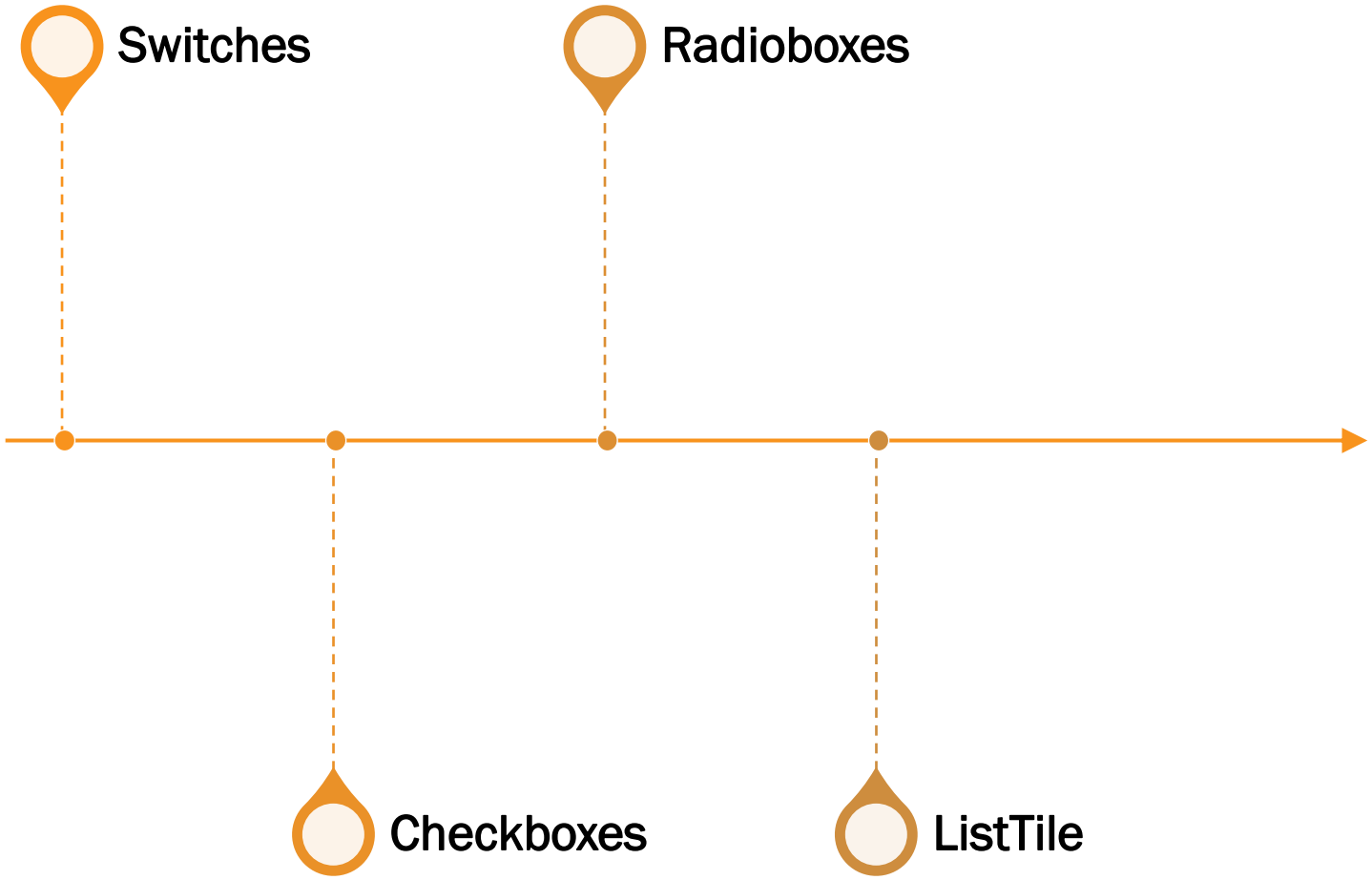
Flutter Framework

COURSE 8 (REV 6)

GAVRILUT DRAGOS

Agenda



Switches

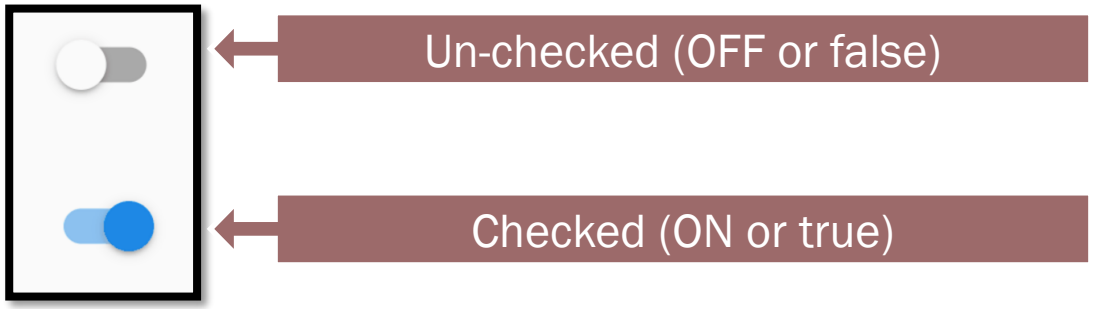
Switches

Switches are UX object that reflect an on/off state (true / false) value.

It is usually recommended to use a switch when the change happens immediately → meaning that once you enable it, the change that reflects it will be associated (for example: enabling Wifi)

Flutter has one widget (call **Switch**) that can be used for this kind of tasks.

A switch looks like in the following way:



One main difference is that these widgets don't have an associated label (a text that explain what that checkbox represents). This means that usually, a Widget like this will be used as part of a Raw widget (so that it can incorporate a Text widget).

Switch

Constructor:

```
Switch({
   required bool value,
   required void Function(bool)? onChanged,
   Color? activeColor,
   Color? activeTrackColor,
   Color? inactiveThumbColor,
   Color? inactiveTrackColor,
   Color? focusColor,
   Color? hoverColor,
   … }
)
```

Switch

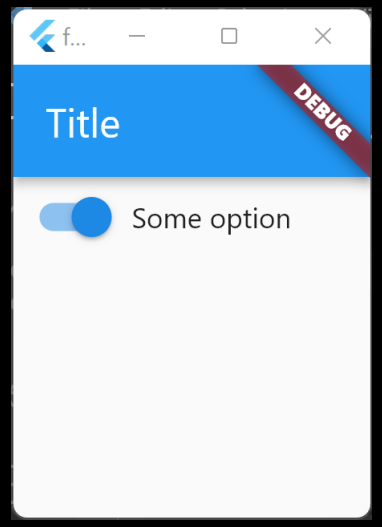
Let's see a very simple example.

```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatefulWidget {
  @override
  State<MyApp> createState() => MyAppState(); }
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(child: Switch(onChanged: (b) => {}, value: true))));
  }
}
```

Switch

So … now we have a switch, but not text (label) associated with it. Let's see how we can add one.

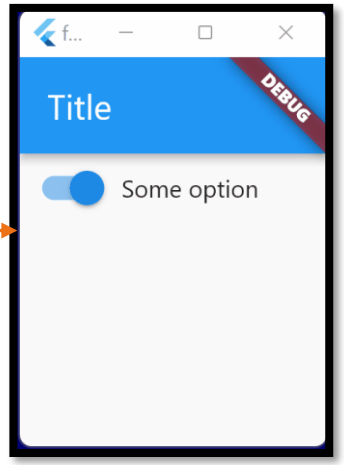
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Switch(onChanged: (b) => {}, value: true),
              Text("Some option")
            ])));
  }
}
```

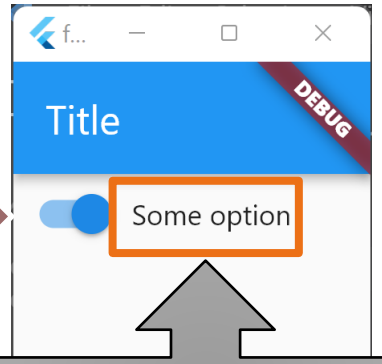


Switch

There are however a couple of things you might notice once .

You can click on the switch, but nothing happens. That's because in the code, there is no state (no variable) that changes when onChanged callback is called.





You can not click on the text and expect the switch to change (like in a regular checkbox)

Switch

A functional example:

```
class MyAppState extends State<MyApp> {
  bool optValue = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Switch(
                  onChanged: (b) => setState(() => optValue = b),
                  value: optValue),
              Text(optValue ? "I agree" : "I disagree")
            ])));
  }
}
```

Switch

To disable a switch, just change the onChange parameter value to null. Notice that just the switch will be disable, the associated text will not.

```
class MyAppState extends State<MyApp> {
  bool optValue = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Switch(onChanged: null, value: optValue),
              Text(optValue ? "I agree" : "I disagree")
            ])));
  }
}
```

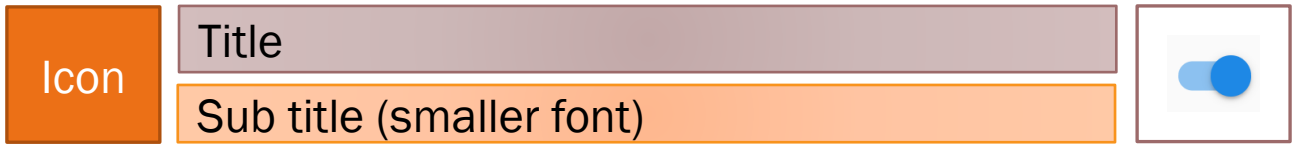
Switch

The next example sets the color of the text to look like something that was disabled.

```
class MyAppState extends State<MyApp> {
  bool optValue = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Switch(onChanged: null, value: optValue),
              Text(optValue ? "I agree" : "I disagree",
                   style: TextStyle(color: Colors.black45))
            ])));
  }
}
```

SwitchListTile

This is one way of trying to have a switch that also have a text associated. However, this does not solve the problem with clicking on the text. And since this is a very often case, a derived widget that has a label, an icon and a subtext was created ➔ **SwitchListTile**



SwitchListTile

Constructor

```
SwitchListTile({
   required bool value,
   required void ValueChanged<bool>? onChanged,
   Widget? title,
   Widget? subtitle,
   Widget? secondary,
   Color? activeColor,
   Color? tileColor,
   Color? inactiveThumbColor,
   Color? inactiveTrackColor,
   Color? selectedTileColor,
   Color? hoverColor,
   … }
)
```

```
Title label
Subtitle text
Icon
```

SwitchListTile

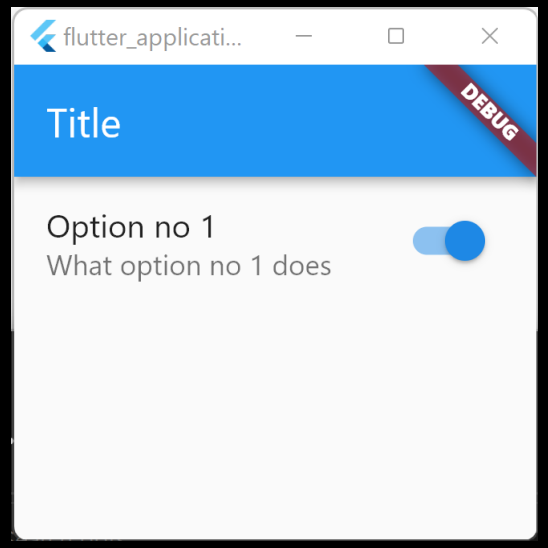
A very simple example

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: SwitchListTile(
              value: true,
              onChanged: (b) => {},
              title: Text("Option no 1"),
            )));
  }
}
```

SwitchListTile

The subtitle parameter is usually an explanation of the option.

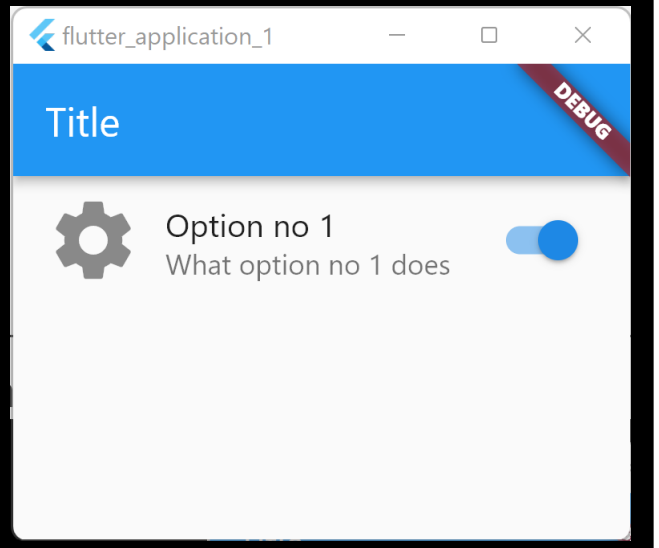
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: SwitchListTile(
              value: true,
              onChanged: (b) => {},
              title: Text("Option no 1"),
              subtitle: Text("What option no 1 does"),
            )));
  }
}
```



SwitchListTile

To show an icon in front of the option, use the secondary parameter.

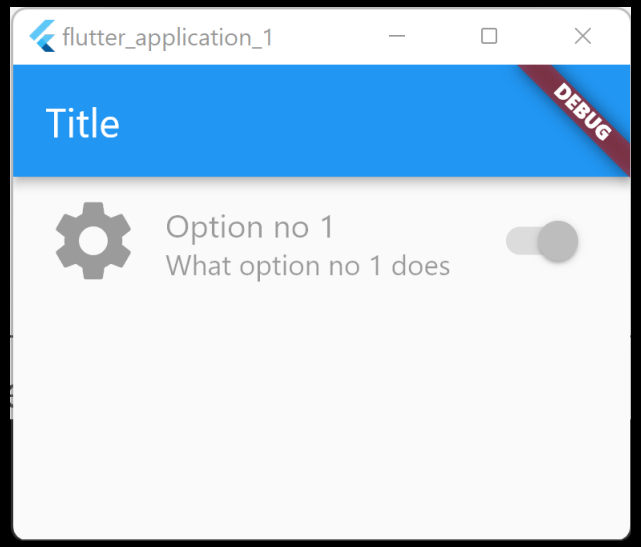
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: SwitchListTile(
              value: true,
              onChanged: (b) => {},
              title: Text("Option no 1"),
              subtitle: Text("What option no 1 does"),
              secondary: Icon(Icons.settings, size: 48),
            )));
  }
}
```



SwitchListTile

To disable the widget (icon, title, subtitle), just set the value of onChanged parameter to null.

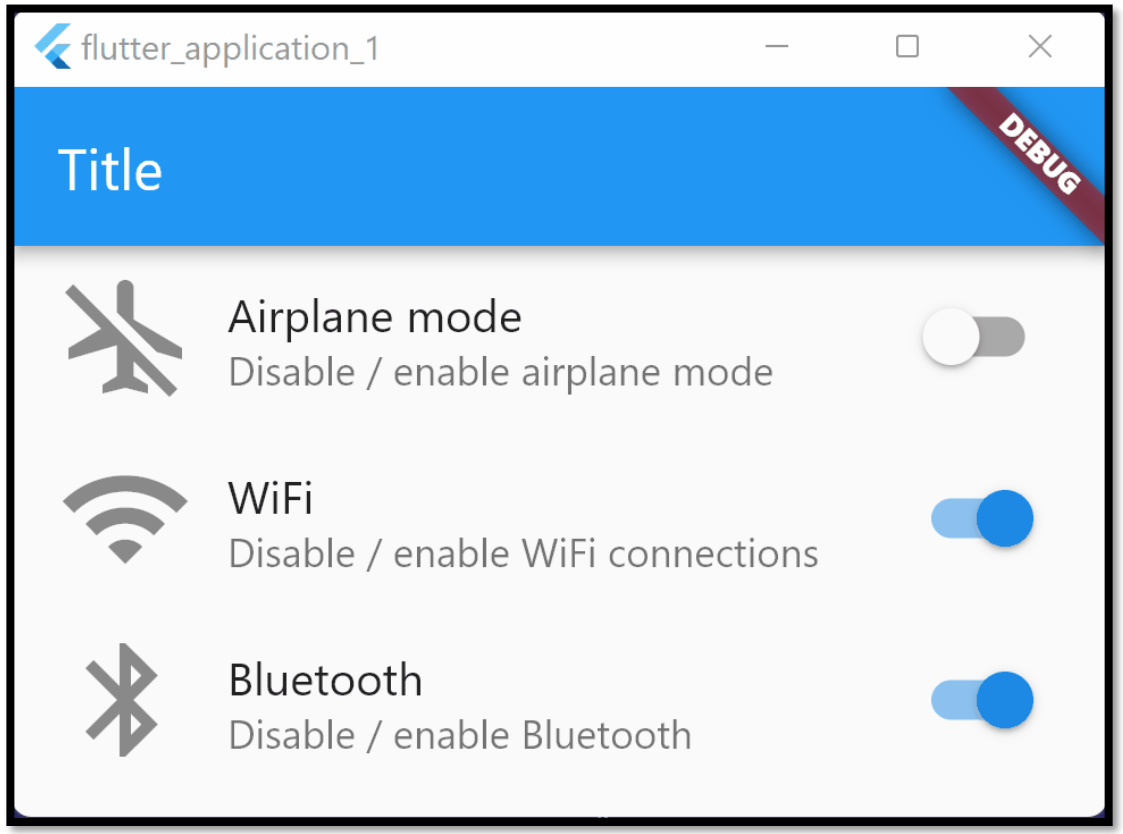
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: SwitchListTile(
              value: true,
              onChanged: null,
              title: Text("Option no 1"),
              subtitle: Text("What option no 1 does"),
              secondary: Icon(Icons.settings, size: 48),
            )));
  }
}
```



SwitchListTile

Let's build a more complex example:

- 3 options (Airplane mode, wifi and Bluetooth)
- If Airplane mode is not checked, then wifi and Bluetooth will be disabled (not checkable)
- All 3 options should have icons
- The Airplane mode icon should be different pending on airplane mode status



SwitchListTile

A more complex example

```
class MyAppState extends State<MyApp> {
  bool airplaneMode = false, enableWifi = true, enableBluetooth = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title ")),
            body: …




        ));
  }
}
```

First, we need some local Boolean variable to hold the state for the three options

SwitchListTile

A more complex example

```
class MyAppState extends State<MyApp> {
  bool airplaneMode = false, enableWifi = true, enableBluetooth = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Column(
              children: [
                SwitchListTile(…),
                SwitchListTile(…),
                SwitchListTile(…),
               ])));
  }
}
```

We will use a Column widget with three children, one for each options

SwitchListTile

A more complex example

```
class MyAppState extends State<MyApp> {
  bool airplaneMode = false, enableWifi = true, enableBluetooth = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text
            body: Column(
              children: [
                SwitchListTile(…)
                SwitchListTile(…),
                SwitchListTile(…),
                ])));
  }
}
```

```
SwitchListTile(
  value: airplaneMode,
  onChanged: (b) => setState(() { airplaneMode = b; }),
  title: Text("Airplane mode"),
  subtitle: Text("Disable / enable airplane mode"),
  secondary: Icon(airplaneMode ? Icons.airplanemode_active
                               : Icons.airplanemode_inactive,
                  size: 48),
  hoverColor: Colors.amber,
)
```

SwitchListTile

A more complex example

```
class MyAppState extends State<MyApp> {
  bool airplaneMode = false, enableWifi = true, enableBluetooth = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Column(
              children: [
                SwitchListTile(…),
                SwitchListTile(…),
                SwitchListTile(…),
                ])));
  }
}
```

```
SwitchListTile(
  value: enableWifi,
  onChanged: airplaneMode ? null
                          : (b) => setState(() {
                                       enableWifi = b;
                                   }),
  title: Text("WiFi"),
  subtitle: Text("Disable / enable WiFi connections"),
  secondary: Icon(Icons.wifi, size: 48),
)
```

SwitchListTile

A more complex example

```
class MyAppState extends State<MyApp> {
  bool airplaneMode = false, enableWifi = true, enableBluetooth = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Column(
              children: [
                SwitchListTile(…),
                SwitchListTile(…),
                SwitchListTile(…),
                ])));
  }
}
```

```
SwitchListTile(
  value: enableBluetooth,
  onChanged: airplaneMode ? null
                          : (b) => setState(() {
                                      enableBluetooth = b;
                                  }),
  title: Text("Bluetooth"),
  subtitle: Text("Disable / enable Bluetooth"),
  secondary: Icon(Icons.bluetooth, size: 48),
)
```

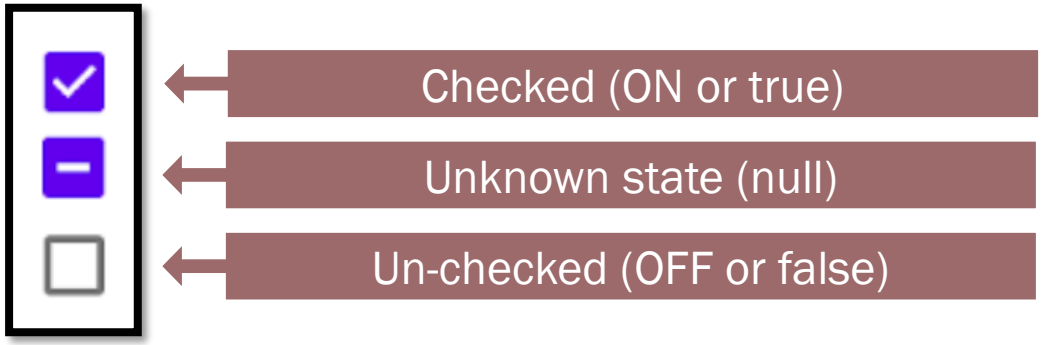
Checkboxes

Checkbox

Checkboxes are UX objects that can have usually two states (checked or unchecked). In some cases, a checkbox has a 3rd state (unknown).

A checkbox is similar to a switch in terms of how it works, interface, etc.

It is recommended to use a checkbox when you are setting multiple options (on or off) that are going to be applied when a button will be pressed (similar to a configuration panel).



One main difference is that these widgets don't have an associated label (a text that explain what that checkbox represents). This means that usually, a Widget like this will be used as part of a Raw widget (so that it can incorporate a Text widget).

Checkbox

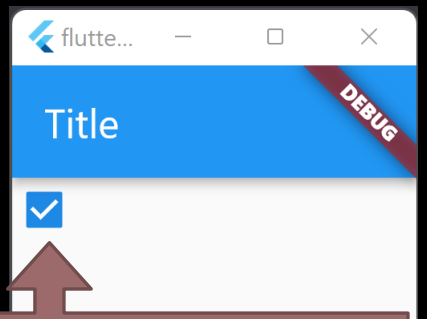
Constructor:

```
Checkbox({
   required bool value,
   required void Function(bool?)? onChanged,
   Color? activeColor,
   Color? checkColor,
   Color? focusColor,
   Color? hoverColor,
   bool tristate = false
   … }
)
```

Checkbox

A very simple example:

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Checkbox(
              value: true,
              onChanged: (b) => {},
            )));
  }
}
```

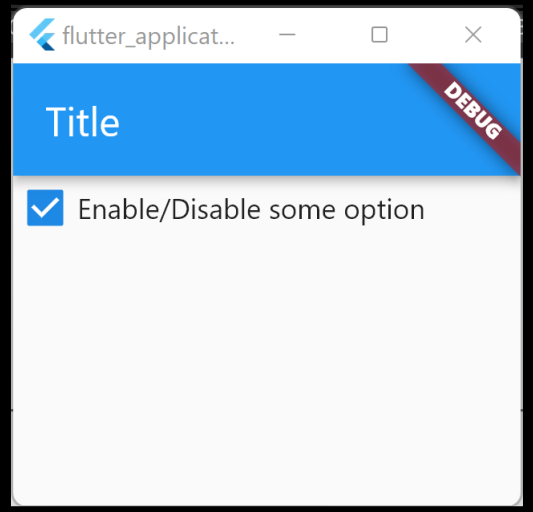


Just like in the case of Switches, there aren't any labels associated with a checkbox

Checkbox

To add a text, a similar technique like in the case of switches can be applied (use a Row with a checkbox and a text). However, the same limitations applies: disabling the checkbox will not disable the text, text can not be clicked, etc.

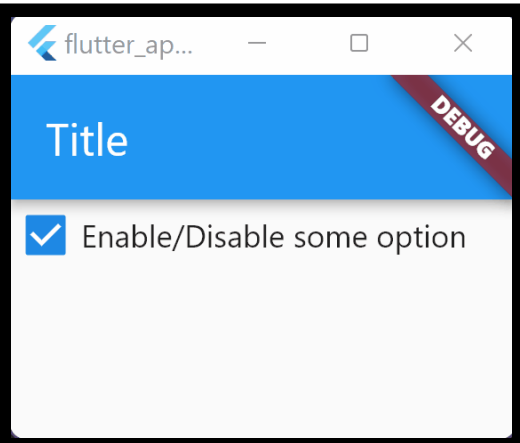
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Checkbox(value: true, onChanged: (b) => {},),
              Text("Enable/Disable some option")
            ])));
  }
}
```



Checkbox

A more complex example:

```
class MyAppState extends State<MyApp> {
  bool option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp( home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Checkbox(
                value: option_1,
                onChanged: (b) => setState(() { option_1 = b ?? false; }),
              ),
              Text("Enable/Disable some option")
            ])));
  }
}
```



Checkbox

A more complex example:

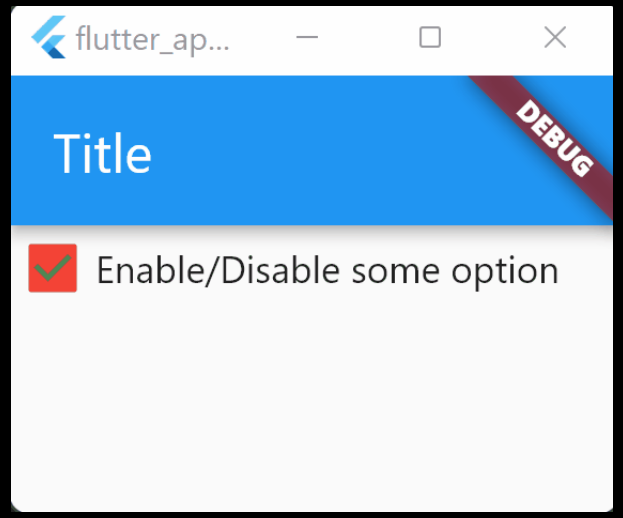
```
class MyAppState extends State<MyApp> {
  bool option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp( home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Checkbox(
                value: option_1,
                onChanged: (b) => setState(() { option_1 = b ?? false; }),
              ),
              Text("Enable/Disable some option")
            ])));
  }
}
```

Another difference from a Switch is that the parameter for onChange is bool? (to reflect a possible tristate).

Checkbox

A more complex example (with colors):

```
class MyAppState extends State<MyApp> {
  bool option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [ Checkbox(
                value: option_1,
                hoverColor: Colors.amber,
                activeColor: Colors.red,
                checkColor: Colors.green,
                onChanged: (b) => setState(() { option_1 = b ?? false; }),
              ), Text("Enable/Disable some option")
            ]))));}}
```



Checkbox

A tri-state example:

```
class MyAppState extends State<MyApp> {
  bool? option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Checkbox(
                value: option_1,
                tristate: true,
                onChanged: (b) => setState(() { option_1 = b;}),
              ), Text("Enable/Disable some option")
            ])));
}}
```

Checkbox

A tri-state example:

```
class MyAppState extends State<MyApp> {
  bool? option_1 = true;
  @override
```

First, the option_1 is no longer a bool, it is now a bool? to reflect all 3 possible values:
[true for checked, false for unchecked and null for unknown]

```
                  itle")),
                body: Row(children: [
                  Checkbox(
                    value: option_1,
                    tristate: true,
                    onChanged: (b) => setState(() { option_1 = b;}),
                  ), Text("Enable/Disable some option")
                ])));
}}
```

Checkbox

A tri-state example:

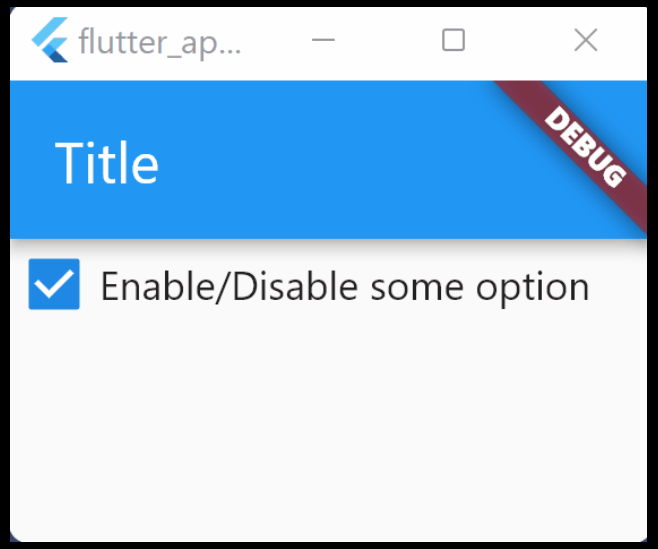
```
class MyAppState extends State<MyApp> {
  bool? option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Checkbox(
                value: option_1,
                tristate: true,
                onChanged: (b) => setState(() { option_1 = b;}),
                                                  option")
            ])));
}}
```

Second, tristate parameter must be set to true.

Checkbox

A tri-state example:

```
class MyAppState extends State<MyApp> {
  bool? option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Checkbox(
                value: option_1,
                tristate: true,
                onChanged: (b) => setState(() { option_1 = b;}),
              ), Text("Enable/Disable some option")
            ])));
}}
```

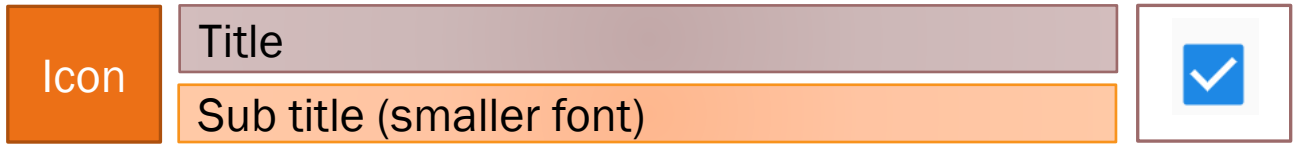


Third, on the onChanged callback just copy the value of b to option_1.

CheckboxListTile

Similar to a switch where there is a special widget called SwitchListTile, there is a similar form for a Checkbox as well (called CheckboxListTile) that has the same properties:

- You can click on the entire space, and this action will trigger an a check or uncheck for the object
- There is a title (and a subtitle)
- There is an icon



CheckboxListTile

Constructor

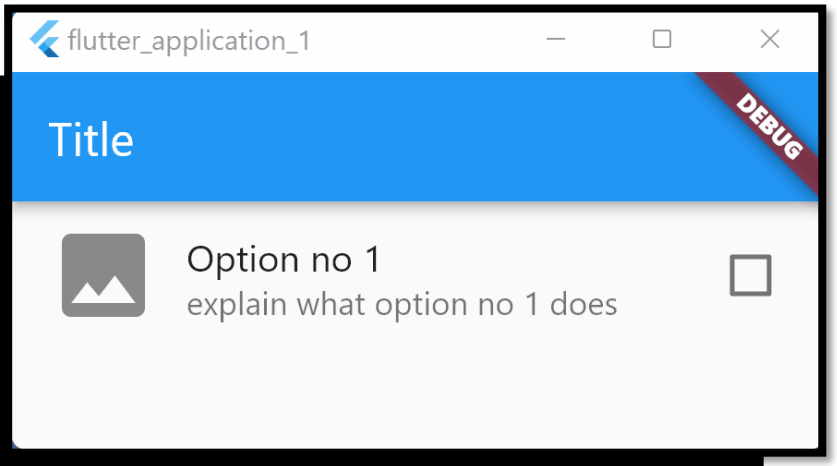
```
CheckboxListTile({
   required bool value,
   required void ValueChanged<bool?>? onChanged,
   Widget? title,
   Widget? subtitle,
   Widget? secondary,
   Color? activeColor,
   Color? tileColor,
   Color? checkColor,
   Color? selectedTileColor,
   bool tristate = false
   … }
)
```

Title label
Subtitle text
Icon

CheckboxListTile

A simple example

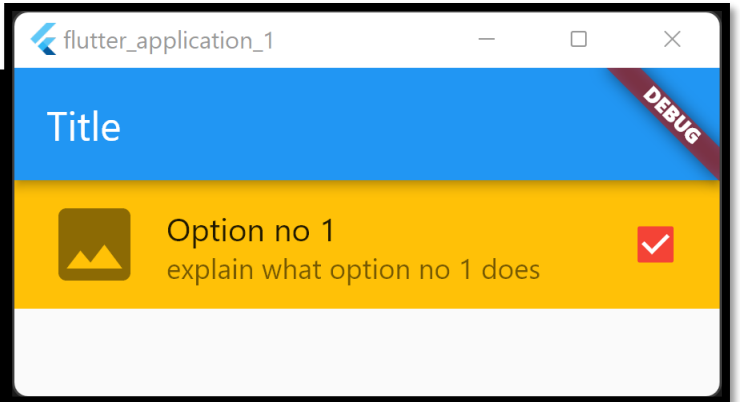
```
class MyAppState extends State<MyApp> {
  bool? option_1 = true;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: CheckboxListTile(
                title: Text("Option no 1"),
                subtitle: Text("explain what option no 1 does"),
                secondary: Icon(Icons.photo, size: 48),
                onChanged: (b) => setState(() { option_1 = b; }),
                value: option_1)));
  }
}
```



CheckboxListTile

A simple example with colors

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Title")),
          body: CheckboxListTile(
              title: Text("Option no 1"),
              subtitle: Text("explain what option no 1 does"),
              secondary: Icon(Icons.photo, size: 48),
              tileColor: Colors.amber,
              activeColor: Colors.red,
              onChanged: (b) => setState(() { option_1 = b; }),
              value: option_1)));
}
```

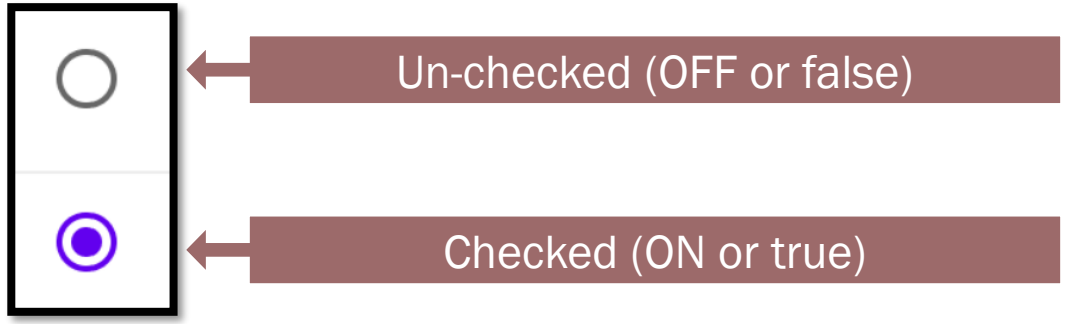


Radioboxes

Radioboxes

Radioboxes are UI widgets that can have only one of them being checked at some moment of time. Radioboxes use a groupValue to compare their value with the one from the group. The one that has the same value as the one from the **groupValue** will be checked, the rest will not be.

Flutter has one widget (called Radio) that is defined as a template/generic. This is very helpful when dealing with enum values.



One main difference is that these widgets don't have an associated label (a text that explain what that radiobox represents). This means that usually, a Widget like this will be used as part of a Raw widget (so that it can incorporate a Text widget).

Radio

Constructor

```
Radio<T>({
   required T value,
   required T? groupValue,
   required void ValueChanged<T?>? onChanged,
   Color? activeColor,
   Color? focusColor,
   Color? hoverColor,
   Color? overlayColor,
   … }
)
```

Radio

A regular usage for a Radio will be as follows

```
class MyAppState extends State<MyApp> {
  String s = "2";

}
```

First let's create a local variable that will be the groupValue for several Radio widgets.

In out case it will be a string variable with three possible values: "1", "2" and "3"

Radio

A regular usage for a Radio will be as follows

```
class MyAppState extends State<MyApp> {
  String s = "2";

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Radio<String>(…),
              Radio<String>(…),
              Radio<String>(…)
            ])));
  }
}
```

Second, we will create a Row widget with 3 Radio widgets.

Notice that each Radio widget is a template of type String

Radio

A regular usage for a Radio will be as follows

```
class MyAppState extends State<MyApp> {
  String s = "2";

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: T
            body: Row(children: [
              Radio<String>(…),
              Radio<String>(…),
              Radio<String>(…)
            ])));
  }
}
```

```
Radio<String>(
  value: "1",
  groupValue: s,
  onChanged: (b) => setState(() { s = "1"; })
)
```

Radio

A regular usage for a Radio will be as follows

```
class MyAppState extends State<MyApp> {
  String s = "2";

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: T
            body: Row(children: [
              Radio<String>(…),
              Radio<String>(…),
              Radio<String>(
            ])));
  }
}
```

```
Radio<String>(
  value: "1",
  groupValue: s,
  onChanged: (b) => setState(() { s = "1"; })
)
```

This translates in *when string variable s has the value "1", this Radio will be checked.*

Radio

A regular usage for a Radio will be as follows

```
class MyAppState extends State<MyApp> {
  String s = "2";

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: T
            body: Row(children: [
              Radio<String>(…),
              Radio<String>(…),
              Radio<String>(…)
            ])));
  }
}
```

```
Radio<String>(
  value: "1",
  groupValue: s,
  onChanged: (b) => setState(() { s = "1"; })
)
```

When click, change the value of variable "s" to "1" so that this Radio will be checked.

Alternatively, one can also write { s = b; }

Radio

A regular usage for a Radio will be as follows

```
class MyAppState extends State<MyApp> {
  String s = "2";

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Radio<String>(…),
              Radio<String>(…),
              Radio<String>(…)
            ])));
  }
}
```

```
Radio<String>(
  value: "2"
  groupValue: s,
  onChanged: (b) => setState(() { s = "2"; })
)
```

Radio

A regular usage for a Radio will be as follows

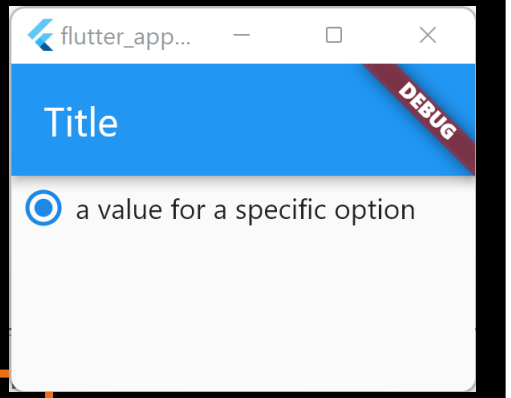
```
class MyAppState extends State<MyApp> {
  String s = "2";

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Radio<String>(…),
              Radio<String>(…),
              Radio<String>(…)
            ])));
  }
}
```


```
Radio<String>(
  value: "3"
  groupValue: s,
  onChanged: (b) => setState(() { s = "3"; })
)
```

Radio

One way of adding a label to a radio box is to put it in a Row widget and add a Text after it.

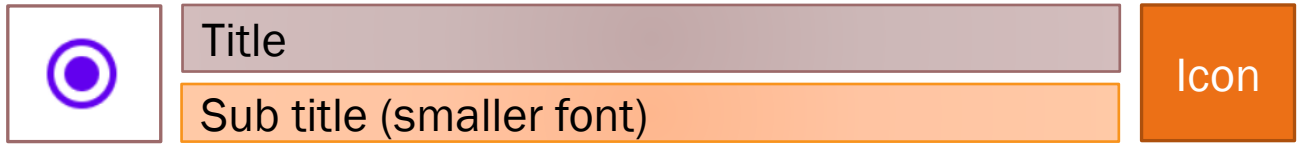
```
class MyAppState extends State<MyApp> {
  String s = "2";
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Row(children: [
              Radio<String>(
                  value: "1",
                  groupValue: s,
                  onChanged: (b) => setState(() { s = "1"; })),
              Text("a value for a specific option")
            ])));  }
}
```



RadioListTile

The other option is to use a RadioListTile (similar to the one from Checkbox and Switch). This widget has the same parameters as with the CheckboxListTile or SwitchListTile:

- A title
- A subtitle
- A secondary parameter that usually serves as an icon



Another observation here is that by default, the radio button is placed on the left side (as different from the checkbox that by default is position on the right side).

RadioListTile

Constructor

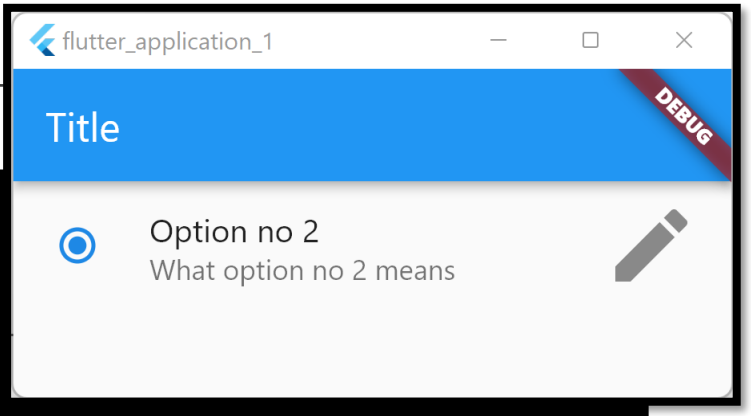
```
RadioListTile<T>({
   required T value,
   required T? groupValue,
   required void ValueChanged<T?>? onChanged,
   Widget? title,
   Widget? subtitle,
   Widget? secondary,
   Color? activeColor,
   Color? tileColor,
   Color? selectedTileColor,
   … }
)
```

Title label
Subtitle text
Icon

RadioListTile

A very simple example:

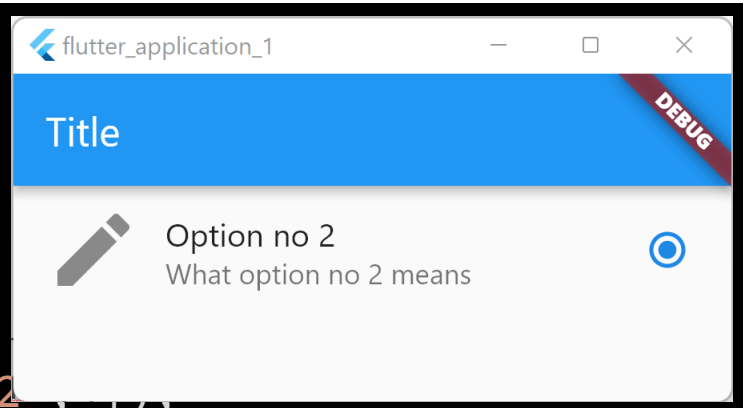
```
class MyAppState extends State<MyApp> {
  String s = "2";
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold( appBar: AppBar(title: Text("Title")),
            body: RadioListTile<String>(
              value: "2",
              onChanged: (b) => setState(() { s = "2"; }),
              groupValue: s,
              title: Text("Option no 2"),
              subtitle: Text("What option no 2 means"),
              secondary: Icon(Icons.edit, size: 48),
            )));
} }
```



RadioListTile

We can change the position of the Radio button by using controlAffinity parameter.

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Title")),
          body: RadioListTile<String>(
            value: "2",
            onChanged: (b) => setState(() { s = "2" ; });
            groupValue: s,
            title: Text("Option no 2"),
            subtitle: Text("What option no 2 means"),
            secondary: Icon(Icons.edit, size: 48),
            controlAffinity: ListTileControlAffinity.trailing,
          )));
}
```



RadioListTile

Let's see how we can use a RadioListTile with an enum

```
enum Transport { Bike, Taxi, Airplane }

class MyAppState extends State
  Transport? transport;

}
```

Let's consider Transport enum with 3 values.

We will also create a local variable of Transport type

RadioListTile

Let's see how we can use a RadioListTile with an enum

```
enum Transport { Bike, Taxi, Airplane }

class MyAppState extends State<MyApp> {
  Transport? transport;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(appBar: AppBar(title:
            body: Column(children: [
              RadioListTile<Transport>(…),
              RadioListTile<Transport>(…),
              RadioListTile<Transport>(…)
            ])));
  }
}
```

```
RadioListTile<Transport> (
  value: Transport.Bike,
  onChanged: (b) => setState(() {
                  transport = Transport.Bike;
                }),
  groupValue: transport,
  title: Text("Use a bike"),
  subtitle: Text("More exercise for you"),
  secondary: Icon(Icons.bike_scooter, size: 48),
  controlAffinity: ListTileControlAffinity.trailing,
)
```

RadioListTile

Let's see how we can use a RadioListTile with an enum

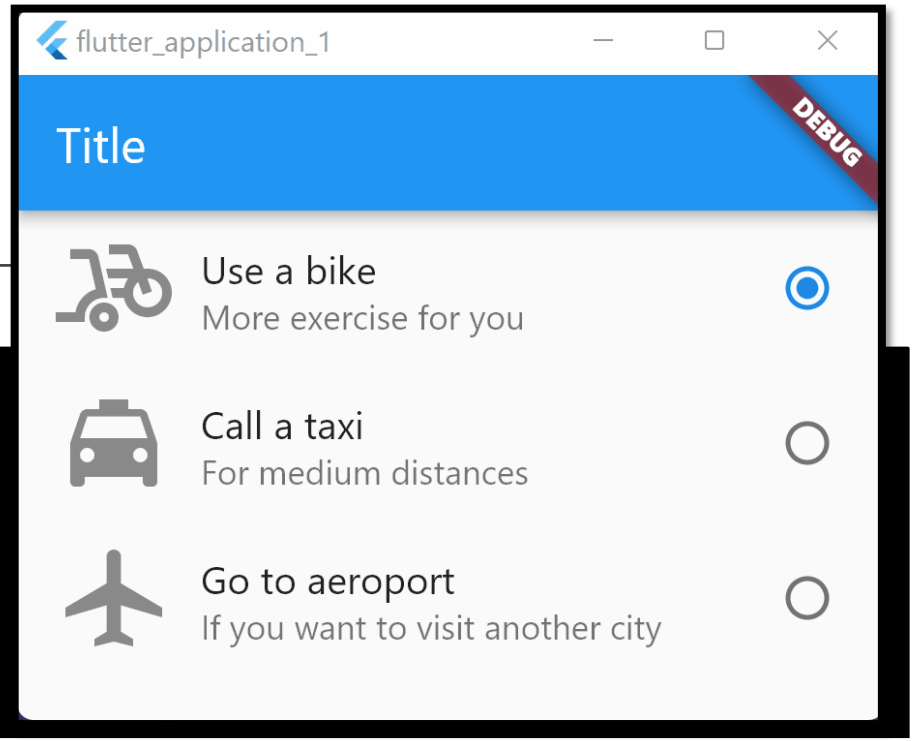
```
enum Transport { Bike, Taxi, Airplane }

class MyAppState extends State<MyApp> {
  Transport? transport;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(appBar: AppBar(title:
            body: Column(children: [
              RadioListTile<Transport>(…),
              RadioListTile<Transport>(…),
              RadioListTile<Transport>(…)
            ])));
  }
}
```

```
RadioListTile<Transport> (
  value: Transport.Taxi,
  onChanged: (b) => setState(() {
                  transport = Transport.Taxi;
                }),
  groupValue: transport,
  title: Text("Call a taxi"),
  subtitle: Text("For medium distances"),
  secondary: Icon(Icons.local_taxi, size: 48),
  controlAffinity: ListTileControlAffinity.trailing,
)
```

RadioListTile

Let's see how we can use a RadioListTile with an enum

```
enum Transport { Bike, Taxi, Airplane }

class MyAppState extends State<MyApp> {
  Transport? transport;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(appBar: AppBar(title:
            body: Column(children: [
              RadioListTile<Transport>(…),
              RadioListTile<Transport>(…),
              RadioListTile<Transport>(…)
            ])));
  }
}
```




```
RadioListTile<Transport> (
  value: Transport.Airplane,
  onChanged: (b) => setState(() {
                  transport = Transport.Airplane;
                }),
  groupValue: transport,
  title: Text("Go to aeroport"),
  subtitle: Text("If you want to visit another city"),
  secondary: Icon(Icons.airplanemode_active,
                  size: 48),
  controlAffinity: ListTileControlAffinity.trailing,
)
```

RadioListTile

Make sure that you put different values for all RadioListTiles in a group ! If you don't you might get to check on and end up having 2 or more selected !

```
class MyAppState extends State<MyApp> {
  int opt = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("
            body: Column(children: [
              RadioListTile<int>(…),
              RadioListTile<int>(…),
              RadioListTile<int>(…),
            ])));
  }
}
```

```
RadioListTile<int>(
    value: 1,
    onChanged: (b) => setState(() { opt = 1; }),
    groupValue: opt,
    title: Text("Option 1"),
)
```

RadioListTile

Make sure that you put different values for all RadioListTiles in a group ! If you don't you might get to check on and end up having 2 or more selected !

```
class MyAppState extends State<MyApp> {
  int opt = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text(…
            body: Column(children: [
              RadioListTile<int>(…),
              RadioListTile<int>(…),
              RadioListTile<int>(…),
            ])));
  }
}
```

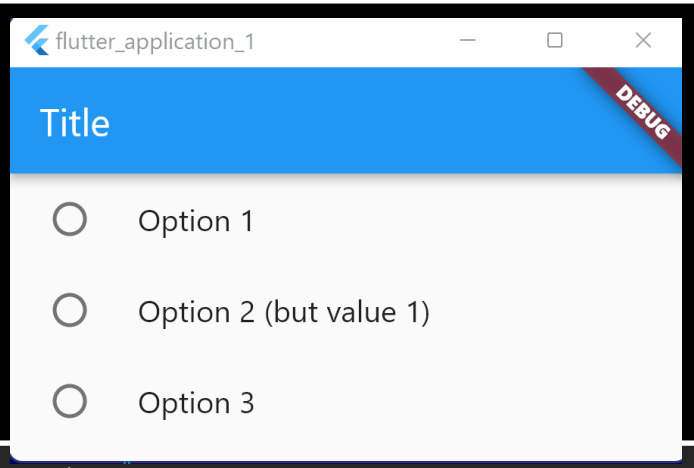
The correct value should have been 2 and not 1

```
RadioListTile<int>(
    value: 1,
    onChanged: (b) => setState(() { opt = 2; }),
    groupValue: opt,
    title: Text("Option 2 (but value 1)"),
)
```

RadioListTile

Make sure that you put different values for all RadioListTiles in a group ! If you don't you might get to check on and end up having 2 or more selected !

```
class MyAppState extends State<MyApp> {
  int opt = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Column(children: [
              RadioListTile<int>(…),
              RadioListTile<int>(…),
              RadioListTile<int>(…),
            ])));
  }
}
```




```
RadioListTile<int>(
    value: 3,
    onChanged: (b) => setState(() { opt = 3; }),
    groupValue: opt,
    title: Text("Option 3"),
)
```

ListTile

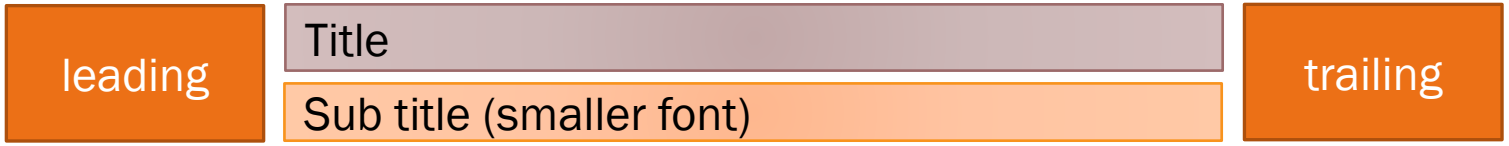
ListTile

A ListTile is a widget that can act like a button (an item) in a list of items.

It is in particular useful if you want to create a list of elements that contain several actions. List tiles (as a concept) is used with another widgets such as: Switch, Checkbox, Radiobox.

As a general format, a list tile is form out of:

- A leading widget
- A trailing widger
- A title
- A sub-title



ListTile

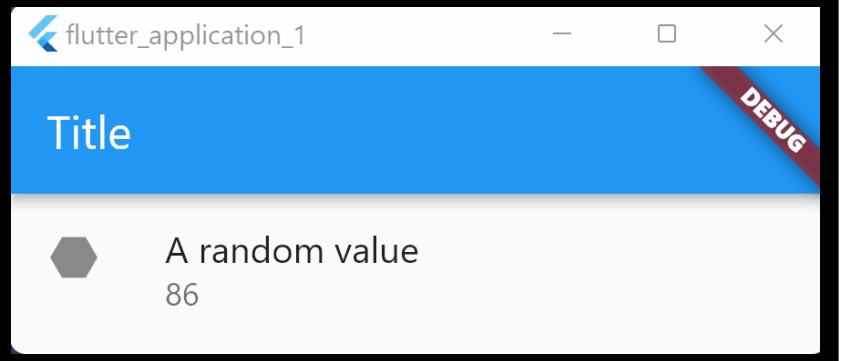
Constructor

```
ListTile ({
   GestureTapCallback? onTap,
   GestureLongPressCallback? onLongPress,
   Widget? leading,
   Widget? title,
   Widget? subtitle,
   Widget? trailing,
   Color? iconColor,
   Color? tileColor,
   Color? selectedTileColor,
   Color? selectedColor,
   Color? hoverColor,
   … }
)
```

ListTile

A simple example for Random numbers generation

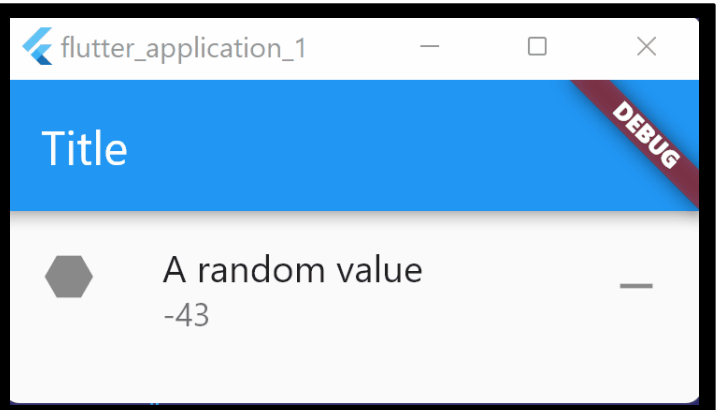
```
class MyAppState extends State<MyApp> {
  int rand = Random().nextInt(100);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: ListTile(
              title: Text("A random value"),
              subtitle: Text("$rand"),
              onTap: () => setState(() { rand = Random().nextInt(100);}),
              leading: Icon(Icons.hexagon),
            )));
  }
}
```



ListTile

Let's build a slightly more complex example (one that generates a number between -50 and +49 and shows an icon with a plus (if the number is positive) or a minus if the number is negative.

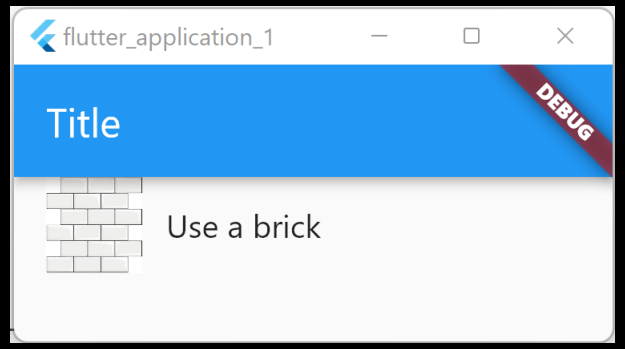
```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Title")),
          body: ListTile(
            title: Text("A random value"),
            subtitle: Text("$rand"),
            onTap: () => setState(() { rand = Random().nextInt(100) - 50; }),
            leading: Icon(Icons.hexagon),
            trailing: Icon(rand >= 0 ? Icons.add : Icons.remove),
            hoverColor: Colors.amber,
          )));
}
```



ListTile

Images can also be used (instead of icons). The next example assumes that tiles.jpg is located in "assets/images" and it is also added in pubspec.yaml file.

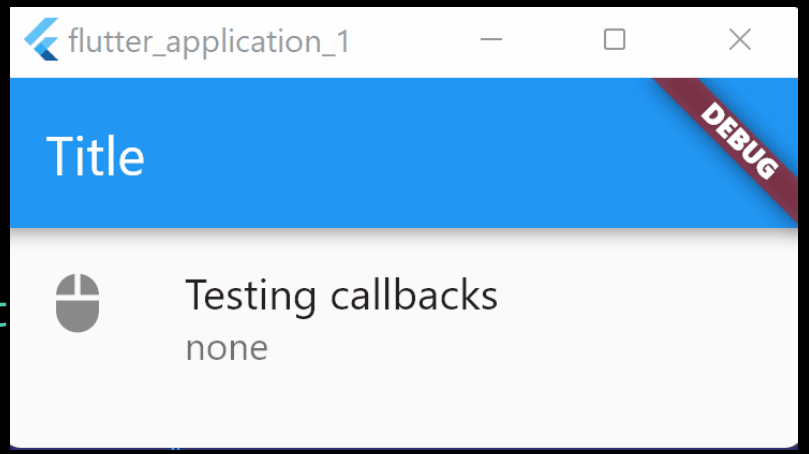
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: ListTile(
              title: Text("Use a brick"),
              onTap: () => {},
              leading: Image.asset("assets/images/tiles.jpg"),
              hoverColor: Colors.amber,
            )));
   }
}
```



ListTile

A simple example to test onTap and onLongPress callbacks.

```
class MyAppState extends State<MyApp> {
  String msg = "none";
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold( appBar: AppBar(title: Text
            body: ListTile(
              title: Text("Testing callbacks"),
              subtitle: Text(msg),
              onTap: () => setState(() { msg = "onTap"; }),
              onLongPress: () => setState(() { msg = "onLongPress"; }),
              leading: Icon(Icons.mouse),
              hoverColor: Colors.lightBlue,
            )));
} }
```



Q & A