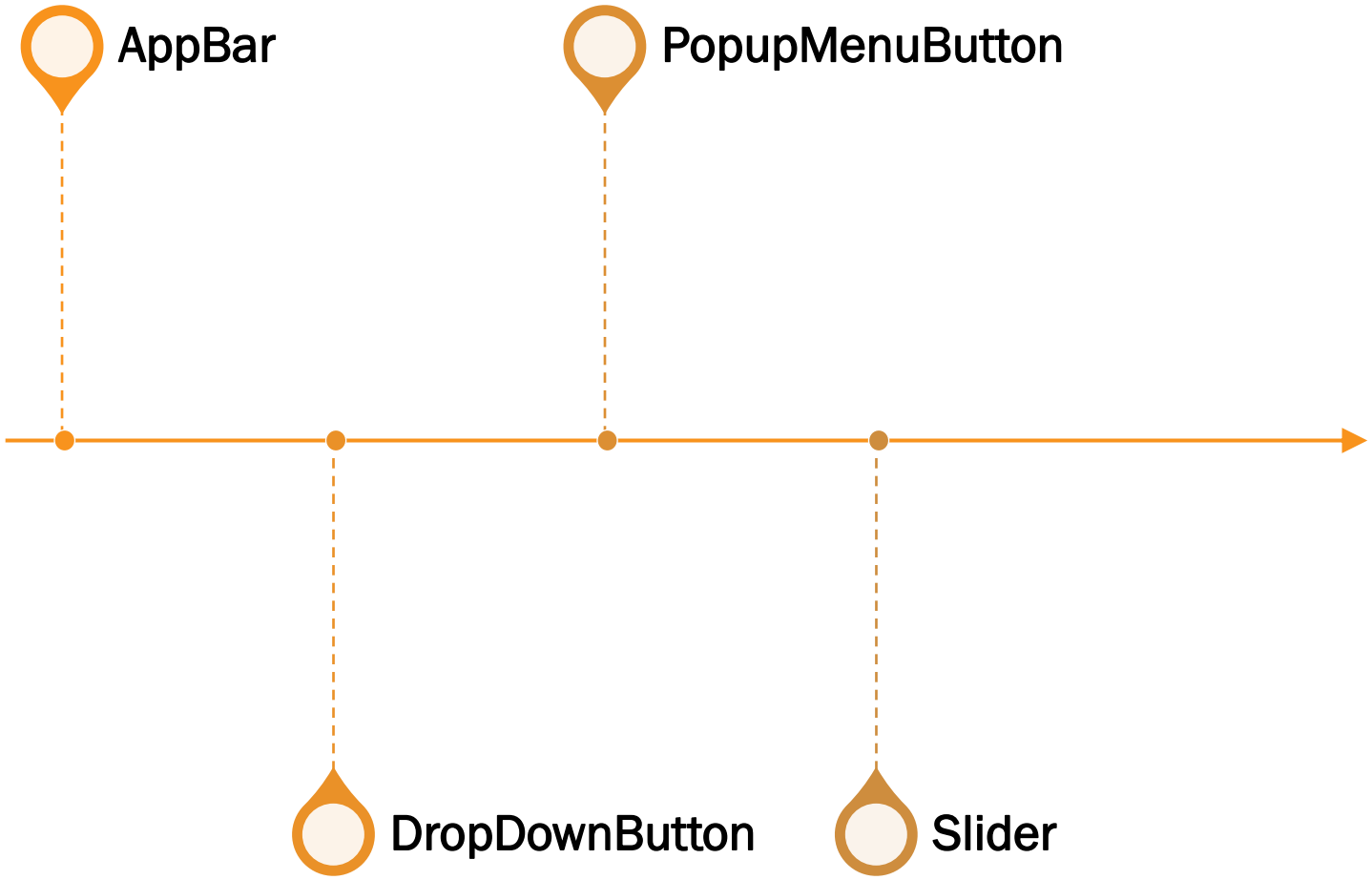
Flutter Framework

COURSE 9 (REV 6)

GAVRILUT DRAGOS

Agenda

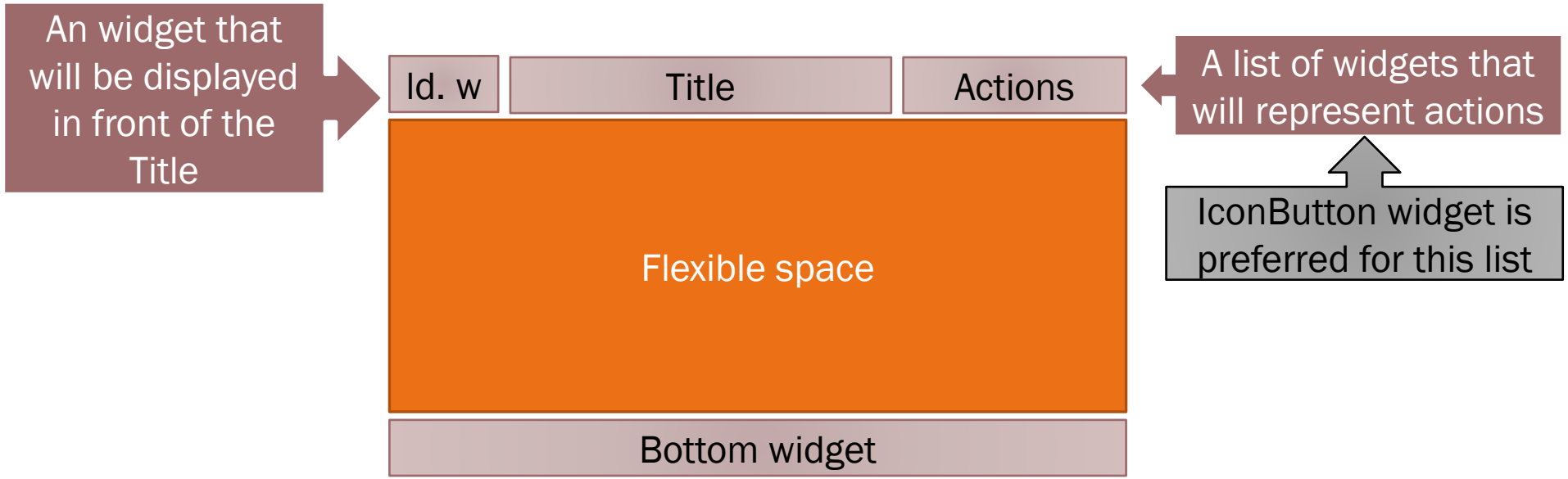


AppBar

AppBar

AppBar component is the main component of a Scaffold (and it holds a form of organization for the current view).

An AppBar has some components:



AppBar

Constructor:

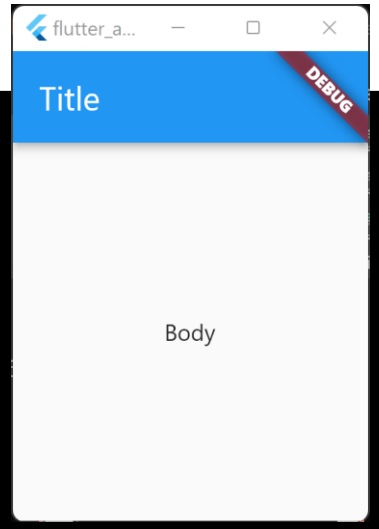
```
AppBar({
   Widget? leading,
   Widget? title,
   Widget? flexibleSpace,
   PreferredSizeWidget? Bottom,
   List<Widget>? actions,
   Color? forengroundColor,
   Color? backgroundColor,
   … }
)
```

```
The leading widget
Title
The flexible space widget
The buttom widget
A list of widgets for actions
```

AppBar

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: Text("Body"))));
  }
}
```

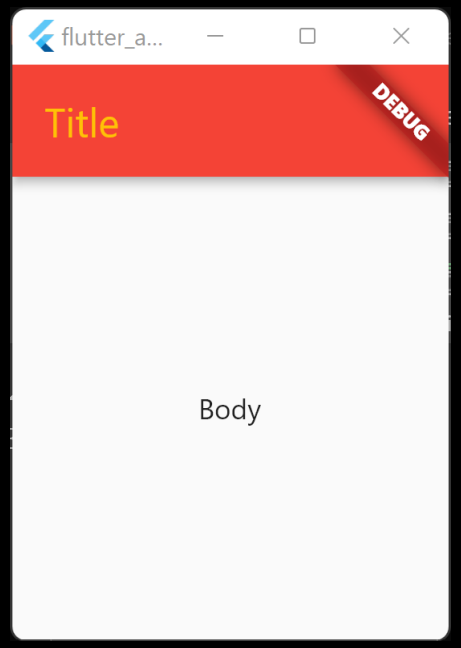


For the next part of this course, we will focus on the build method and different parameters that can be set up for the AppBar

AppBar

To set up the background and title text color use the .forengroundColor and .backgroundColor properties.

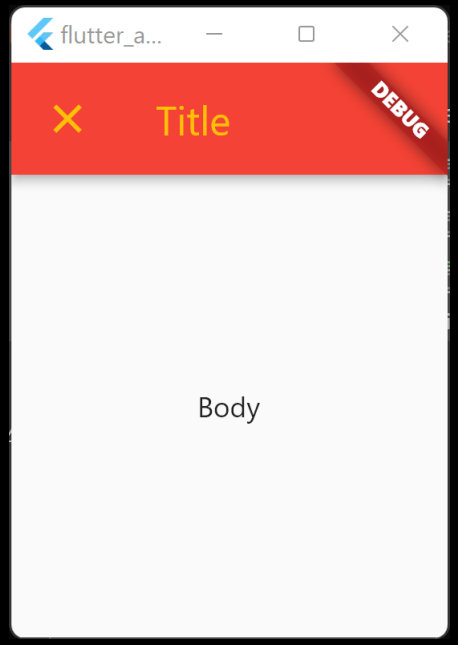
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(
              title: Text("Title"),
              foregroundColor: Colors.amber,
              backgroundColor: Colors.red,
            ),
            body: Center(child: Text("Body"))));
  }
}
```



AppBar

The “leading” widget is usually a close button (however, it can be any widget).

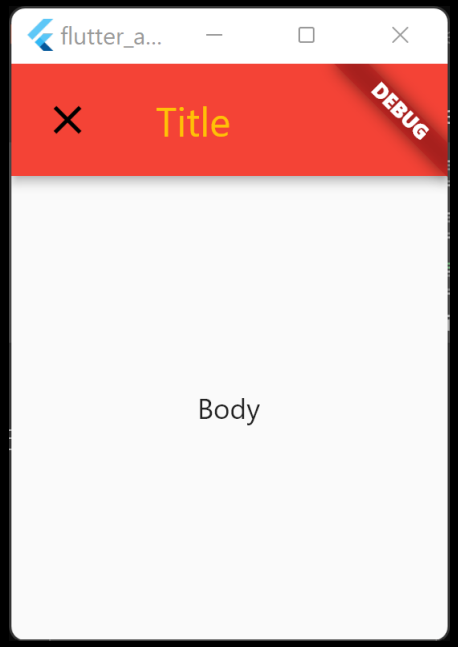
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(
              title: Text("Title"),
              foregroundColor: Colors.amber,
              backgroundColor: Colors.red,
              leading: CloseButton(),
            ),
            body: Center(child: Text("Body"))));
  }
}
```



AppBar

The “leading” widget is usually a close button (however, it can be any widget).

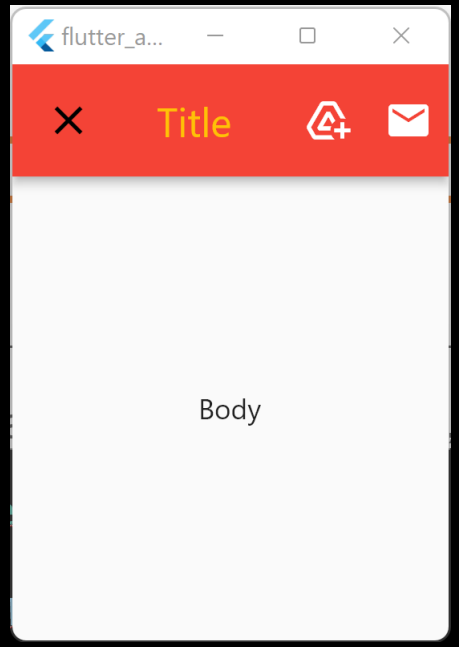
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(
              title: Text("Title"),
              foregroundColor: Colors.amber,
              backgroundColor: Colors.red,
              leading: CloseButton(color: Colors.black),
            ),
            body: Center(child: Text("Body"))));
  }
}
```



AppBar

The “leading” widget is usually a close button (however, it can be any widget).

```
Widget build(BuildContext context) {
  return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
          appBar: AppBar(
            title: Text("Title"),
            foregroundColor: Colors.amber,
            backgroundColor: Colors.red,
            leading: CloseButton(color: Colors.black),
            actions: [ IconButton(…), IconButton(…) ],
          ),
          body: Center(child: Text("Body"))));
}
```



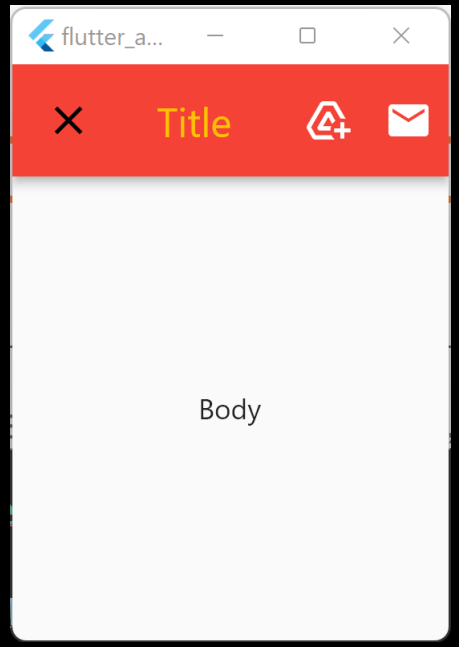
AppBar

The “leading” widget is usually a close button (however, it can be any widget).

```
Widget build(BuildContext context) {
  return MaterialApp(
      debu
      home
```

```
IconButton(
    onPressed: () => {},
    icon: Icon(Icons.add_to_drive),
    color: Colors.white,
)
```

```
          backgroundColor: Colors.red,
          leading: CloseButton(color: Colors.black),
          actions: [ IconButton(…), IconButton(…) ],
        ),
        body: Center(child: Text("Body"))));
}
```

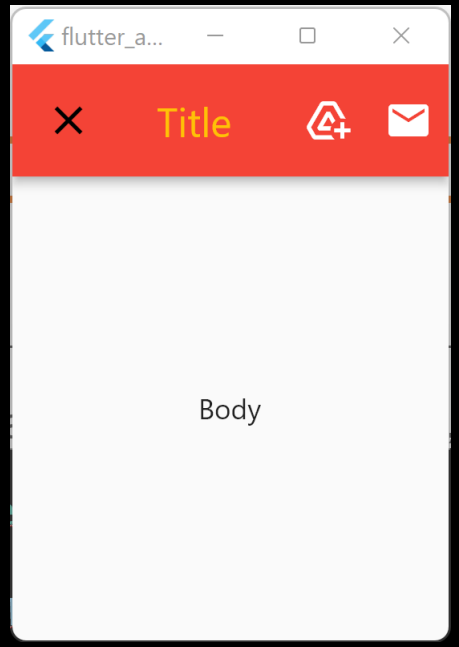


AppBar

The “leading” widget is usually a close button (however, it can be any widget).

```
Widget build(BuildContext context) {
  return MaterialApp(
      debugShowCheckedModeBanne
      home: Scaffold(
          appBar: AppBar(
            title: Text("Title"
            foregroundColor: Co
            backgroundColor: Colors.red,
            leading: CloseButton(color: Colors.black),
            actions: [ IconButton(…), IconButton(…) ],
          ),
          body: Center(child: Text("Body"))));
}
```

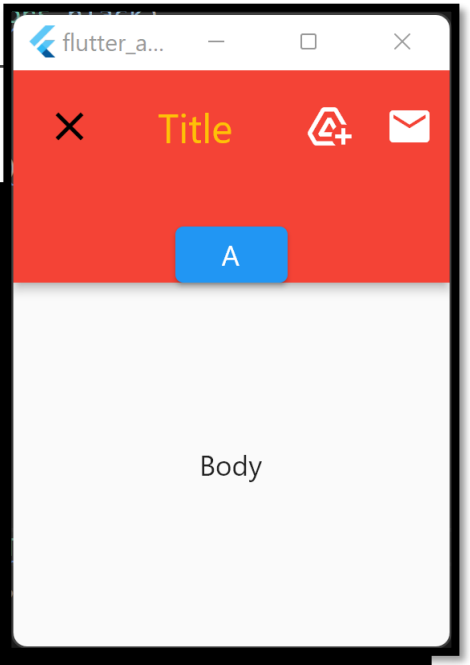
```
IconButton(
    onPressed: () => {},
    icon: Icon(Icons.email),
    color: Colors.white,
)
```



AppBar

The “bottom” widget can be used to increase the size of the AppBar

```
Widget build(BuildContext context) {
  return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(appBar: AppBar(
            title: Text("Title"),
            foregroundColor: Colors.amber,
            backgroundColor: Colors.red,
            leading: CloseButton(color: Colors.black),
            actions: [ IconButton(…), IconButton(…) ],
            bottom: PreferredSize(
                    child: ElevatedButton(child: Text("A"), onPressed:()=>{}),
                    preferredSize: Size(300, 50)),
          ),
          body: Center(child: Text("Body"))));}
```



DropdownButton

DropdownButton

A dropdown button (or combobox) is a widget that allows you to choose from several existing options.

Just like in the case of Radio buttons, a dropdown button is based on a template that reflects the selection. Because of this, it is very useful for enums.

There are two classes relevant for this widget:

1. DropdownButton (the actual button)
2. DropdownMenuItem (the actual item, also a template that holds the value and the widget that reflects that value)

DropdownButton

Constructors:

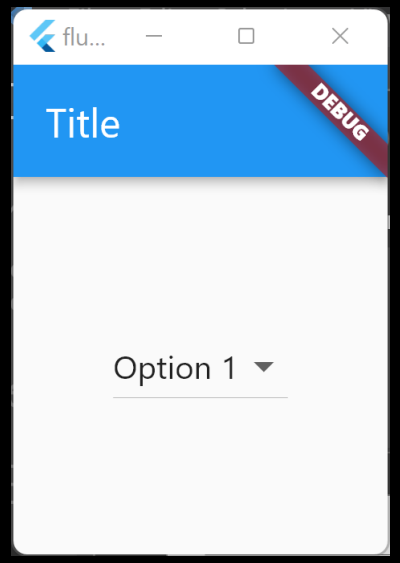
```
DropdownButton<T>({
    Widget? hint,
    Widget? disabledHint,
    Widget? underline,
    Widget? icon,
    double iconSize = 24.0,
    T? value,
    required List<DropdownMenuItem<T>>? items,
    required void Function(T?)? onChanged,
    Color? iconDisabledColor,
    Color? iconEnabledColor,
    Color? dropdownColor,
    Color? focusColor,
    … } )
```

```
DropdownMenuItem<T>({
    required Widget child,
    T? value,
    bool enabled = true,
    AlignamentGeometry alignament = ,
            AlignamentGeometry.centerStart
    … }
)
```

DropdownButton

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: DropdownButton<int>(…))));
  }
}
```



DropdownButton

A very simple example.

```
import 'package:flutter
void main() => runApp(M
class MyApp extends Sta
  @override
  State<MyApp> createSt
}
class MyAppState extend
  @override
  Widget build(BuildCon
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(child: DropdownButton<int>(…))));
  }
}
```

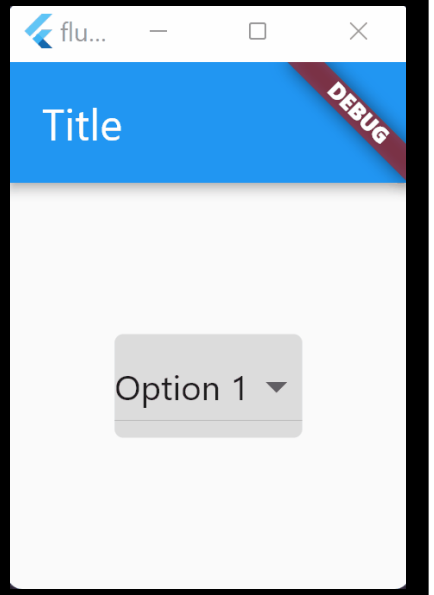
```
DropdownButton<int>(
  items: [
    DropdownMenuItem(value: 1, child: Text("Option 1")),
    DropdownMenuItem(value: 2, child: Text("Option 2")),
    DropdownMenuItem(value: 3, child: Text("Option 3"))
  ],
  value: 1,
  onChanged: (item) => {},
)
```

DropdownButton

To make a drop-down button work, the following steps must be followed:

1. Create a variable that will store the selected value (in our care it variable v of type int)
2. Make sure that the property value from DropdownButton ctor is set to variable v
3. In onChange callback use setState to copy the selected value into variable v

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: DropdownButton<int>(…),
            ))));
} }
```



DropdownButton

To make a drop-down button work, the following steps must be followed:

1. Create a va
2. Make sure t
3. In onChang

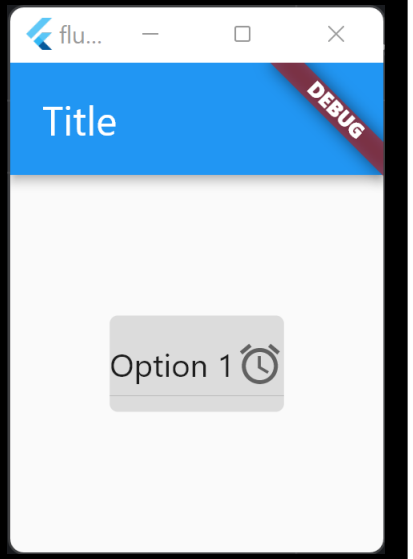
```
DropdownButton<int>(
  items: [
    DropdownMenuItem(value: 1, child: Text("Option 1")),
    DropdownMenuItem(value: 2, child: Text("Option 2")),
    DropdownMenuItem(value: 3, child: Text("Option 3"))
  ],
  value: v,
  onChanged: (item) => setState(() { v = item; }
)
```

```
class MyAppSta
  int? v = 1;
  @override
  Widget build
    return Mat
        home: Scaffold(
            appBar: AppBar(title: Text("
            body: Center(
                child: DropdownButton<int>(…),
            ))));
} }
```

DropdownButton

To add an icon to the dropdown use the icon property.

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: DropdownButton<int>(
                    items: […],
                    value: v,
                    onChanged: (item) => setState(() { v = item; }),
                    icon: Icon(Icons.access_alarm)))));
  }
}
```



DropdownButton

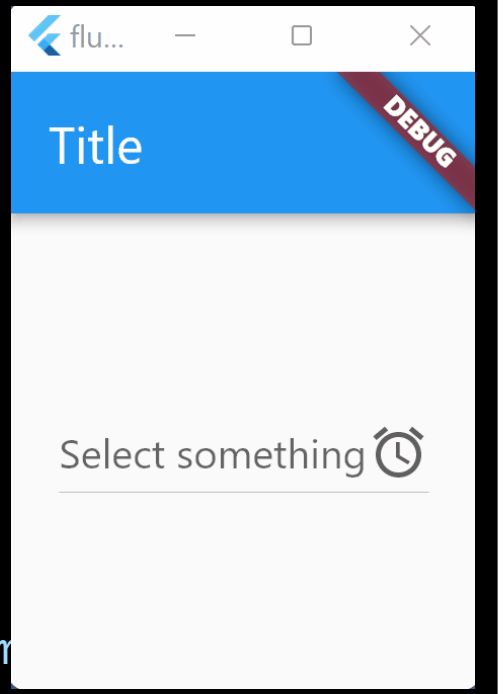
You can also set the enable and disable color for the icon:

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: DropdownButton<int>(
                    items: […],
                    value: v,
                    onChanged: (item) => setState(() { v = item; }),
                    icon: Icon(Icons.access_alarm),
                    iconEnabledColor: Colors.red))));
} }
```

DropdownButton

If value is null, a the widget is **enabled**, the widget from hint is used (if exists).

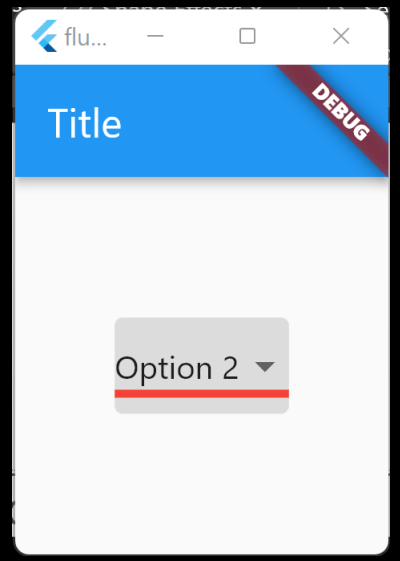
```
class MyAppState extends State<MyApp> {
  int? v = null;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: DropdownButton<int>(
                    items: […],
                    value: v,
                    onChanged: (item) => setState(() { v = item
                    hint: Text("Select something"),
                    icon: Icon(Icons.access_alarm)))));
} }
```



DropdownButton

You can also use underline property to draw something under the widget

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: DropdownButton<int>(
                    items: […],
                    value: v,
                    onChanged: (item) => setState(() { v = item; }),
                    underline: Container(height: 4, color: Colors.red),
                    ))));
} }
```



DropdownButton

But what if we want to create something more complex.

For example, a dropdown menu that has a different icon for each item (and maybe different colors, or different picture, etc).

In this case, the child property for the DropdownMenuItem has to be a complex widget.

DropdownButton

So … let's see a code that will create such a widget.

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  DropdownMenuItem<int> CreateMenuItem(String text, int val, IconData iconData) {…}
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: DropdownButton<int>(
                   items: […],
                   value: v,
                   onChanged: (item) => setState(() { v = item; }))))));
  }
}
```

DropdownButton

So … let's see a code that will create such a widget.

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  DropdownMenuItem<int> CreateMenuItem(String text, int val, IconData iconData) {…}
  @override
  Widget build(BuildContext context)
    return M
        home
                                                                  ; }))))));
  }
}
```

```
return DropdownMenuItem<int>(
        value: val,
        child: Row(
                children: [
                        Icon(iconData),
                        SizedBox(width: 10),
                        Text(text)
                ]
        )
);
```

DropdownButton

So … let's see a code that will create such a widget.

```
class MyAppState extends State<MyApp> {
  int? v = 1;
  DropdownMenuItem<int> CreateMenuItem(String text, int val, IconData iconData) {…}
```

```
items: [
   CreateMenuItem("Start", 1, Icons.start),
   CreateMenuItem("Fast forward", 2, Icons.fast_forward),
   CreateMenuItem("Stop", 3, Icons.stop),
],
```

```
            body: Center
                child: DropdownButton<int>(
                  items: […],
                  value: v,
                  onChanged: (item) => setState(() { v = item; }))))));
  }
}
```

PopupMenuButton

PopupMenuButton

A popup menu button is similar to a DropdownButton, but it has no selection (meaning that after you click on an item, you will just received a callback to notify you that a specific item has been click, but no change in the interface on how that button looks like).

Its main face can be represented by an icon or a child widget (but not both). If none is provided a default icon that looks like 3 vertical points () will be used.

This means that it is possible to make it looks like a DropdownButton if you change its properties upon building.

PopupMenuButton is often used with the AppBar to show the default menu for that application.

PopupMenuButton

Constructor:

```
PopupMenuButton<T>({
   Widget? child,
   Widget? icon,
   double? iconSize,
   T? initialValue,
   required List<PopupMenuEntry<T>> Function(BuildContext) itemBuilder,
   void Function(T)? onSelected,
   void Function()? onCancel,
   String? toolTip,
   Color? color,
   … }
)
```

PopupMenuButton

Constructor:

```
PopupMenuButton<T>({
   Widget? child,
   Widget? icon,
   double? iconSize,
   T? initialValue,
   required List<PopupMenuEntry<T>> Function(BuildContext) itemBuilder,
   void Function(T)? onSelected,
```

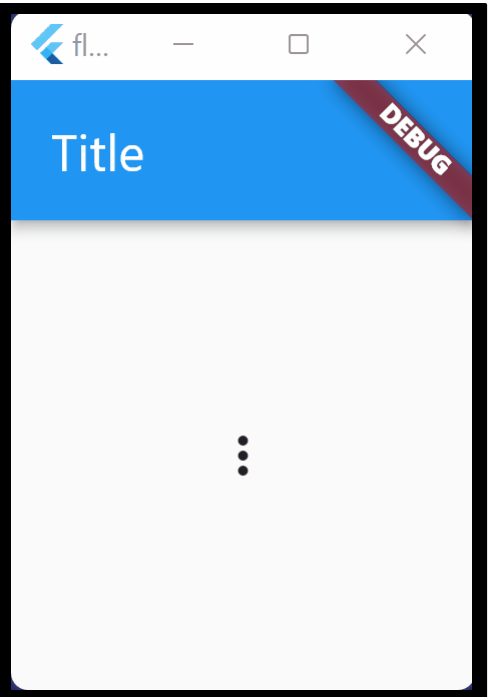
A PopupMenuEntry is a base class for a menu item.
There are several classes that implements this class

- PopupMenuItem → a menu item
- CheckedPopupMenuItem → a menu item with checkmark
- PopupMenuDevider → a vertical line that separates two menu items

PopupMenuButton

So … let's see a code that will create such a widget.

```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => [
        PopupMenuItem<int>(value: 1, child: Text("opt 1")),
        PopupMenuItem<int>(value: 2, child: Text("opt 2")),
        PopupMenuItem<int>(value: 3, child: Text("opt 3"))
      ];
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: PopupMenuButton<int>(itemBuilder: itemBuilderFnc))));
  }
}
```



PopupMenuButton

We can change the default icon by using the icon property:

```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […];
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: PopupMenuButton<int>(
                    itemBuilder: itemBuilderFnc,
                    icon: Icon(Icons.menu))))));
  }
}
```

PopupMenuButton

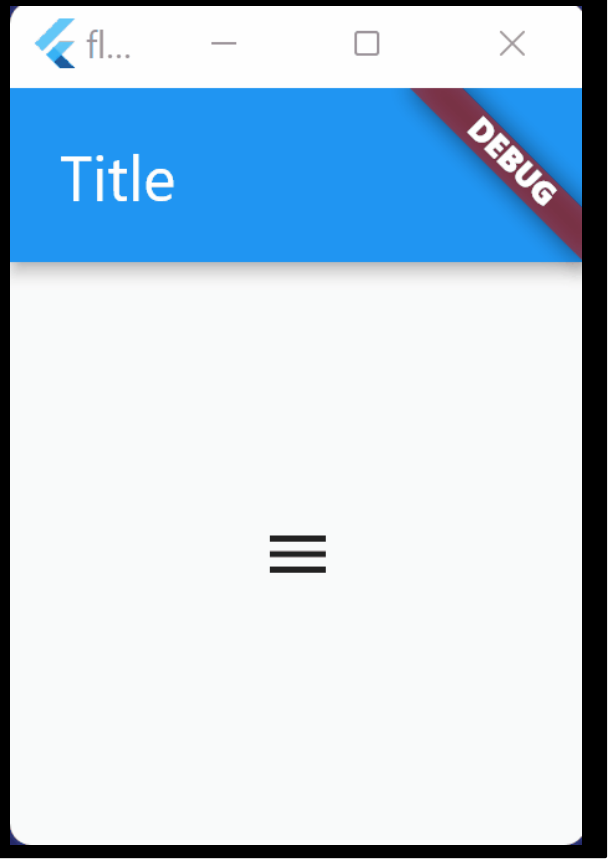
We can also change the color of the menu with the color property:

```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […];
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: PopupMenuButton<int>(
                        itemBuilder: itemBuilderFnc,
                        icon: Icon(Icons.menu),
                        color: Colors.lightBlue,
            ))));
  }
}
```

PopupMenuButton

Or, we ca use the toolTip property to set a new toolTop (the default one is Show menu):

```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […];
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: PopupMenuButton<int>(
                        itemBuilder: itemBuilderFnc,
                        icon: Icon(Icons.menu),
                        tooltip: "Press to see a menu",
            ))));
  }
}
```



PopupMenuButton

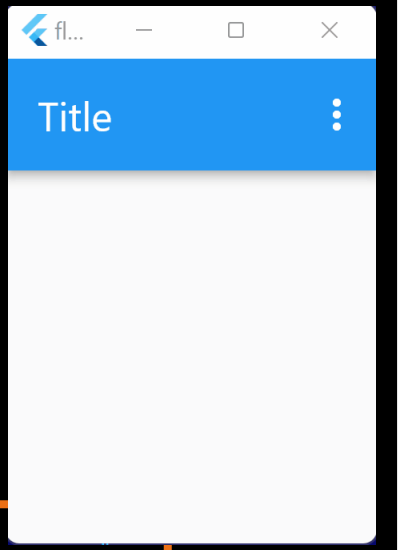
You can also use the **child** property if you want to set up a text or something different than an icon

```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […];
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Title")),
            body: Center(
                child: PopupMenuButton<int>(
                        itemBuilder: itemBuilderFnc,
                        child: Text("Press me"),
            ))));
  }
}
```

PopupMenuButton

Usually, a PopupMenuButton is being used with the AppBar as one of its actions button:

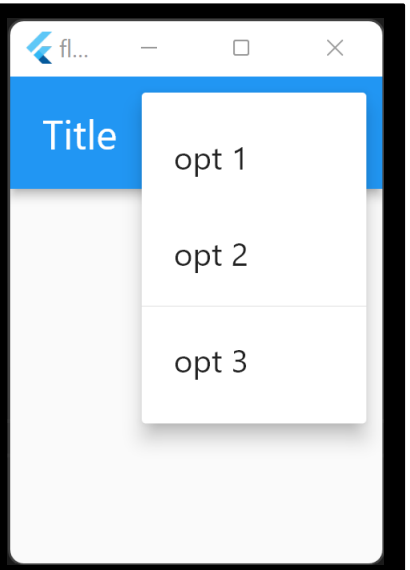
```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […];
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        debugShowCheckedModeBanner: false,
        home: Scaffold(
          appBar: AppBar(
                  title: Text("Title"),
                  actions: [
                      PopupMenuButton<int>(itemBuilder: itemBuilderFnc),
                  ]),
          ));
  }
}
```



PopupMenuButton

For the menu, we can use a PopupMenuDivider to separate different items from the menu (in our case, “opt 1” and “opt 2” from “opt 3”

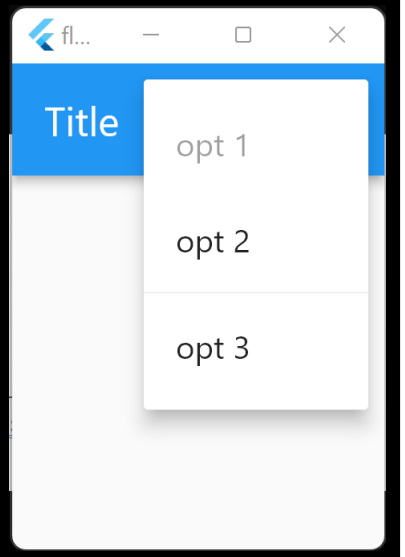
```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => [
        PopupMenuItem<int>(value: 1, child: Text("opt 1")),
        PopupMenuItem<int>(value: 2, child: Text("opt 2")),
        PopupMenuDivider(height: 5),
        PopupMenuItem<int>(value: 3, child: Text("opt 3"))
      ];
  @override
  Widget build(BuildContext context) {…}
}
```



PopupMenuButton

For PopuMenuItem instances, there is a property called enable that can be use to disable or enable an item.

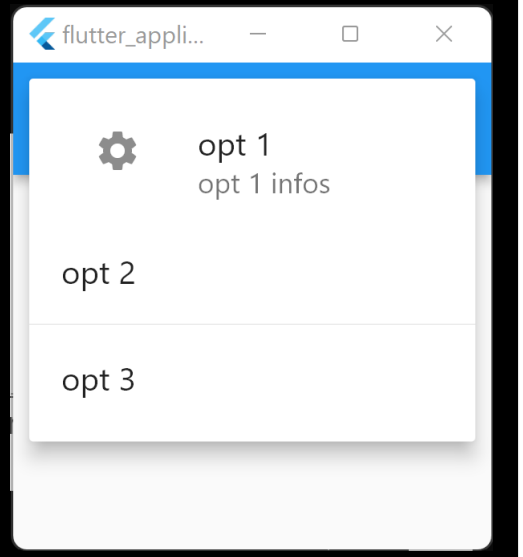
```
class MyAppState extends State<MyApp> {
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => [
        PopupMenuItem<int>(value: 1, enabled: false, child: Text("opt 1")),
        PopupMenuItem<int>(value: 2, child: Text("opt 2")),
        PopupMenuDivider(height: 5),
        PopupMenuItem<int>(value: 3, child: Text("opt 3"))
      ];
  @override
  Widget build(BuildContext context) {…}
}
```



PopupMenuButton

To create a more complex menu item (for example something with an icon) one can use a Row or a ListTile for the child property.

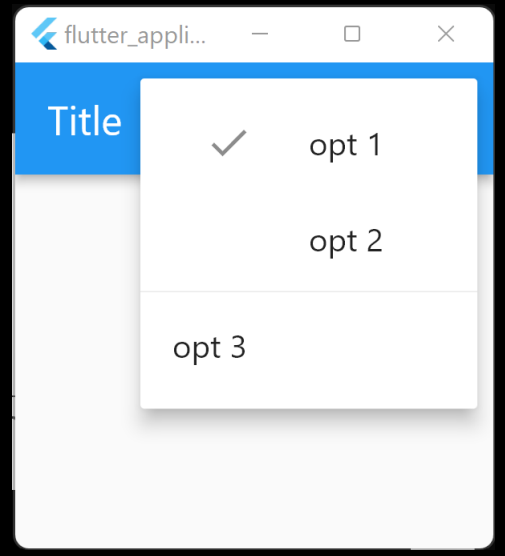
```
List<PopupMenuEntry<int>> itemBuilderFnc(context) => [
      PopupMenuItem<int>(
          value: 1,
          child: ListTile(
            leading: Icon(Icons.settings),
            title: Text("opt 1"),
            subtitle: Text("opt 1 infos"),
          )),
      PopupMenuItem<int>(value: 2, child: Text("opt 2")),
      PopupMenuDivider(height: 5),
      PopupMenuItem<int>(value: 3, child: Text("opt 3"))
    ];
```



PopupMenuButton

And you can use a CheckedPopupMenuItem if some of the items from your menu should be checked or not (via checked property).

```
List<PopupMenuEntry<int>> itemBuilderFnc(context) => [
      CheckedPopupMenuItem<int>(
          value: 1,
          checked: true,
          child: Text("opt 1")),
      CheckedPopupMenuItem<int>(
          value: 2,
          checked: false,
          child: Text("opt 2")),
      PopupMenuDivider(height: 5),
      PopupMenuItem<int>(value: 3, child: Text("opt 3"))
    ];
```



PopupMenuButton

Let's see a more complex example where we have a menu with 3 options (red, green and blue) that if pressed will change the color of the AppBar accordingly.

The initial color of the AppBar should be different (e.g. a grey)

PopupMenuButton

Step 1 → create two variables (“c” for the actual color, and cList – an array with red, green and blue)

```
class MyAppState extends State<MyApp> {
  var cList = [Colors.red, Colors.green, Colors.blue];
  Color c = Colors.grey;

}
```

PopupMenuButton

Step 2 → create the itemBuilderFnc to be used when constructing a PopupMenuButton

```
class MyAppState extends State<MyApp> {
  var cList = [Colors.red, Colors.green, Colors.blue];
  Color c = Colors.grey;
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => [
        PopupMenuItem<int>(value: 0, child: Text("Red")),
        PopupMenuItem<int>(value: 1, child: Text("Green")),
        PopupMenuItem<int>(value: 2, child: Text("Blue"))
      ];
```

Notice that the value of each item is an index in **cList** array to the color it represents.

```
}
```

PopupMenuButton

Step 3 → create the build method with an AppBar that uses “c” variable for its background color.

```
class MyAppState extends State<MyApp> {
  var cList = [Colors.red, Colors.green, Colors.blue];
  Color c = Colors.grey;
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […]
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        debugShowCheckedModeBanner: false,
        home: Scaffold(
          appBar: AppBar(title: Text("Title"),
                         backgroundColor: c,
                         actions: [ PopupMenuButton<int>(…) ])
          ));
  }
}
```

PopupMenuButton

Step 3 → create the build method with an AppBar that uses “c” variable for its background color.

```
class MyAppState extends State<MyApp> {
  var cList = [Colors.red, Colors.green, Colors.blue];
  Color c = Colors.grey;
  List<PopupMenuEntry<int>> itemBuilderFnc(context) => […]
  @override
  Widget build(Bui
    return Materia
        debugShowC
        home: Scaf
          appBar: AppBar(title: Text('Title'),
                         backgroundColor: c,
                         actions: [ PopupMenuButton<int>(…) ])
          ));
  }
}
```

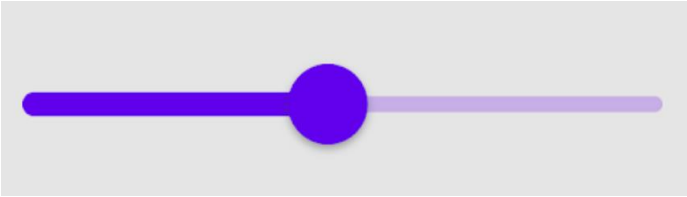
```
PopupMenuButton<int>(
   itemBuilder: itemBuilderFnc,
   onSelected: (i) { setState(() { c = cList[i]; }); }
)
```

Slider

Slider

A slider is a widget that can be used to select a numerical value within a specific interval.

It is usually used for things like music / SFX volume, color (RGB / Saturation) selection, brightness selection, etc.



Slider

Constructor:

```
Slider({
   required double value,
   required void Function(double)? onChanged,
   double min = 0.0,
   double max = 1.0,
   int? divisions,
   String? label,
   Color? activeColor,
   Color? inactiveColor,
   … }
)
```

Slider

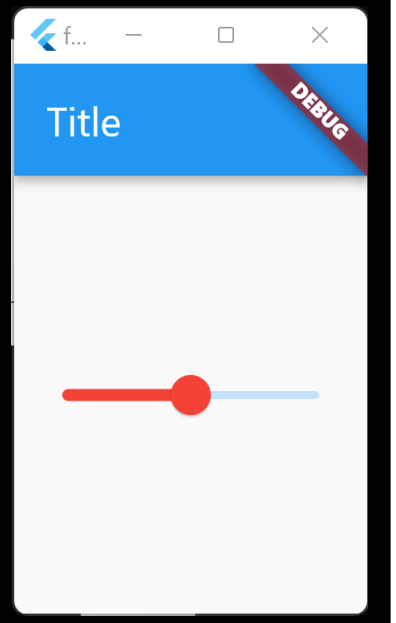
A very simple example:

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
               appBar: AppBar( title: Text("Title")),
               body: Center(
                     child: Slider(value: 20,
                                   min: 10,
                                   max: 30,
                                   onChanged: (v) => {})),
                       ));
  }
}
```

Slider

Use activeColor property to change slider color.

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
              appBar: AppBar( title: Text("Title")),
              body: Center(
                    child: Slider(value: 20,
                                  min: 10,
                                  max: 30,
                                  activeColor: Colors.red ,
                                  onChanged: (v) => {})),
                    ));
  }
}
```



Slider

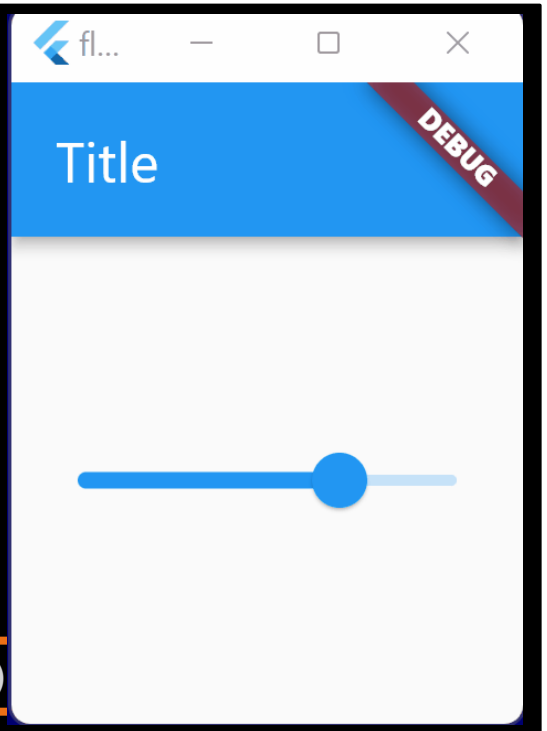
To disable a slider, set the onChange callback to null.

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
              appBar: AppBar( title: Text("Title")),
              body: Center(
                    child: Slider(value: 20,
                                  min: 10,
                                  max: 30,
                                  onChanged: null)),
                      ));
  }
}
```

Slider

The value of the slider has to be used via the onChanged callback

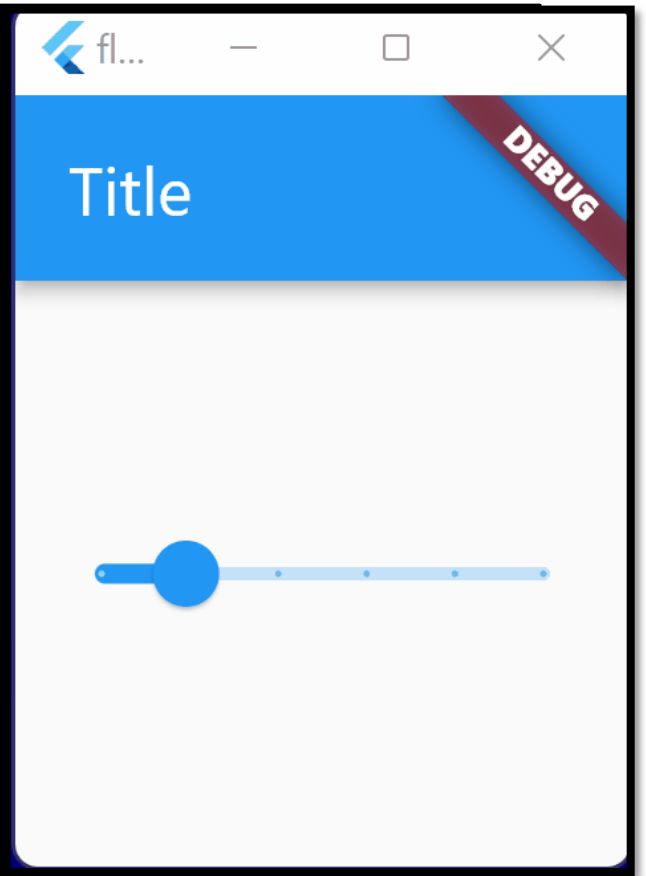
```
class MyAppState extends State<MyApp> {
  double v = 20;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
      appBar: AppBar( title: Text("Title")),
      body: Center(
          child: Slider(
              value: v,
              min: 10, max: 30,
              onChanged: (val) => setState(() { v = val; }))))
    ));
  }
}
```



Slider

To show the label, use division and label together (just like in this example).

```
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
    appBar: AppBar( title: Text("Title")),
    body: Center(
        child: Slider(
            value: v,
            min: 10,
            max: 30,
            divisions: 5,
            label: v.toString(),
            onChanged: (val) => setState(() { v = val; })
        )),
  ));
}
```



Q & A