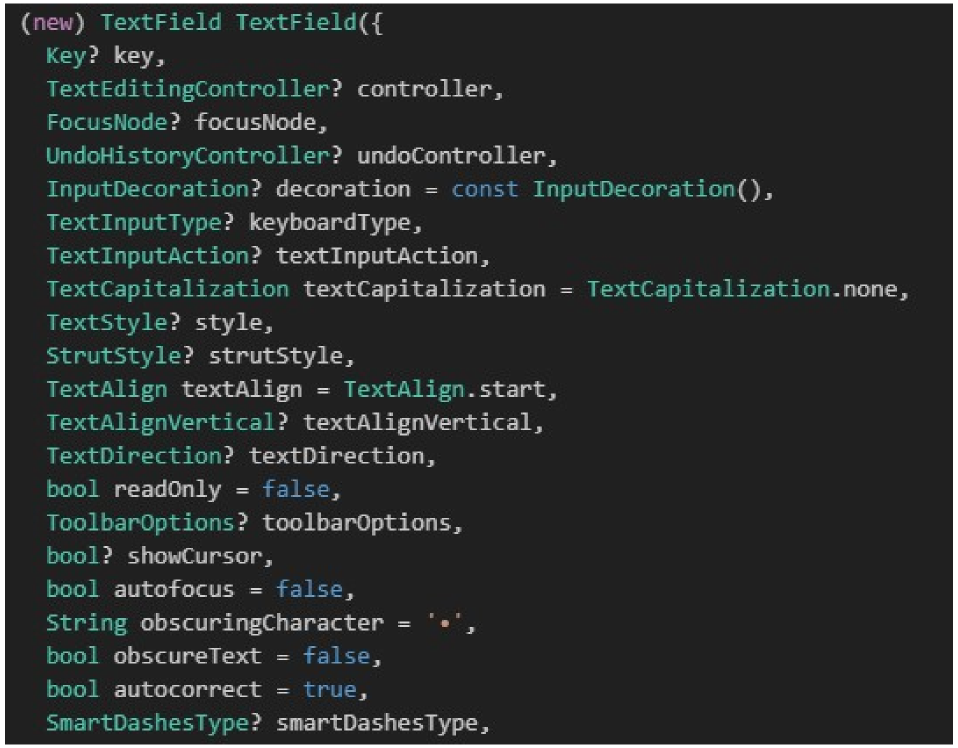
```
TextField(
  onChanged: (value) {
    print(value);
    textFieldValue = value;
  },
), // TextField
```



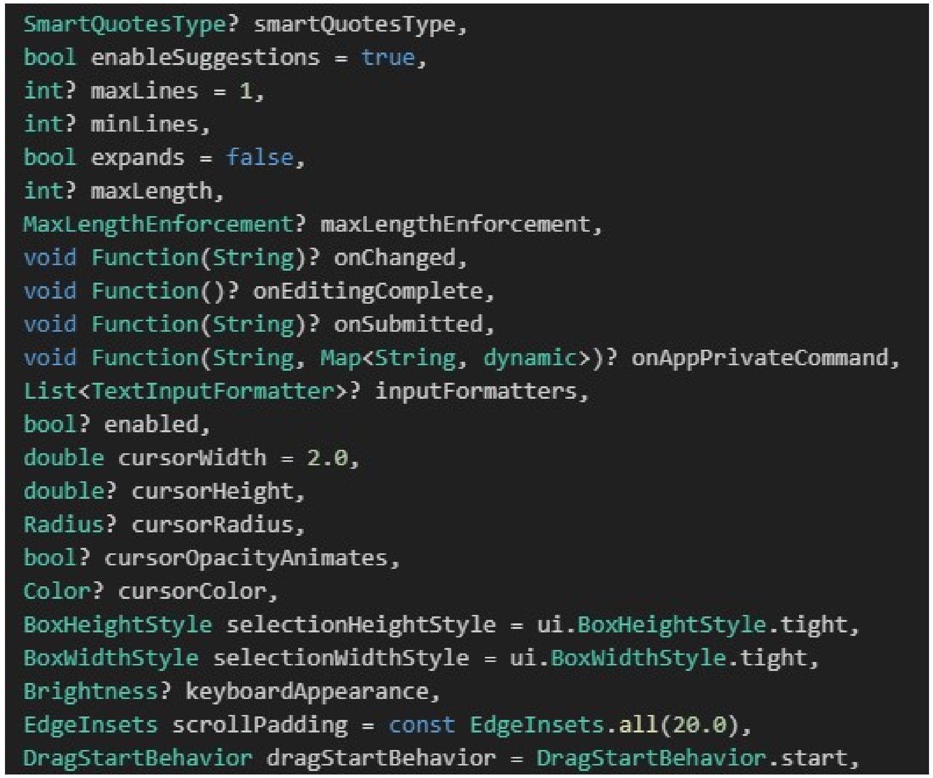
```
class _MainPageState extends State<MainPage> {
  double sliderValue = 0;
  String textFieldValue = "";
```



TextField c-tor has a lot of parameters



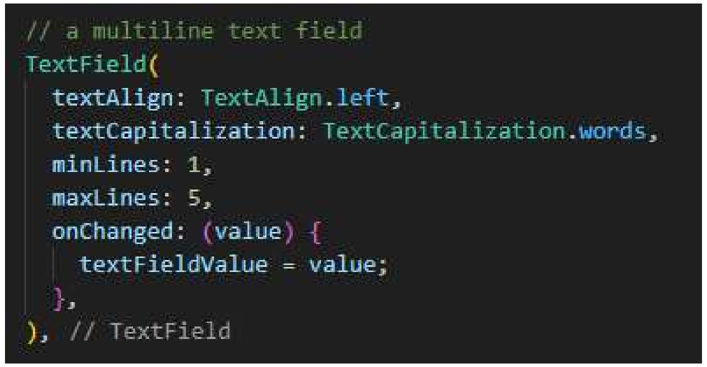
```
(new) TextField TextField({
  Key? key,
  TextEditingController? controller,
  FocusNode? focusNode,
  UndoHistoryController? undoController,
  InputDecoration? decoration = const InputDecoration(),
  TextInputType? keyboardType,
  TextInputAction? textInputAction,
  TextCapitalization textCapitalization = TextCapitalization.none,
  TextStyle? style,
  StrutStyle? strutStyle,
  TextAlign textAlign = TextAlign.start,
  TextAlignVertical? textAlignVertical,
  TextDirection? textDirection,
  bool readOnly = false,
  ToolbarOptions? toolbarOptions,
  bool? showCursor,
  bool autofocus = false,
  String obscuringCharacter = '•',
  bool obscureText = false,
  bool autocorrect = true,
  SmartDashesType? smartDashesType,
```



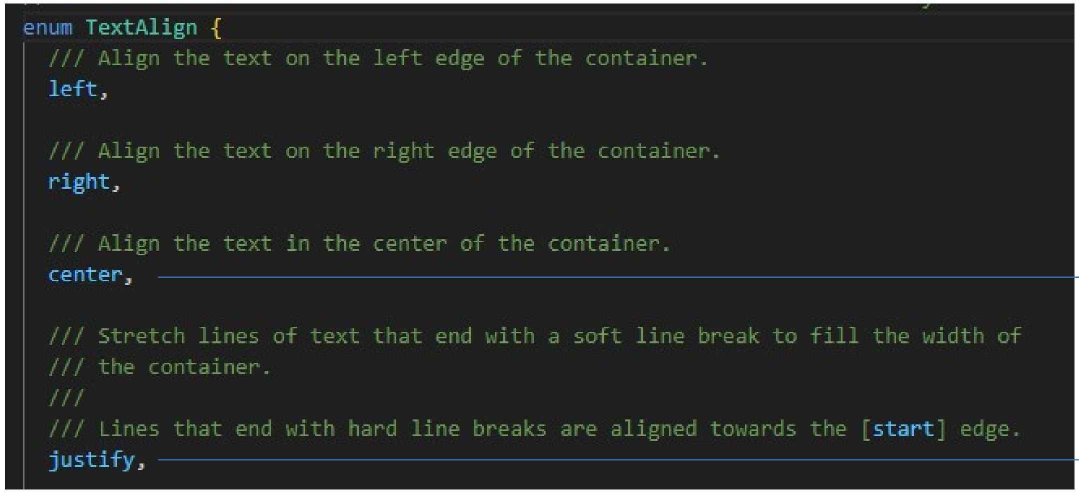
```
SmartQuotesType? smartQuotesType,
bool enableSuggestions = true,
int? maxLines = 1,
int? minLines,
bool expands = false,
int? maxLength,
MaxLengthEnforcement? maxLengthEnforcement,
void Function(String)? onChanged,
void Function()? onEditingComplete,
void Function(String)? onSubmitted,
void Function(String, Map<String, dynamic>)? onAppPrivateCommand,
List<TextInputFormatter>? inputFormatters,
bool? enabled,
double cursorWidth = 2.0,
double? cursorHeight,
Radius? cursorRadius,
bool? cursorOpacityAnimates,
Color? cursorColor,
BoxHeightStyle selectionHeightStyle = ui.BoxHeightStyle.tight,
BoxWidthStyle selectionWidthStyle = ui.BoxWidthStyle.tight,
Brightness? keyboardAppearance,
EdgeInsets scrollPadding = const EdgeInsets.all(20.0),
DragStartBehavior dragStartBehavior = DragStartBehavior.start,
```

and even more …

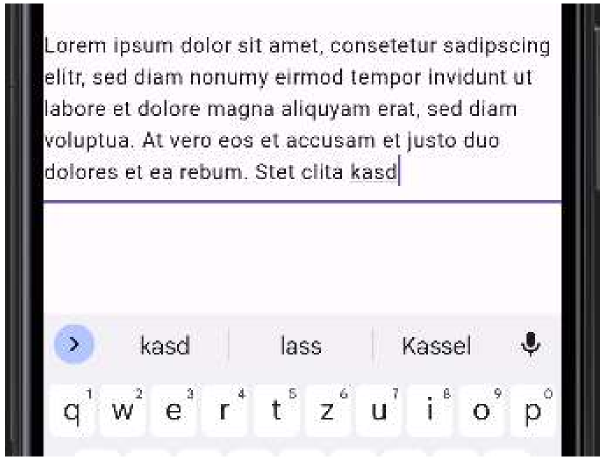
Multiline TextField with text alignment



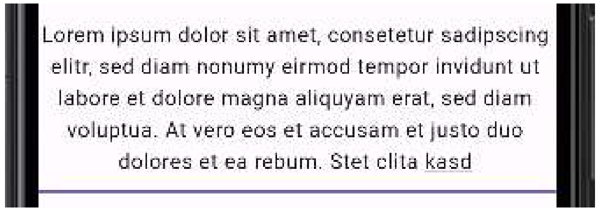
```
// a multiline text field
TextField(
  textAlign: TextAlign.left,
  textCapitalization: TextCapitalization.words,
  minLines: 1,
  maxLines: 5,
  onChanged: (value) {
    textFieldValue = value;
  },
), // TextField
```



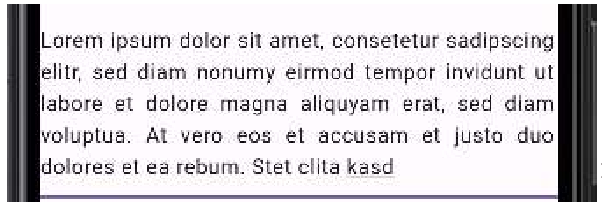
```
enum TextAlign {
  /// Align the text on the left edge of the container.
  left,

  /// Align the text on the right edge of the container.
  right,

  /// Align the text in the center of the container.
  center,

  /// Stretch lines of text that end with a soft line break to fill the width of
  /// the container.
  ///
  /// Lines that end with hard line breaks are aligned towards the [start] edge.
  justify,
```



Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd

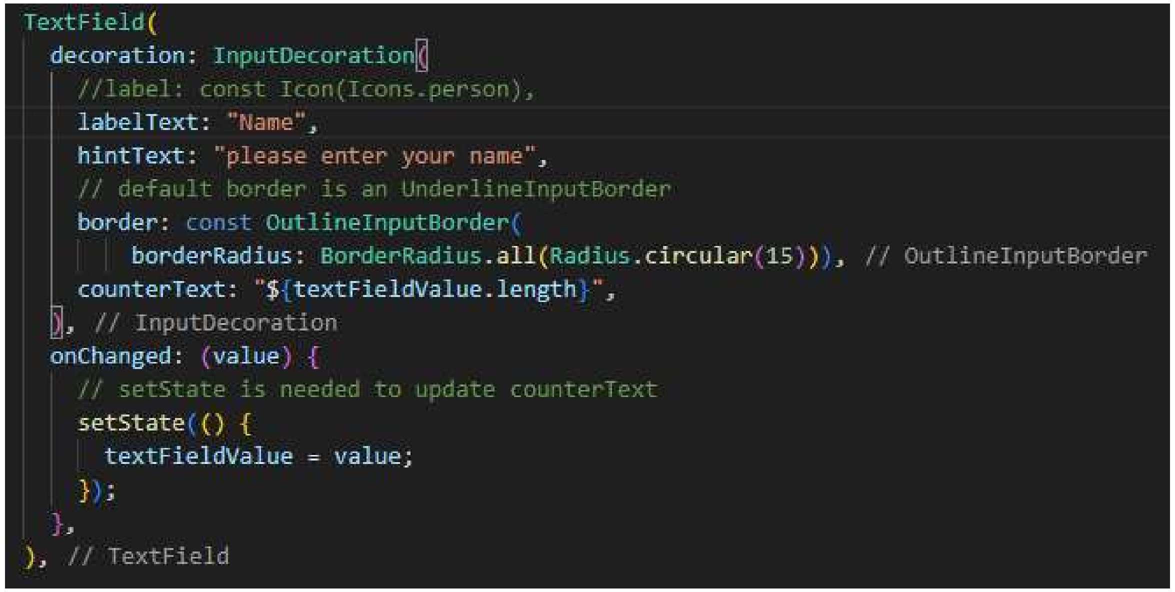


Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd



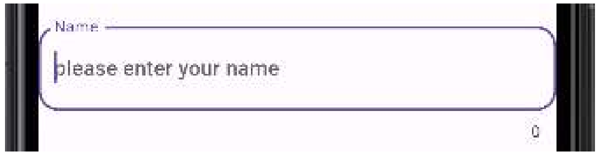
Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd

TextField with decoration

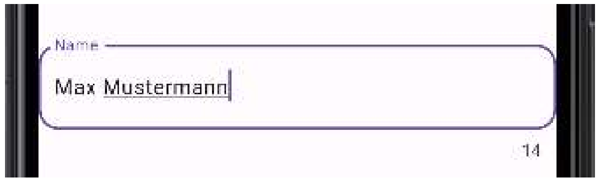


```
TextField(
  decoration: InputDecoration(
    //label: const Icon(Icons.person),
    labelText: "Name",
    hintText: "please enter your name",
    // default border is an UnderlineInputBorder
    border: const OutlineInputBorder(
        borderRadius: BorderRadius.all(Radius.circular(15))), // OutlineInputBorder
    counterText: "${textFieldValue.length}",
  ), // InputDecoration
  onChanged: (value) {
    // setState is needed to update counterText
    setState(() {
      textFieldValue = value;
    });
  },
), // TextField
```

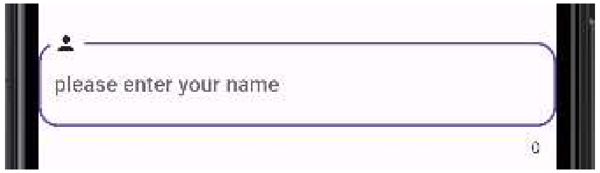
Without user input:



With user input:



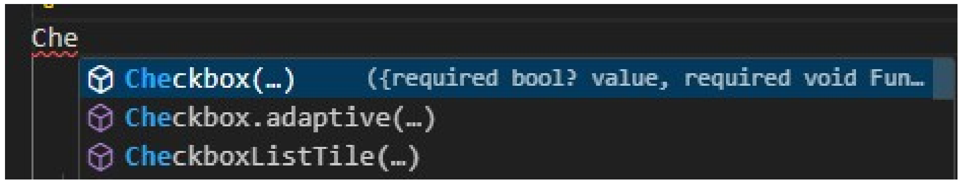
Using “label” instead of “labelText” *) :



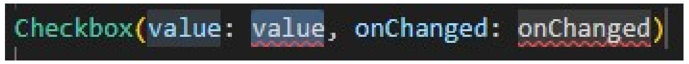
*) you cannot define both label and labelText (this leads to an exception)

Define a Checkbox

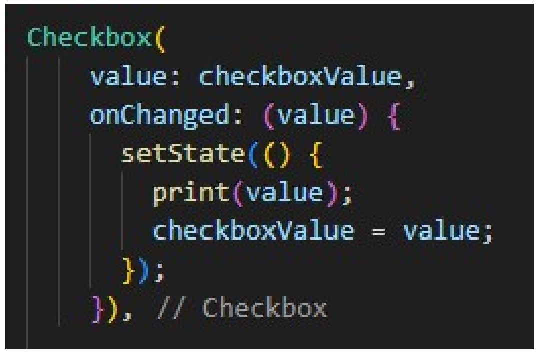
IntelliSense of VS Code helps as usual:



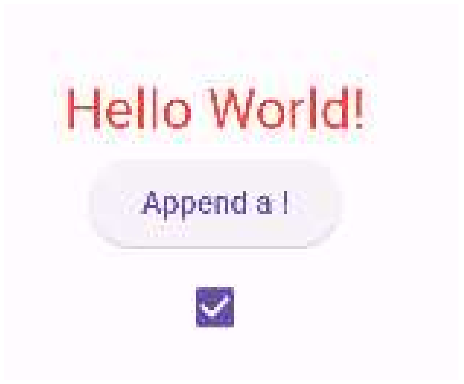
```
Che
  Checkbox(…)        ({required bool? value, required void Fun…
  Checkbox.adaptive(…)
  CheckboxListTile(…)
```

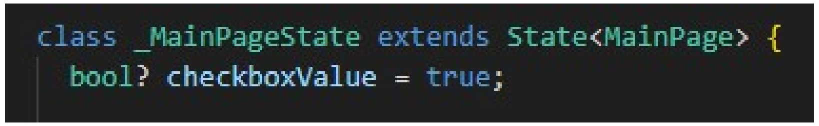


```
Checkbox(value: value, onChanged: onChanged)
```



```
Checkbox(
    value: checkboxValue,
    onChanged: (value) {
      setState(() {
        print(value);
        checkboxValue = value;
      });
    }), // Checkbox
```

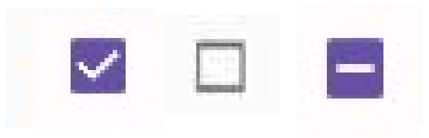




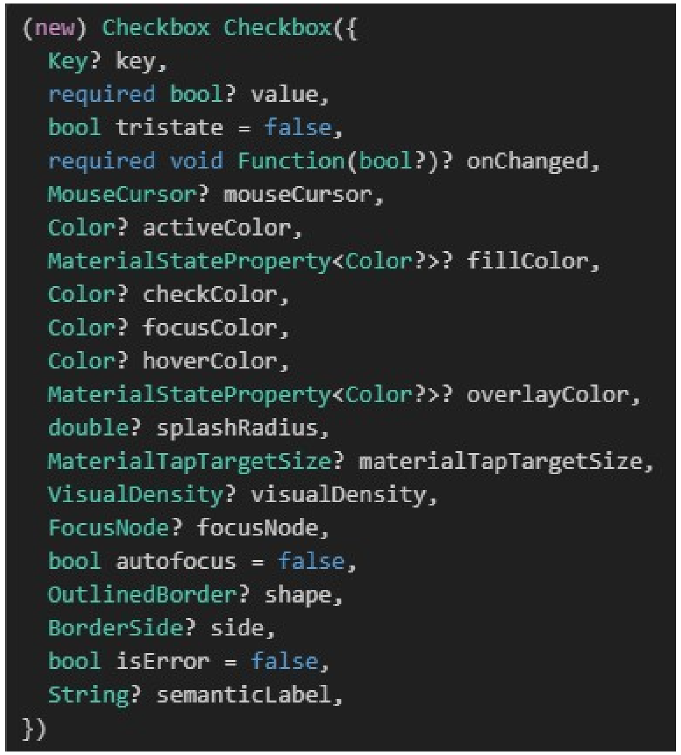
```
class _MainPageState extends State<MainPage> {
  bool? checkboxValue = true;
```

Type “bool**?**” is needed for **tristate checkboxes** with values true, false, null. See next slide.

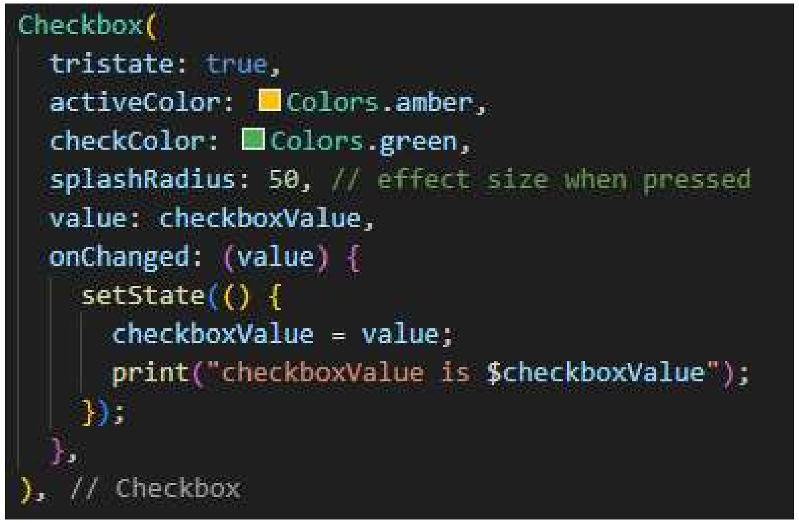
Tristate checkbox



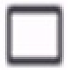
IntelliSense shows the c-tor arguments:



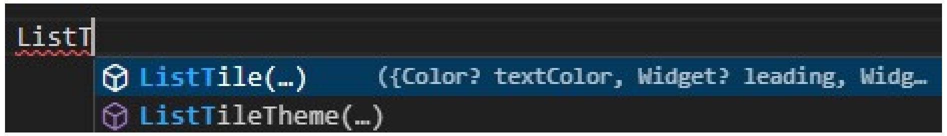
```
(new) Checkbox Checkbox({
  Key? key,
  required bool? value,
  bool tristate = false,
  required void Function(bool?)? onChanged,
  MouseCursor? mouseCursor,
  Color? activeColor,
  MaterialStateProperty<Color?>? fillColor,
  Color? checkColor,
  Color? focusColor,
  Color? hoverColor,
  MaterialStateProperty<Color?>? overlayColor,
  double? splashRadius,
  MaterialTapTargetSize? materialTapTargetSize,
  VisualDensity? visualDensity,
  FocusNode? focusNode,
  bool autofocus = false,
  OutlinedBorder? shape,
  BorderSide? side,
  bool isError = false,
  String? semanticLabel,
})
```



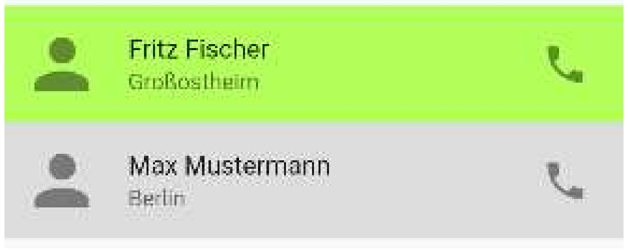
```
Checkbox(
  tristate: true,
  activeColor: Colors.amber,
  checkColor: Colors.green,
  splashRadius: 50, // effect size when pressed
  value: checkboxValue,
  onChanged: (value) {
    setState(() {
      checkboxValue = value;
      print("checkboxValue is $checkboxValue");
    });
  },
), // Checkbox
```

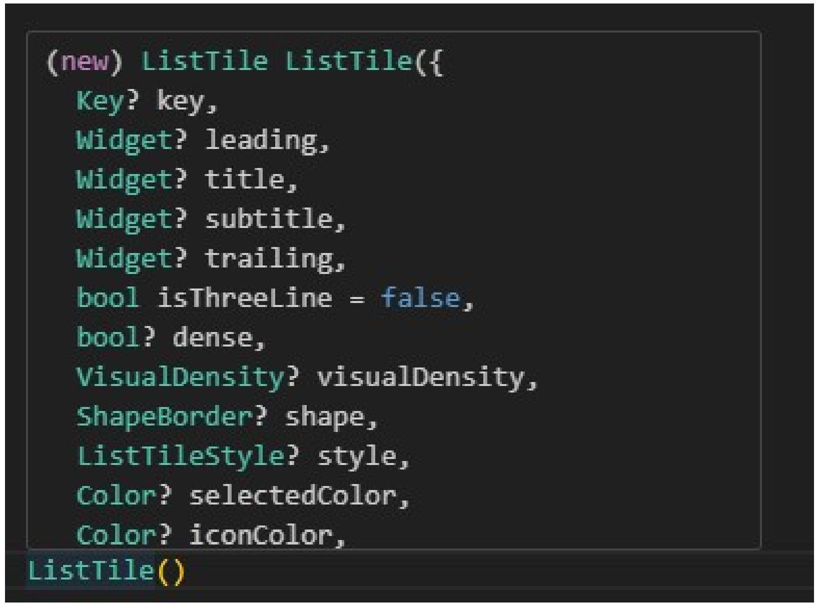
checkboxValue:	true	
	false	
	null	

Define ListTiles*) as preparation for CheckboxListTiles

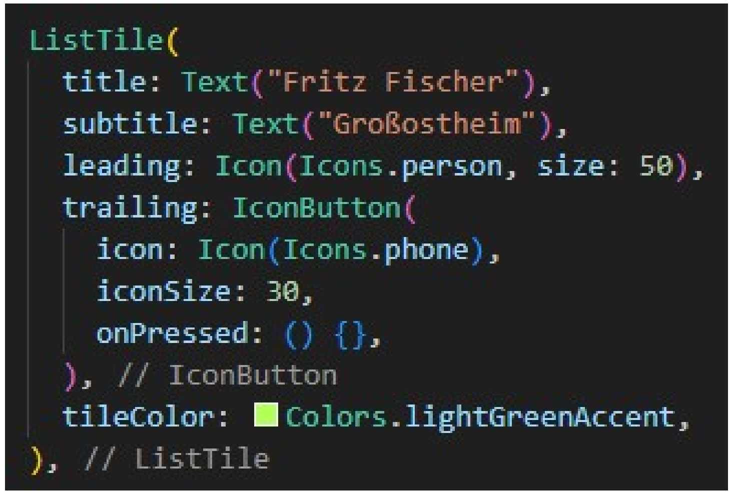


```
ListT
  ListTile(…)        ({Color? textColor, Widget? leading, Widg…
  ListTileTheme(…)
```





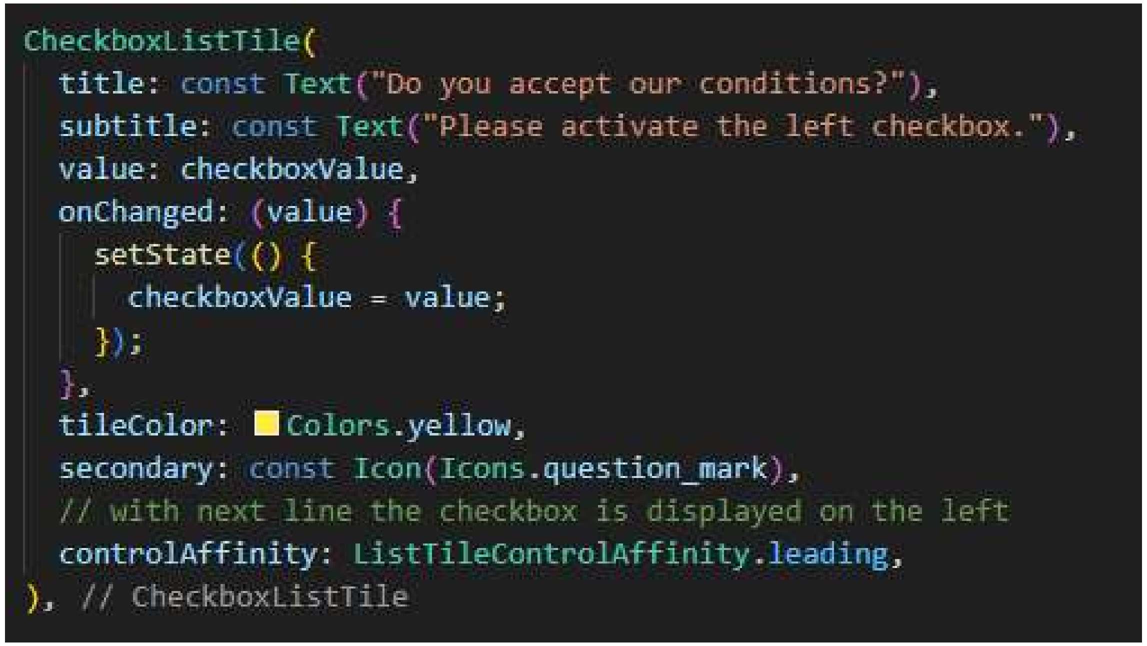
```
(new) ListTile ListTile({
  Key? key,
  Widget? leading,
  Widget? title,
  Widget? subtitle,
  Widget? trailing,
  bool isThreeLine = false,
  bool? dense,
  VisualDensity? visualDensity,
  ShapeBorder? shape,
  ListTileStyle? style,
  Color? selectedColor,
  Color? iconColor,
ListTile()
```



```
ListTile(
  title: Text("Fritz Fischer"),
  subtitle: Text("Großostheim"),
  leading: Icon(Icons.person, size: 50),
  trailing: IconButton(
    icon: Icon(Icons.phone),
    iconSize: 30,
    onPressed: () {},
  ), // IconButton
  tileColor: 🟩 Colors.lightGreenAccent,
), // ListTile
```

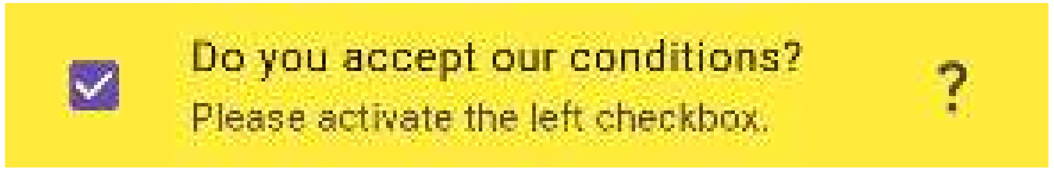
*) “Tile” in German: Fliese, Ziegel, Plättchen

CheckboxListTile

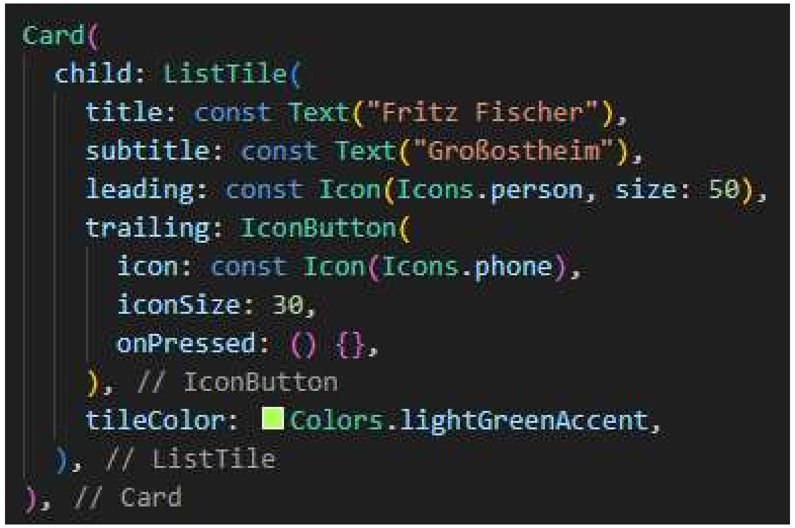


```dart
CheckboxListTile(
  title: const Text("Do you accept our conditions?"),
  subtitle: const Text("Please activate the left checkbox."),
  value: checkboxValue,
  onChanged: (value) {
    setState(() {
      checkboxValue = value;
    });
  },
  tileColor: 🟨Colors.yellow,
  secondary: const Icon(Icons.question_mark),
  // with next line the checkbox is displayed on the left
  controlAffinity: ListTileControlAffinity.leading,
), // CheckboxListTile
```

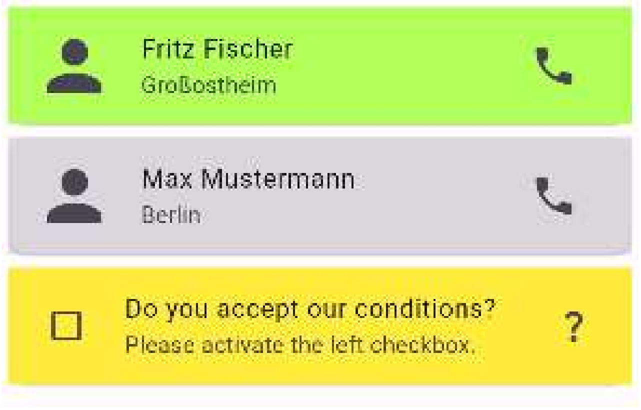
Tapping anywhere in the tile is modifying the Checkbox !


Do you accept our conditions?
Please activate the left checkbox.

Surrounding ListTile with a Card widget



```
Card(
  child: ListTile(
    title: const Text("Fritz Fischer"),
    subtitle: const Text("Großostheim"),
    leading: const Icon(Icons.person, size: 50),
    trailing: IconButton(
      icon: const Icon(Icons.phone),
      iconSize: 30,
      onPressed: () {},
    ), // IconButton
    tileColor: 🟩Colors.lightGreenAccent,
  ), // ListTile
), // Card
```

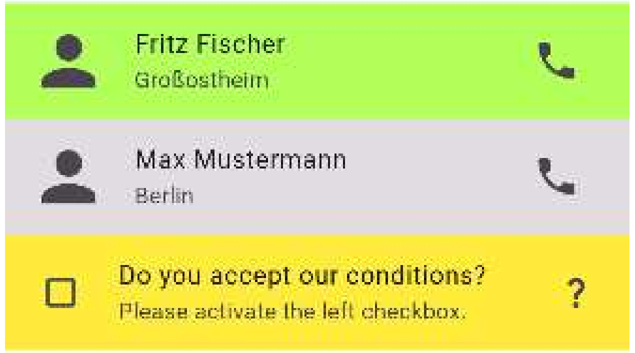


Fritz Fischer
Großostheim

Max Mustermann
Berlin

Do you accept our conditions?
Please activate the left checkbox.

Same without Card:



Fritz Fischer
Großostheim

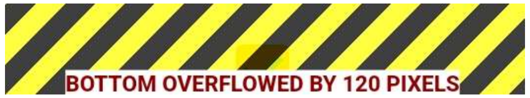
Max Mustermann
Berlin

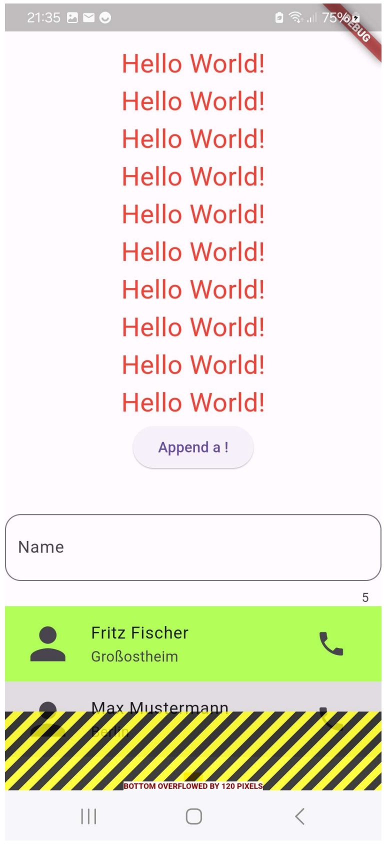
Do you accept our conditions?
Please activate the left checkbox.

Overflow at the bottom

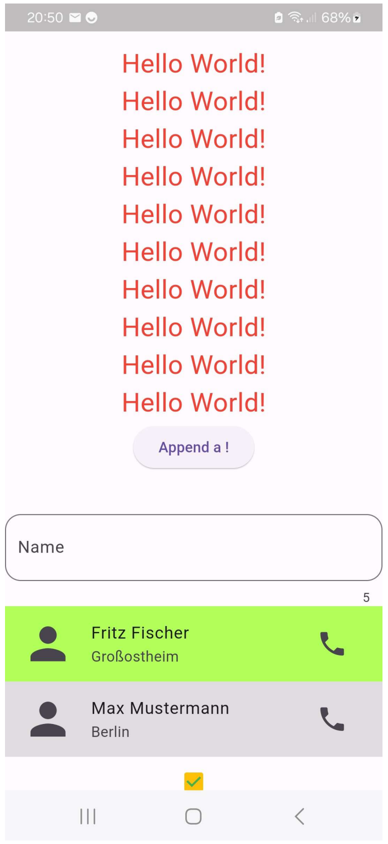
In case we have too many UI elements, those on the bottom are not visible.

When debugging, we get a hatched (in German "schraffierte") warning.





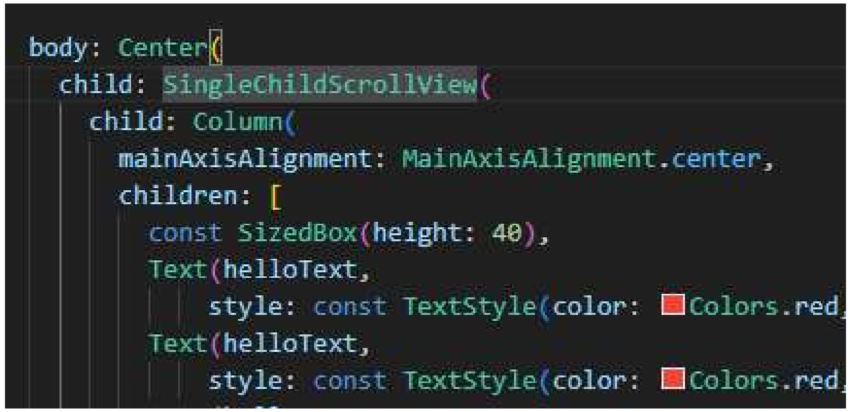
Warning shown during debugging.



When installed as Release, elements are just cut off.

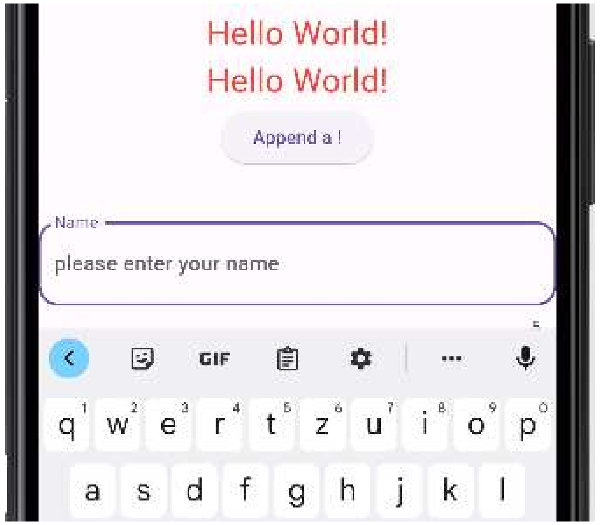
SingleChildScrollView

Solution:
Wrap the column with a SingleChildScrollView:

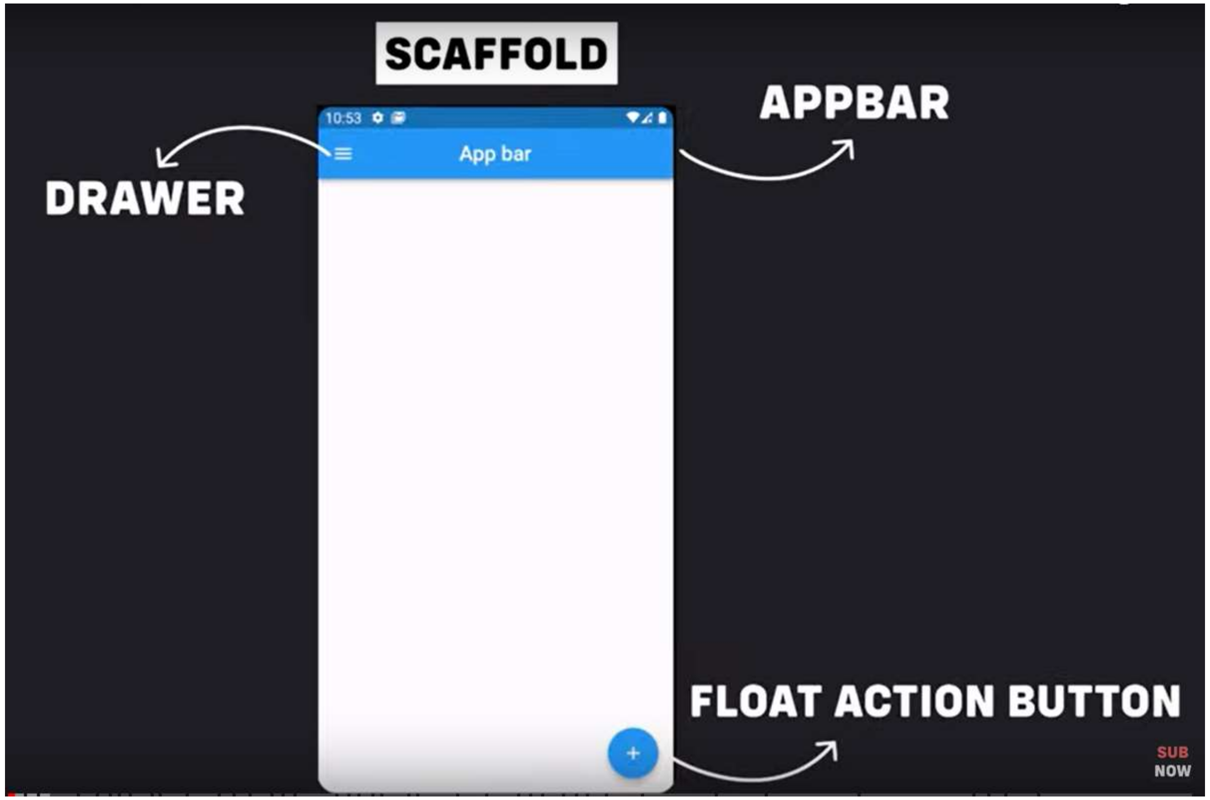


```
body: Center(
  child: SingleChildScrollView(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const SizedBox(height: 40),
        Text(helloText,
            style: const TextStyle(color: Colors.red,
        Text(helloText,
            style: const TextStyle(color: Colors.red,
```

Additional advantage:
during typing in a TextField, it is automatically scrolled up above the virtual android keyboard:

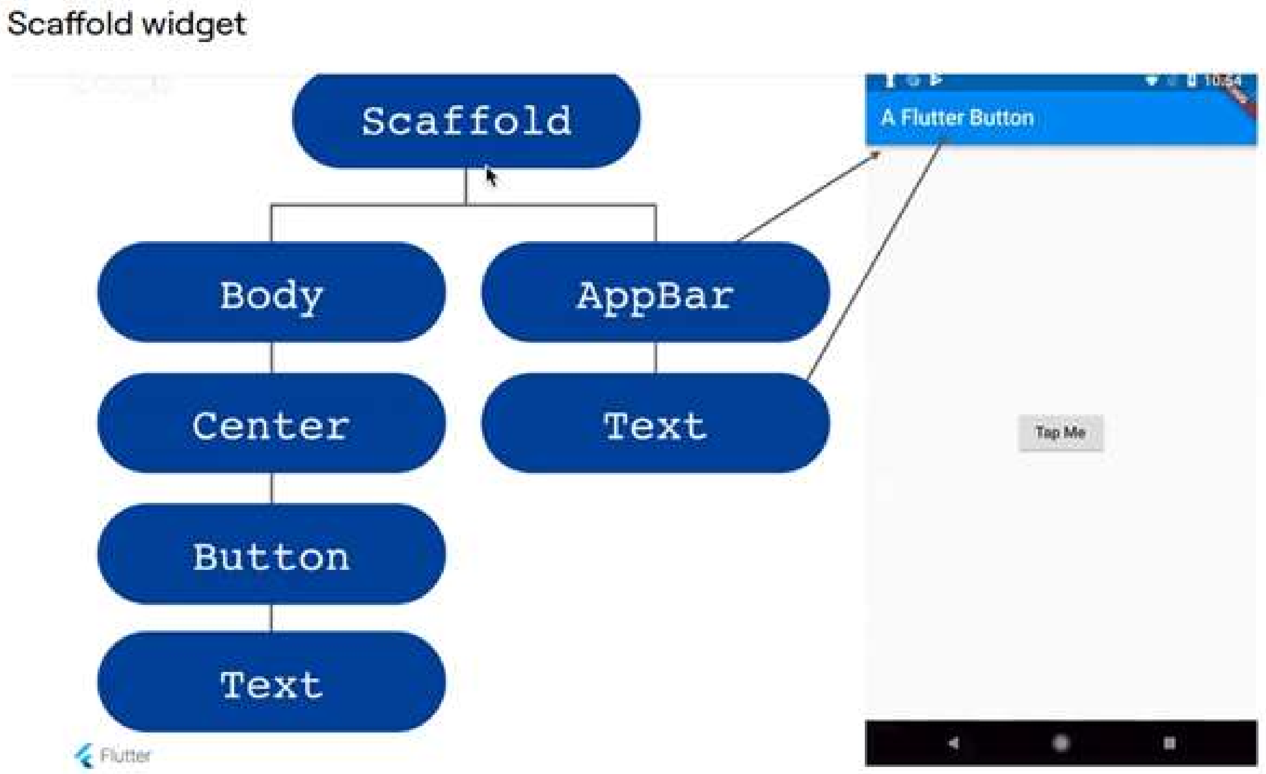


Adding some more standard UI elements



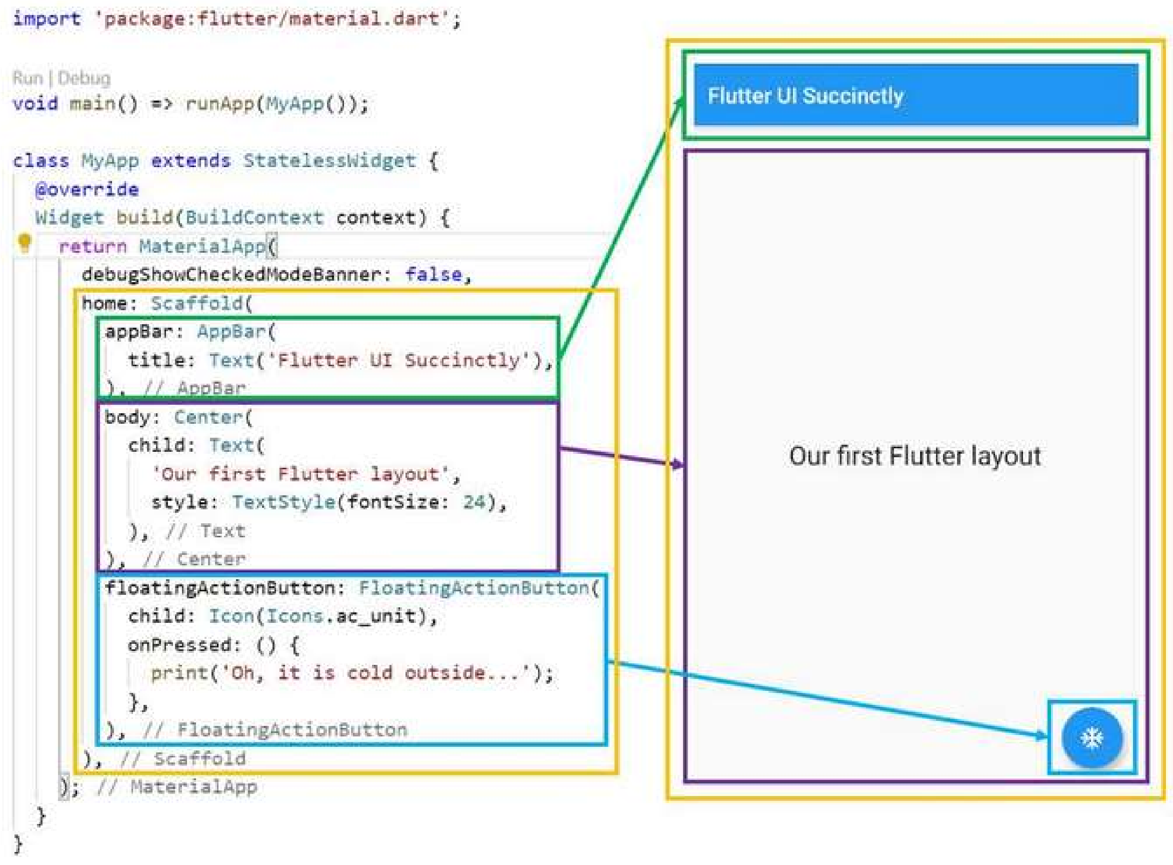
Copied from https://www.youtube.com/watch?app=desktop&v=D4nhaszNW4o at 3:34

The widget tree below Scaffold



Copied from https://antonioleiva.com/flutter-mobile-development-for-busy-people-droidkast-05/

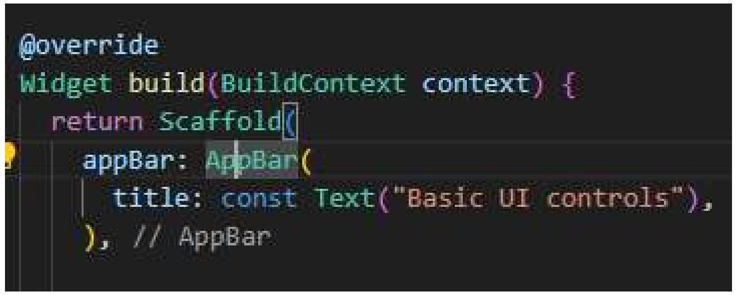
Scaffold’s c-tor arguments

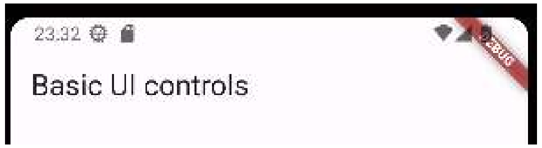


```
import 'package:flutter/material.dart';

Run | Debug
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter UI Succinctly'),
        ), // AppBar
        body: Center(
          child: Text(
            'Our first Flutter layout',
            style: TextStyle(fontSize: 24),
          ), // Text
        ), // Center
        floatingActionButton: FloatingActionButton(
          child: Icon(Icons.ac_unit),
          onPressed: () {
            print('Oh, it is cold outside...');
          },
        ), // FloatingActionButton
      ), // Scaffold
    ); // MaterialApp
  }
}
```

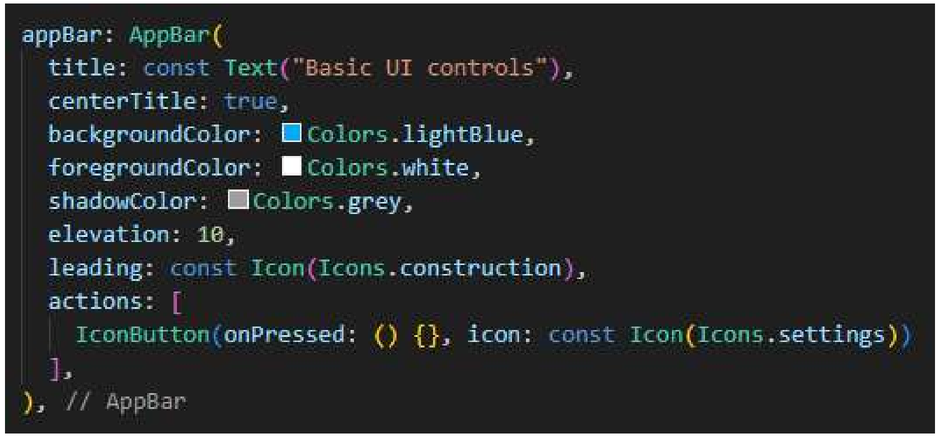
Copied from https://antonioleiva.com/flutter-mobile-development-for-busy-people-droidkast-05/

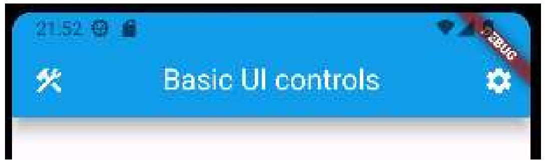
Defining an AppBar


```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text("Basic UI controls"),
    ), // AppBar
```

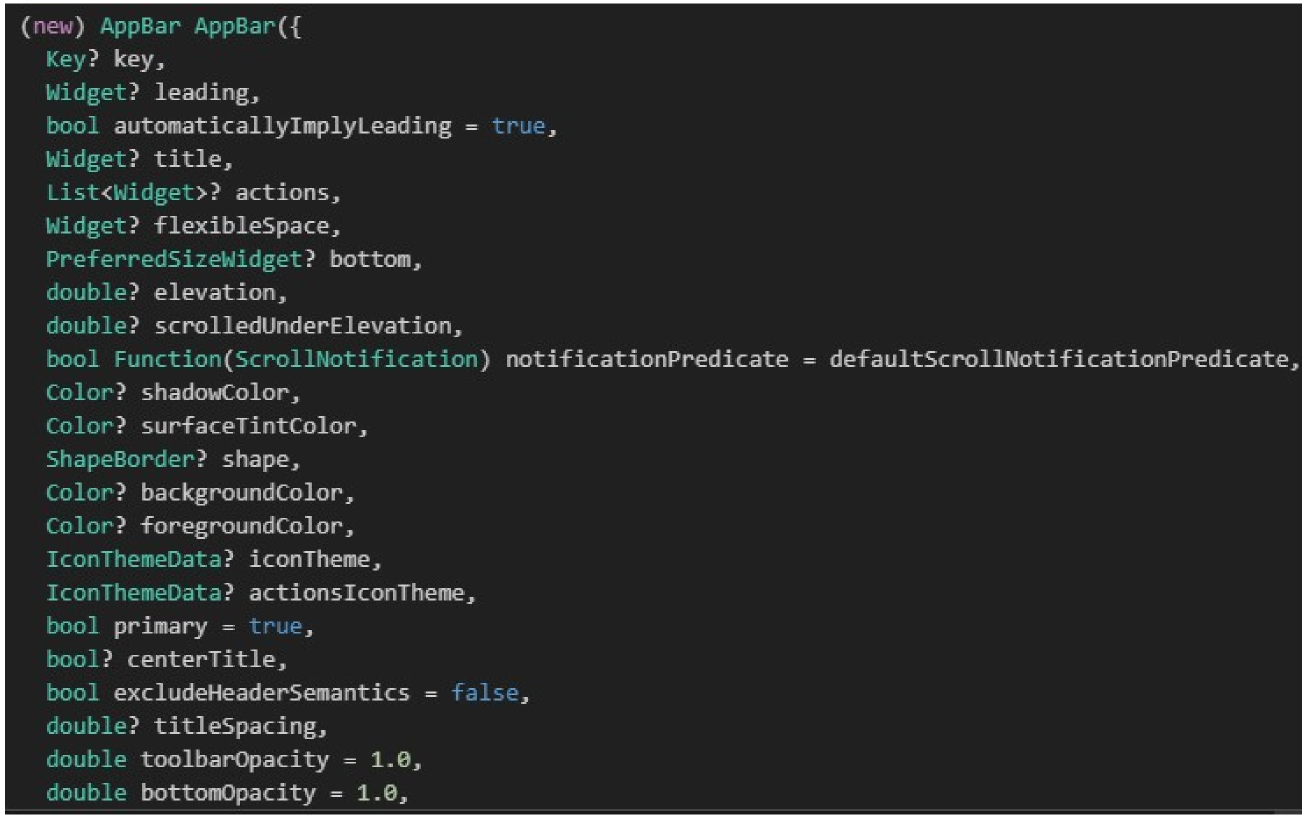


Setting more properties:


```
appBar: AppBar(
  title: const Text("Basic UI controls"),
  centerTitle: true,
  backgroundColor: ■Colors.lightBlue,
  foregroundColor: □Colors.white,
  shadowColor: ■Colors.grey,
  elevation: 10,
  leading: const Icon(Icons.construction),
  actions: [
    IconButton(onPressed: () {}, icon: const Icon(Icons.settings))
  ],
), // AppBar
```

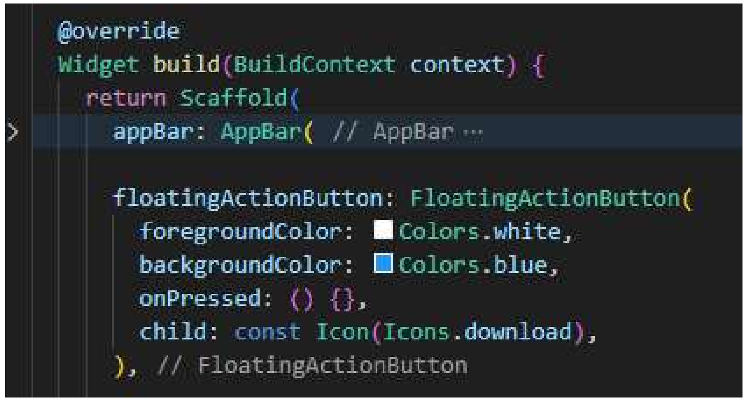


More c-tor arguments for an AppBar



```
(new) AppBar AppBar({
  Key? key,
  Widget? leading,
  bool automaticallyImplyLeading = true,
  Widget? title,
  List<Widget>? actions,
  Widget? flexibleSpace,
  PreferredSizeWidget? bottom,
  double? elevation,
  double? scrolledUnderElevation,
  bool Function(ScrollNotification) notificationPredicate = defaultScrollNotificationPredicate,
  Color? shadowColor,
  Color? surfaceTintColor,
  ShapeBorder? shape,
  Color? backgroundColor,
  Color? foregroundColor,
  IconThemeData? iconTheme,
  IconThemeData? actionsIconTheme,
  bool primary = true,
  bool? centerTitle,
  bool excludeHeaderSemantics = false,
  double? titleSpacing,
  double toolbarOpacity = 1.0,
  double bottomOpacity = 1.0,
```

Defining a Floating Action Button



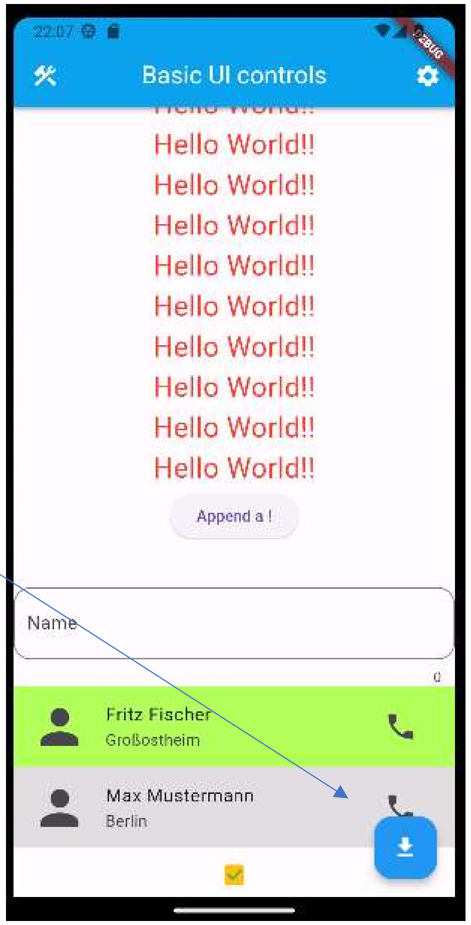
```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar( // AppBar ···

    floatingActionButton: FloatingActionButton(
      foregroundColor: Colors.white,
      backgroundColor: Colors.blue,
      onPressed: () {},
      child: const Icon(Icons.download),
    ), // FloatingActionButton
```

Properties:

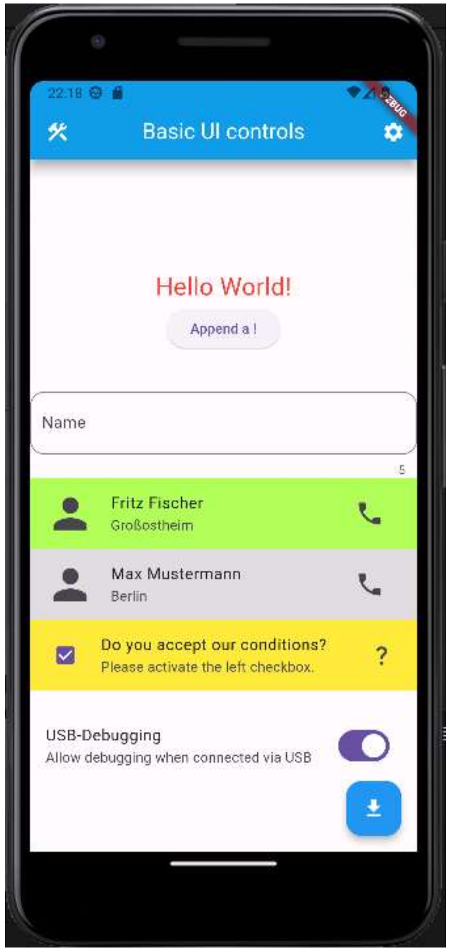
It is displayed above the other UI elements.

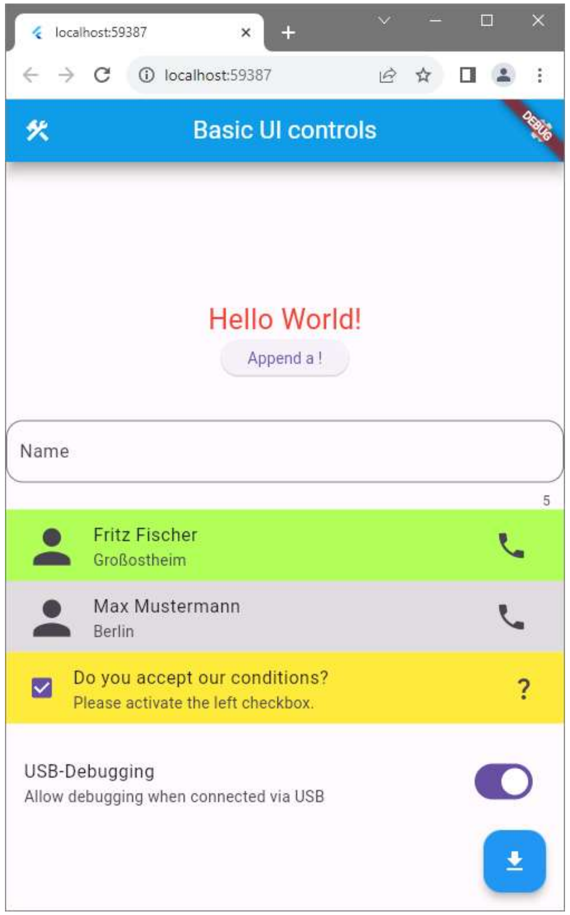
It stays at a fixed position during scrolling.

It “jumps up” when a virtual keyboard is swapped in.

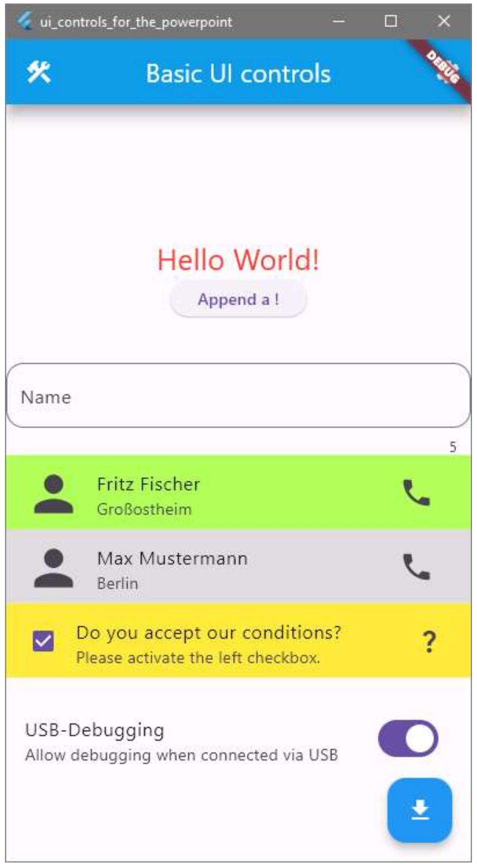


Same UI on different devices

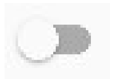
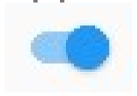




As Windows app:

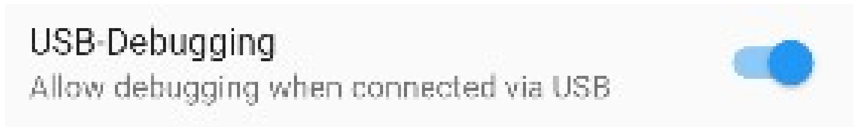

ui_controls_for_the_powerpoint
DEBUG
Basic UI controls
Hello World!
Append a !
Name
5
Fritz Fischer
Großostheim
Max Mustermann
Berlin
Do you accept our conditions?
Please activate the left checkbox.
?
USB-Debugging
Allow debugging when connected via USB

Exercise (based on what you learned about Checkbox and CheckboxListTile)

- Define a Switch in your App.
  It has 2 states:  

  Hint: Let IntelliSense help you by entering “Sw” in a new line of your code.

- Define a SwitchListTile in your App which looks like



USB-Debugging
Allow debugging when connected via USB

- Find and watch YouTube videos on Flutter Switch and Flutter SwitchListTile.

  Some samples:
  https://www.youtube.com/watch?v=MnR2xBcqgw8
  https://www.youtube.com/watch?v=J_8bZ2trhrM
  https://www.youtube.com/watch?v=0igIjvtEWNU