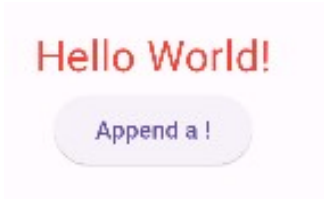
Stateful widgets and dynamic UI elements

- 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 });
```

Go to Definition F12
Go to Type Definition

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:

```
26                ElevatedButton(
27                    onPressed: () {
28                      helloText += "!";
29                      print("helloText is now $helloText");
30                    },
```

```
ROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS
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!!!
```

Debugging build method of Stateless Widgets

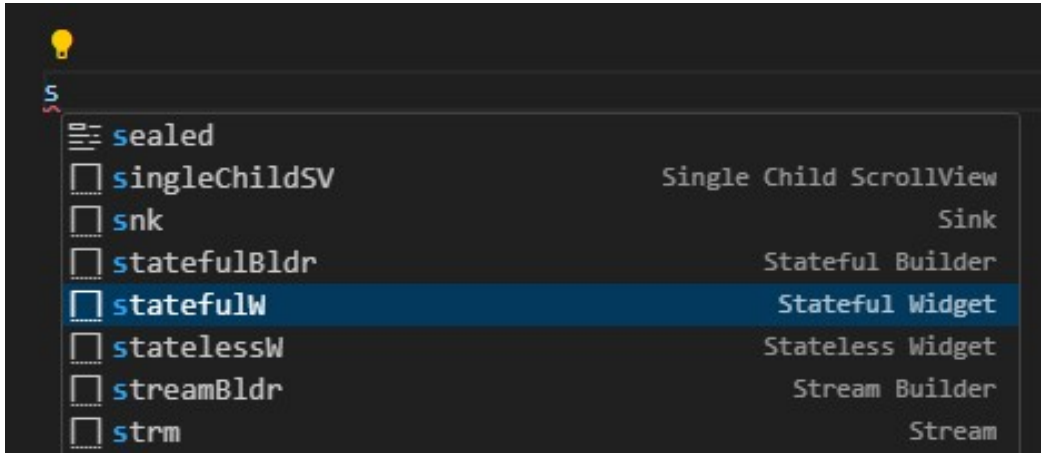
Set breakpoint on build method:

```
16    @override
17    Widget build(BuildContext context) {
18      return MaterialApp(
19        home: Scaffold(
20          body: Center(
21            child: Column(
22              mainAxisAlignment: MainAxisAlignment.center,
23              children: [
24                Text(helloText,
25                    style: TextStyle(color: Colors.red, fontSize: 25)), // Text
26                ElevatedButton(
27                    onPressed: () {
28                      helloText += "!";
29                      print("helloText is now $helloText");
30                    },
31                    child: Text("Append a !")) // ElevatedButton
32              ],
33            ), // 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:



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.

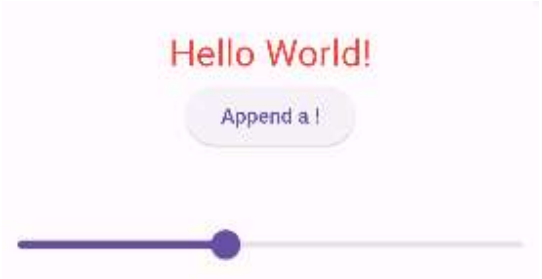
Alternatives how to call “setState”

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.

Define a TextField

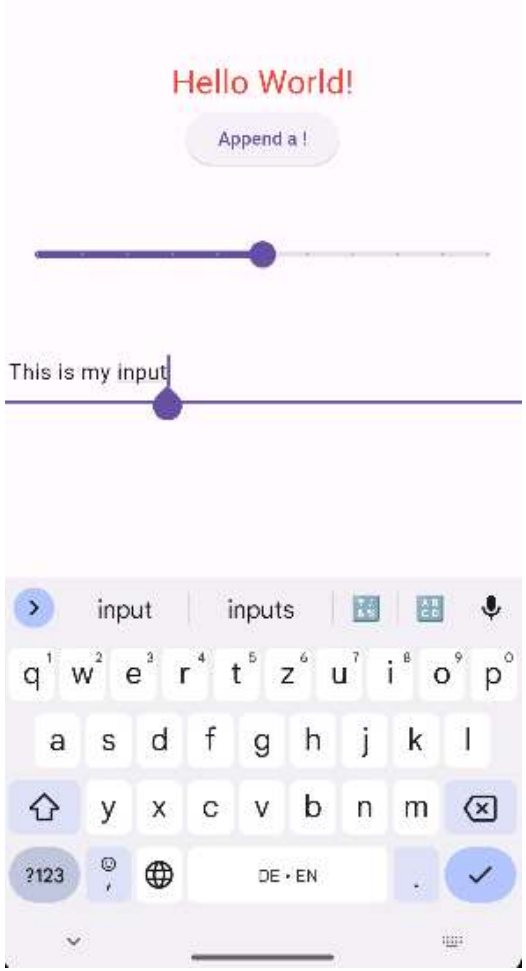
Again you can start with IntelliSense of VS Code:

```
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

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

```dart
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,
```