Flutter Framework

COURSE 7 (REV 3)

GAVRILUT DRAGOS

Agenda

Images & assets

Buttons

- ElevatedButton
- TextButton
- OutlinedButton
- FloatingActionButton
- IconButton
- CloseButton
- BackButton

Icons

Predefinea icons in flutter

Images

Images

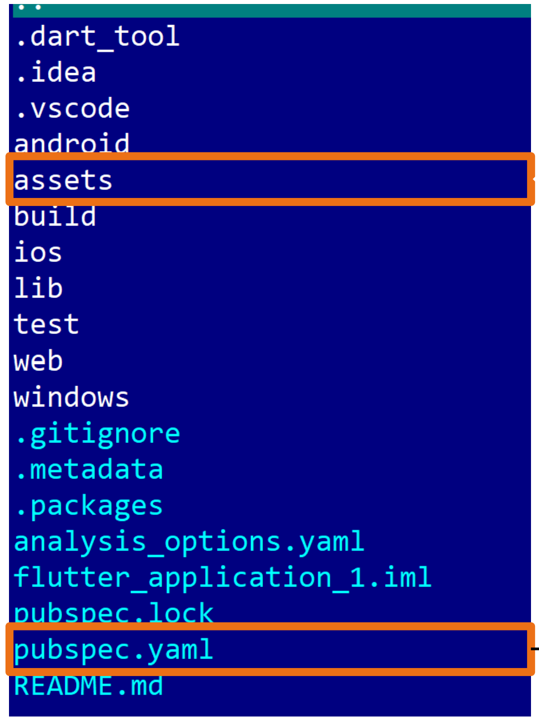
Assets (and in particular images) are an integral part of any application or game.

In flutter, assets have to be defined in the **pubspec.yaml** file.

To add an image to your application, follow the next steps:

1. Create a folder (or a hierarchy of folders) into the root folder of your application. One common usage is to create a folder named “assets” and in that folder create another folder named “images” where all your images will be stored
2. Copy the images that you want to use in that folder.

Images

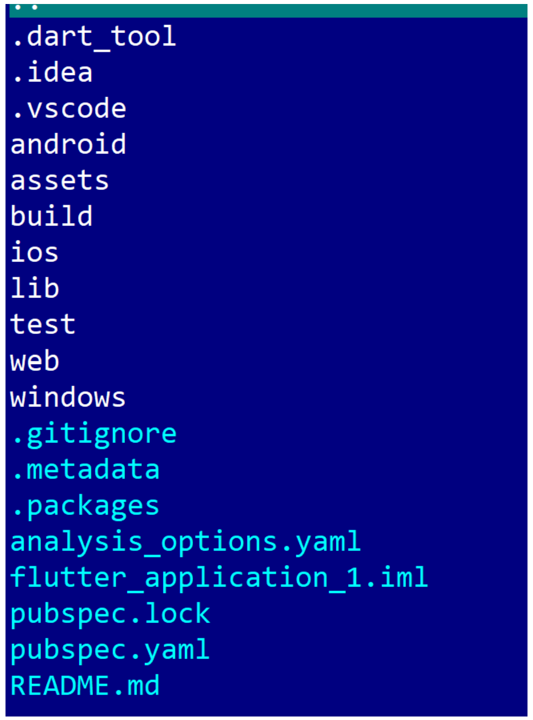


```
..
.dart_tool
.idea
.vscode
android
assets
build
ios
lib
test
web
windows
.gitignore
.metadata
.packages
analysis_options.yaml
flutter_application_1.iml
pubspec.lock
pubspec.yaml
README.md
```

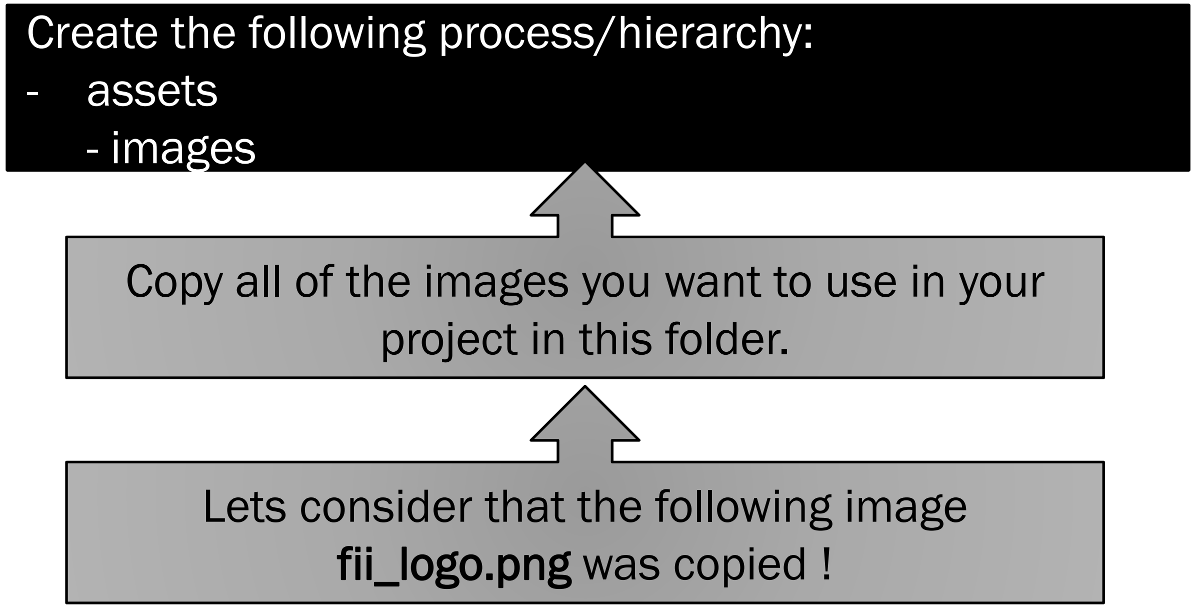
First, create a folder assets in the root of the application.

For simplicity, consider the root of the application the folder where the file **pubspec.yaml** is located

Images



```
..
.dart_tool
.idea
.vscode
android
assets
build
ios
lib
test
web
windows
.gitignore
.metadata
.packages
analysis_options.yaml
flutter_application_1.iml
pubspec.lock
pubspec.yaml
README.md
```



Create the following process/hierarchy:

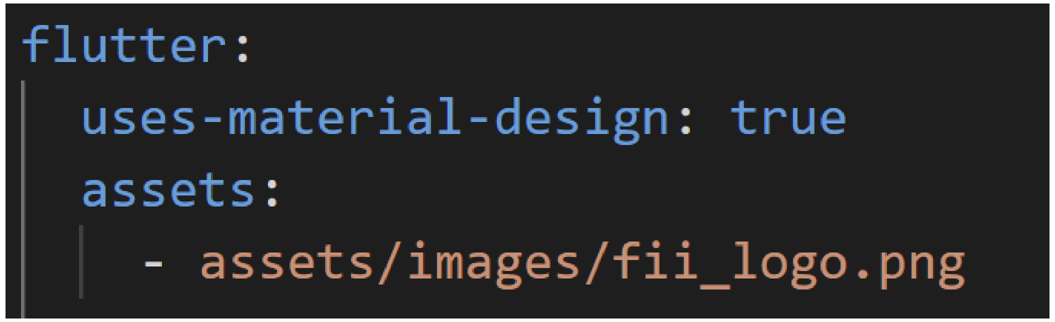
- assets
  - images

Copy all of the images you want to use in your project in this folder.

Lets consider that the following image **fii_logo.png** was copied !

Images

Once the images have been copied in their asset folder, modify the **pubspec.yaml** file in the following way:



```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/images/fii_logo.png
```

If these items already exists just add the relative location of your images

It is also possible to add just the location of the folder (e.g. assets/images) if you want to include all existing images.

Images

After this modify the main.dart file from the application in the following way.

```
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("Test")),
                              body: Image.asset('assets/images/fii_logo.png')));
  }
}
```

Images

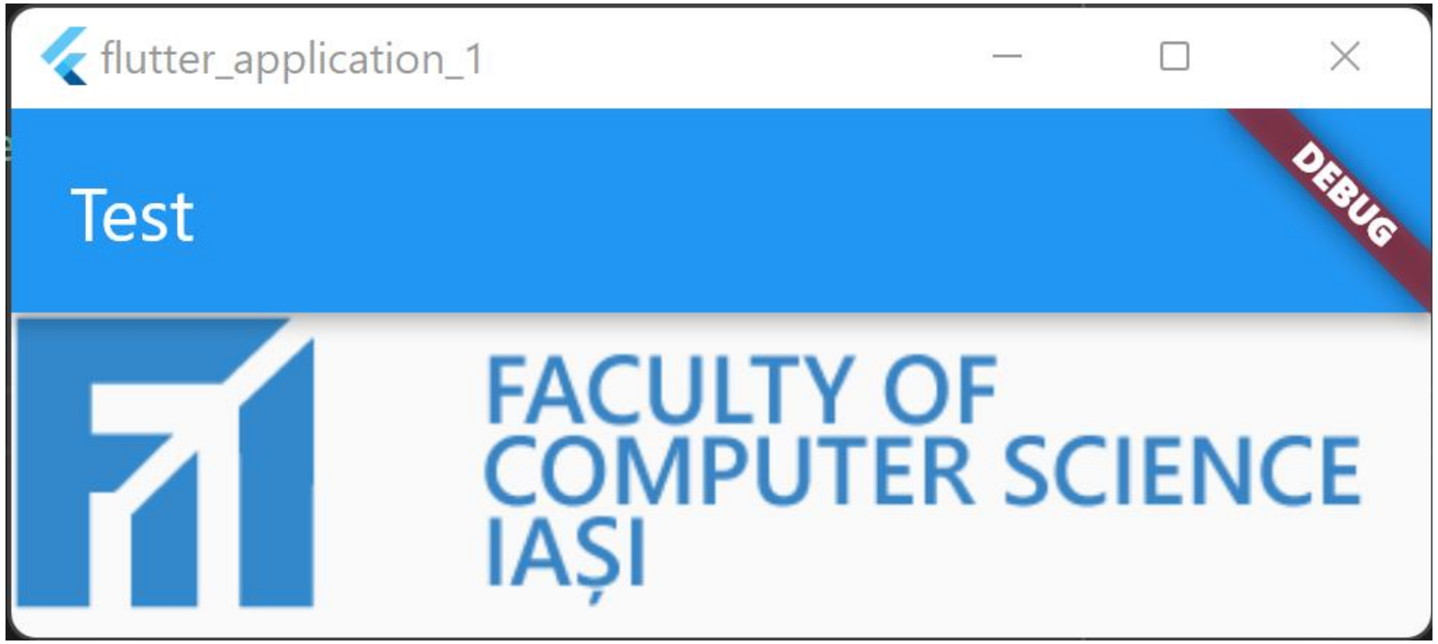
After this modify the main.dart file from the application in the following way.

```dart
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("Test")),
                              body: Image.asset('assets/images/fii_logo.png')));
  }
}
```

The widget images loads an image from a stream / memory / file or asset. In case of assets, the relative path from the root of the application has to be provided.

Images

Finally, when execution this application, you should see something that looks like the next picture.



Images

Image widget has four name constructors:

`Image.asset (String name, {…})`	A hash code for this object
`Image.file (File file, {…})`	Type of the object
`Image.memory (Uint8list bytes, {…})`	Call whenever a non-existing property is called
`Image.network (String uri, {…})`	A string representation for current object

The optional parameters include:

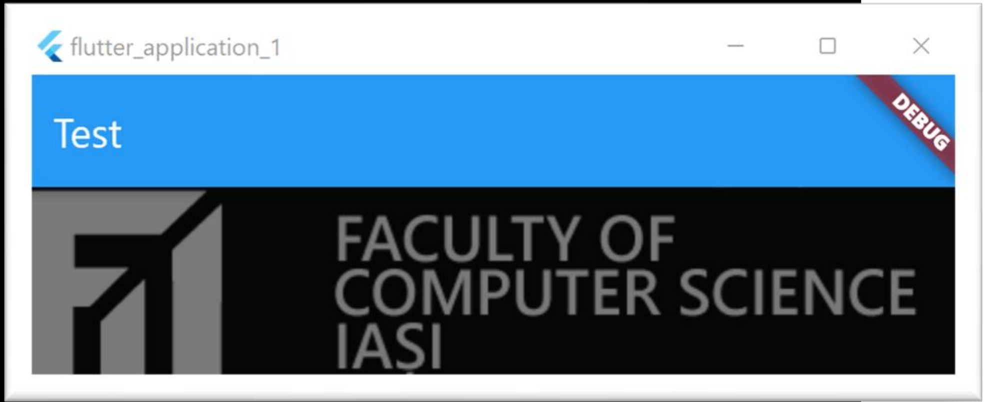
- Color
- Blending
- Scale

More information can be found on: https://api.flutter.dev/flutter/widgets/Image-class.html

Images

For example, in the previous example if we change the build method in the following way:

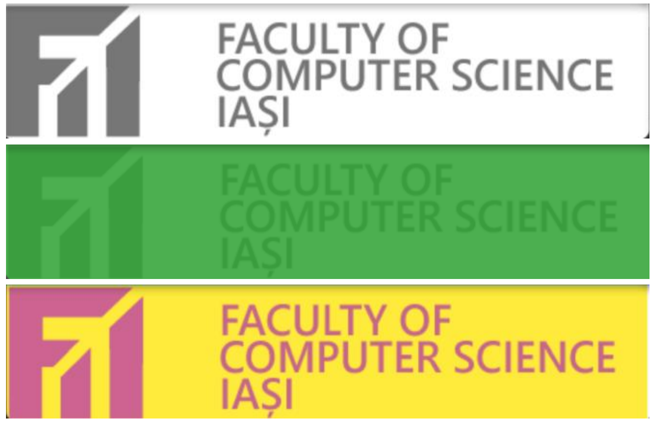
```
import 'package:flutter/material.dart';
void main() => runApp(MyApp());
class MyApp extends StatefulWidget { … }
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Image.asset('assets/images/fii_logo.png',
                colorBlendMode: BlendMode.color, color: Colors.black)));
  }
}
```



Images

Other combinations:

- **colorBlendMode**: BlendMode.color, **color**: Colors.white
- **colorBlendMode**: BlendMode.color, **color**: Colors.green
- **colorBlendMode**: BlendMode. difference, **color**: Colors.red

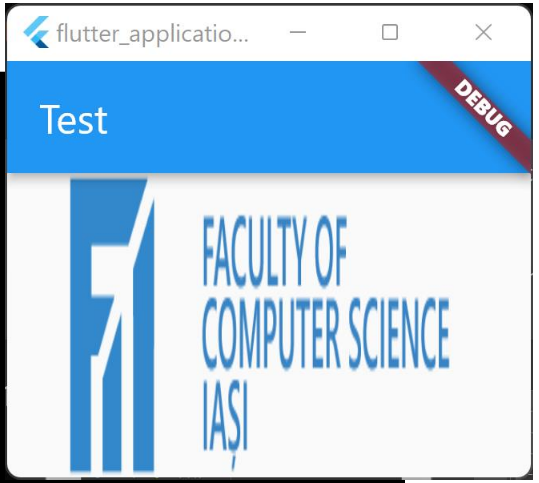


More details on how blend mode and color can be combine to change the way an image looks like can be found here: https://api.flutter.dev/flutter/dart-ui/BlendMode.html

Images

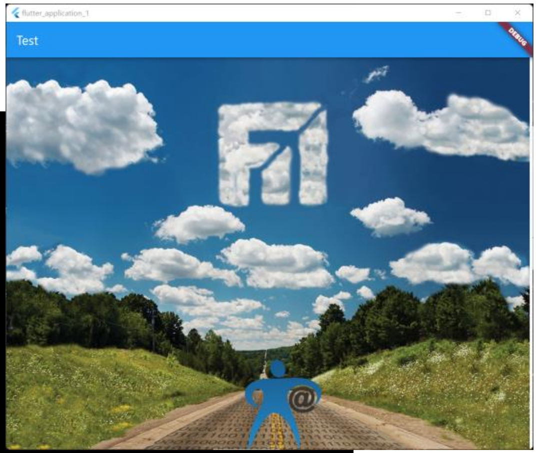
Another useful property is fit and can be use to stretch/skew and image:

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: Container(
                    child: Image.asset('assets/images/fii_logo.png',
                                        fit: BoxFit.fill,),
                    width: 200,
                    height: 200))));
  }
}
```



Images

To load an image from an URI location use Image.network constructor:

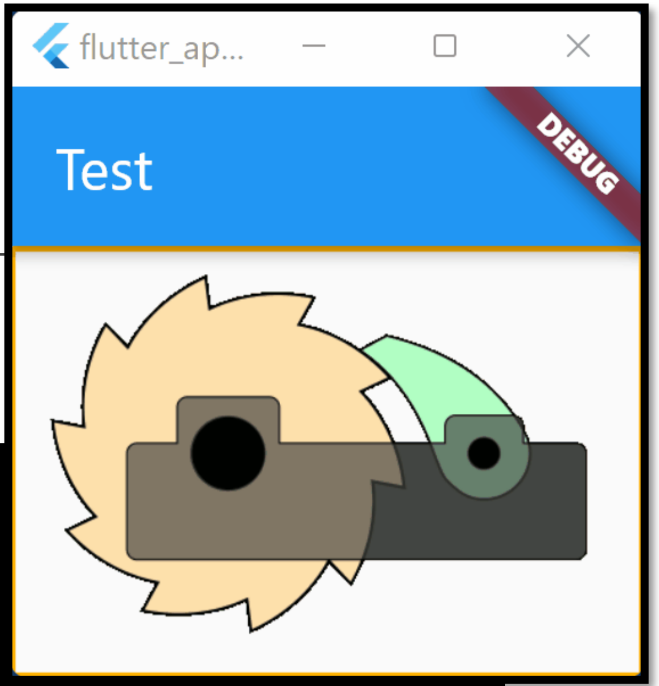


```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Image.network(
                'https://scontent.fias1-1.fna.fbcdn.net/v/t39.30808-
6/278101604_4917352475029667_7419691009543583291_n.jpg?_nc_cat=108&ccb=1-
5&_nc_sid=730e14&_nc_ohc=SaGdsZnlGWcAX-BjoiR&_nc_ht=scontent.fias1-
1.fna&oh=00_AT_B-qyyXlHvcKBGDbvxVNWFaoIVIOxA2yTJle6F3DbthQ&oe=626CF29A')));
  }
}
```

Images

Image widget also works with animated gifs
(either from the network or from your assets):

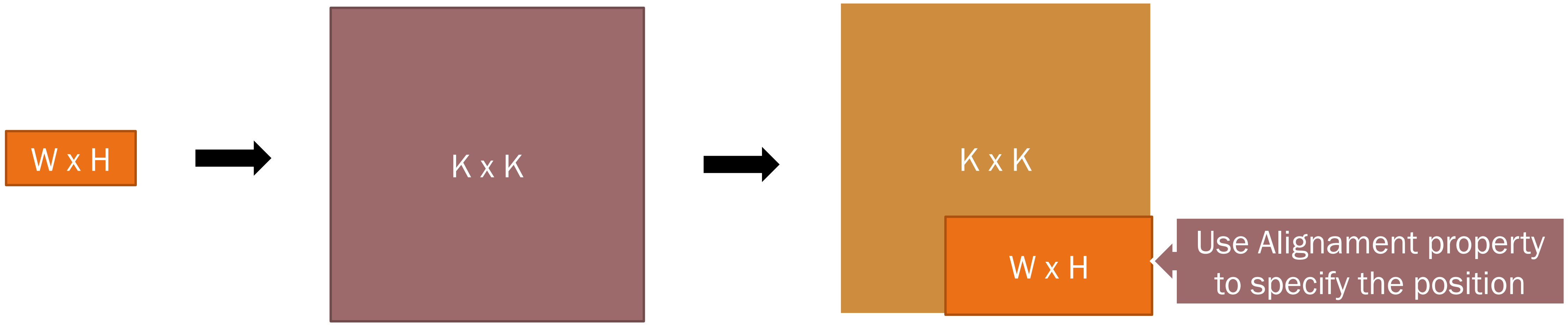
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Image.network(
                'https://upload.wikimedia.org/wikipedia/commons/7/7e/Ratchet_Drawing_Animation.gif')));
  }
}
```



Images

Another interesting properties are width and height that can be used to set up a width and/or a height for an image. Its important to understand that the image will retain its width/height aspect ratio.

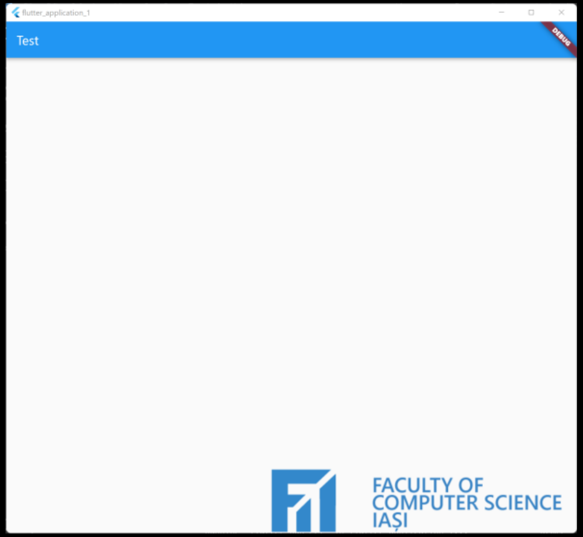
This means that in reality, only one size (either with or height) will be applied and the other one will be computed so that the aspect ratio is mentained.



Images

Using width and height:

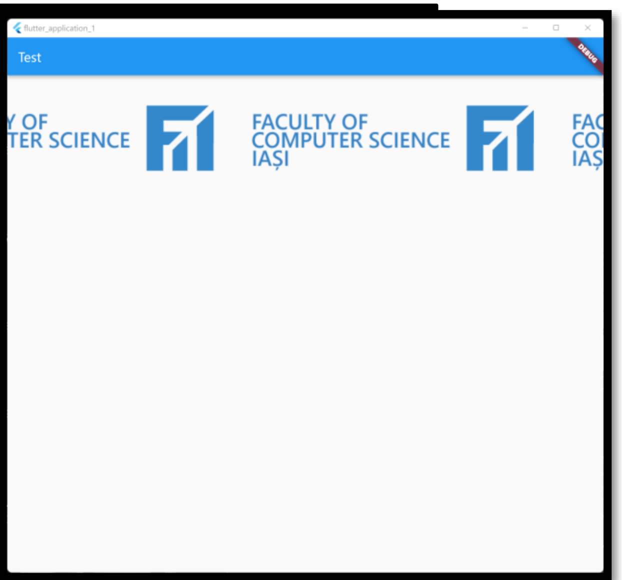
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Image.asset(
              'assets/images/fii_logo.png',
              width: 1000,
              height:1000,
              alignment: Alignment.bottomRight,
            )));
  }
}
```



Images

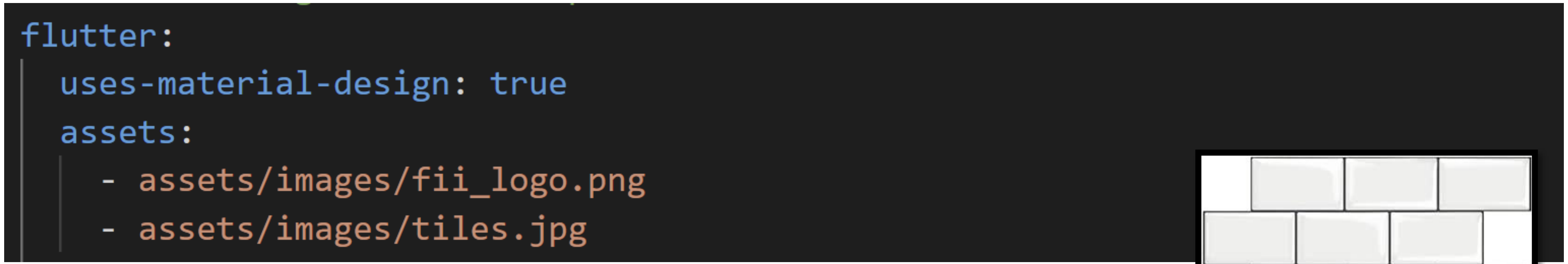
You can also use the repeat function to duplicate an image multiple times. Usually this option has to be combined with width or height property to increase the area where the image will be replicated. Possible values for repeat are **repeatX** or **repeatY** (to repeat an image on axex X or Y) or just simple **repeat** to repeat an image across the entire space (both X and Y axes).

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Image.asset('assets/images/fii_logo.png',
                width: 1000, repeat: ImageRepeat.repeatX)));
  }
}
```



Images

The repeat property is often used with tiles (square images) that put together form a pattern of some sort. Let's consider that we add another image like this in our pubspec.yaml file (and let's consider that its name is tiles.jpg).



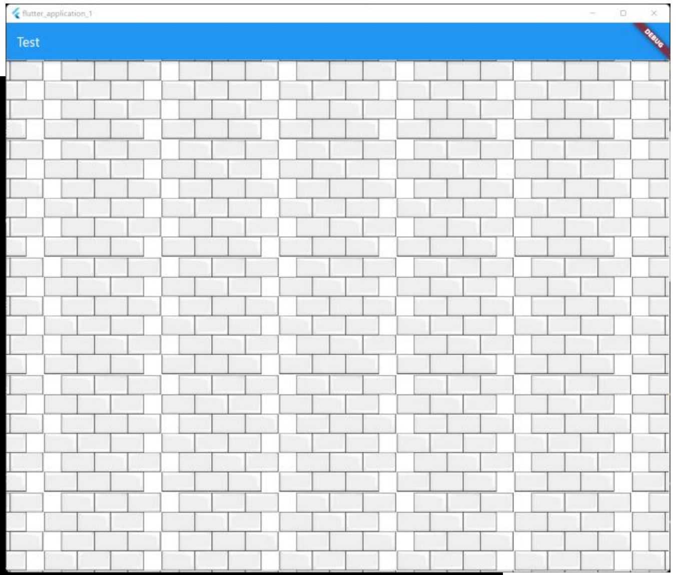
```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/images/fii_logo.png
    - assets/images/tiles.jpg
```

In this example we will use a brick-like image to simulate a wall.

Images

Using repeat property:

```dart
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Image.asset('assets/images/tiles.jpg',
                width: 1000,
                height: 1000,
                scale: 2,
                repeat: ImageRepeat.repeat)));
  }
}
```

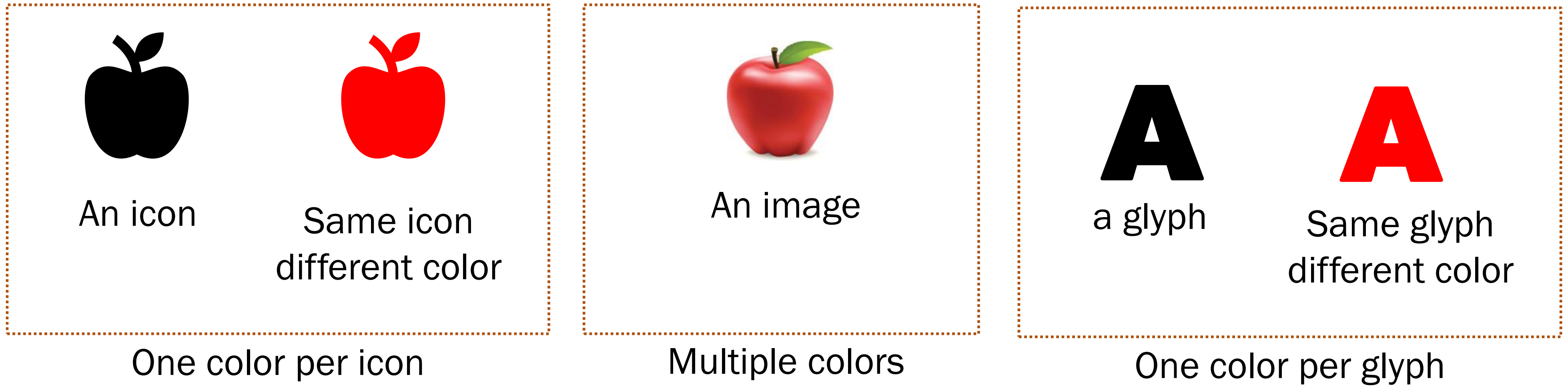


Icons

Icons

Icons are also an integral part of any application.

Icons (as a concept in Flutter) are glyphs (meaning that they are similar cu a character representation from a font). In other words, an icon is a an image with only one color (that can be set up) and a transparent background.



Icons

The widget that draws an icon is called **Icon** and has the following constructor:

```
Icon (IconData? icon, {Key? key,
                        double? size,
                        Color? color,
                        String? semanticLabel,
                        TextDirection? textDirection})
```

Out of this:

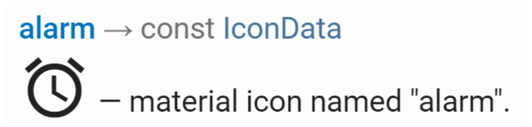
- The icon parameter is one of Icons.xxx existing values
- The size parameter is the size of icon in logical pixels
- The color parameter allows one to change the color of the existing icon / glyph

Icons

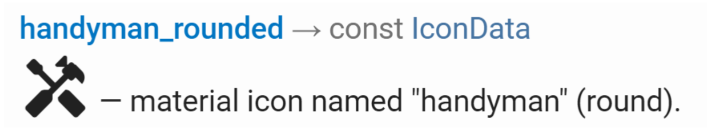
Flutter has more than 8300 predefined icons that can be used (related to accessibility, operations, zooming, calls, messages, social, etc).

You can find a full list of predefined icons here:
https://api.flutter.dev/flutter/material/Icons-class.html

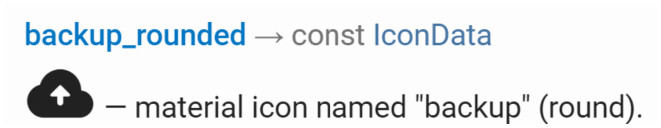
Some examples:


alarm → const IconData

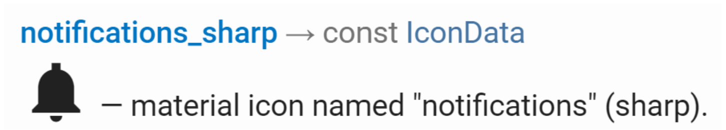
⏰ — material icon named "alarm".


handyman_rounded → const IconData

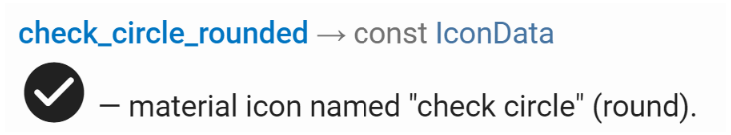
— material icon named "handyman" (round).


backup_rounded → const IconData

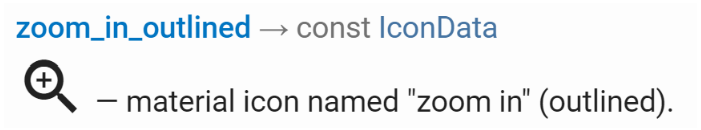
— material icon named "backup" (round).


notifications_sharp → const IconData

🔔 — material icon named "notifications" (sharp).


check_circle_rounded → const IconData

✓ — material icon named "check circle" (round).

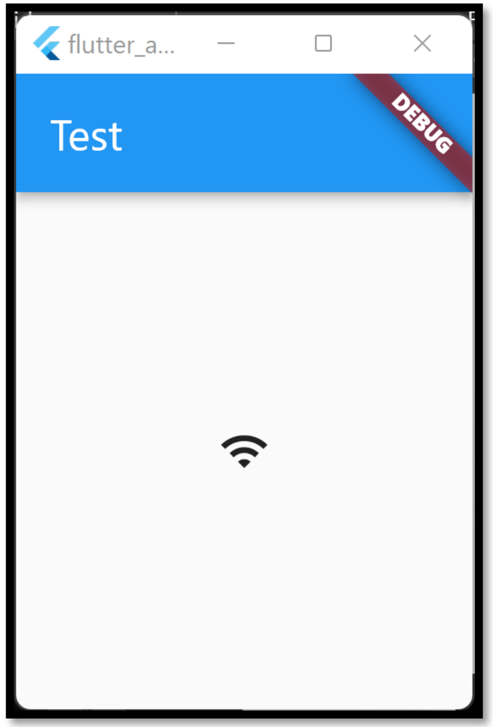

zoom_in_outlined → const IconData

— material icon named "zoom in" (outlined).

Icons

The next example shows the icon for wifi (*Icons.wifi*) in the center of the app.

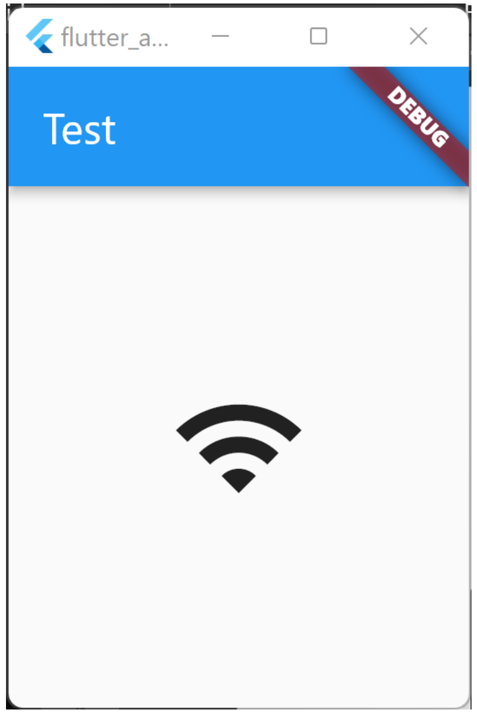
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(child: Icon(Icons.wifi))),
    );
  }
}
```



Icons

The next example shows the icon for wifi (*Icons.wifi*) in the center of the app, but sized to 64 logical pixels.

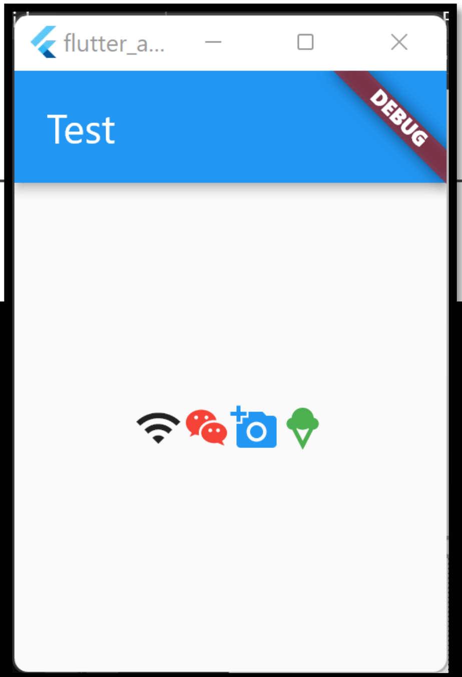
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(child: Icon(Icons.wifi, size: 64))),
    );
  }
}
```



Icons

Multiple icons with different colors.

```
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: Row(
                    children: [
                        Icon(Icons.wifi),
                        Icon(Icons.wechat, color: Colors.red),
                        Icon(Icons.add_a_photo, color: Colors.blue),
                        Icon(Icons.icecream, color: Colors.green)
                    ],
                    mainAxisSize: MainAxisSize.min,))));
  }
```



Icons

For custom icons, Flutter provides two classes that can be use with various assets: ImageIcon

```
ImageIcon (ImageProvider<Object>? image, {Key? key,
                                          double? size,
                                          Color? color,
                                          String? semanticLabel})
```

and AssetImage

```
AssetImage (String assetName, {AssetBundle? bundle, String? package})
```

The most common usage is to combine these two method as follows:

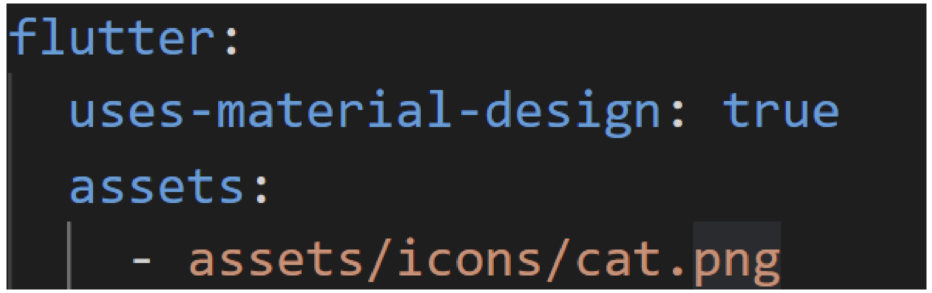
```
ImageIcon (AssetImage (”<path to asset image>”))
```

Icons

Let’s create a custom icon. We will use a PNG image, however for clarity it is recommended to copy these images in another folder (not images) → for example icons

1. **First**, we need to add an image to our assets folder (let’s name it ‘cat.png’). Make sure that the image has transparency.

2. **Second**, add the new image to the pubspec.yaml file

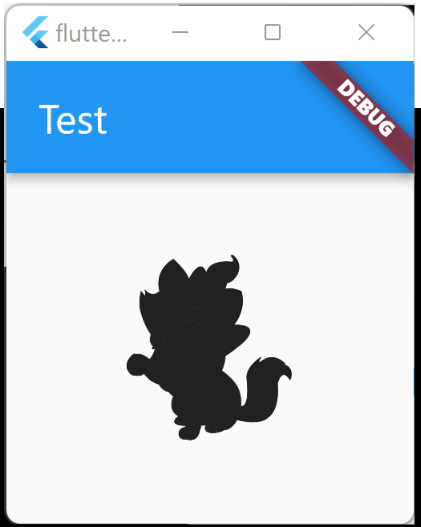


```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/icons/cat.png
```

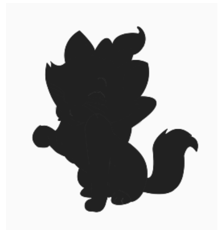
Icons

Now, let's draw a custom icon:

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ImageIcon(AssetImage("assets/icons/cat.png"),
                          size: 96))));
  }
}
```



Icons

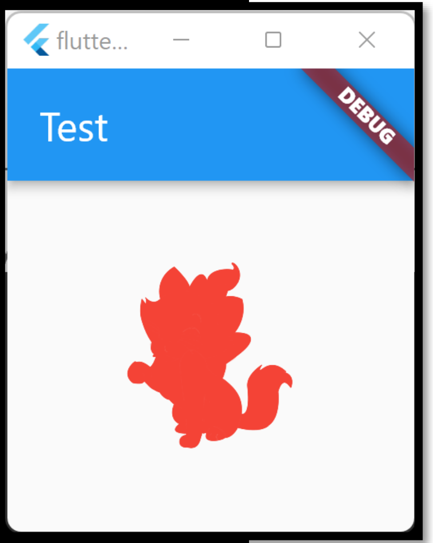
So why did the icon look like this:  and not like the actual image: ?

Well … the reason is that we wanted to create an icon (a glyph) and not another image. As such ImageIcon object will convert the existing image into one color format image that can be used as a glyph. In our case, all pixels that were not transparent (pretty much the entire image of a cat) will have only one color (and appears as a shadow when painted).

Icons

Now that we know how custom icons ca be created, we can apply other colors on them.

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ImageIcon(AssetImage("assets/icons/cat.png"),
                                 size: 96,
                                 color: Colors.red ))));
  }
}
```

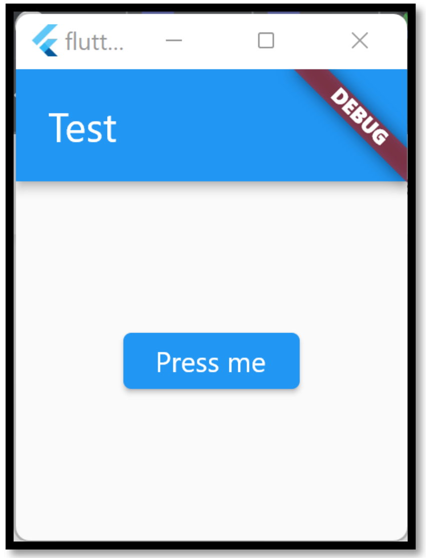


Buttons

Buttons

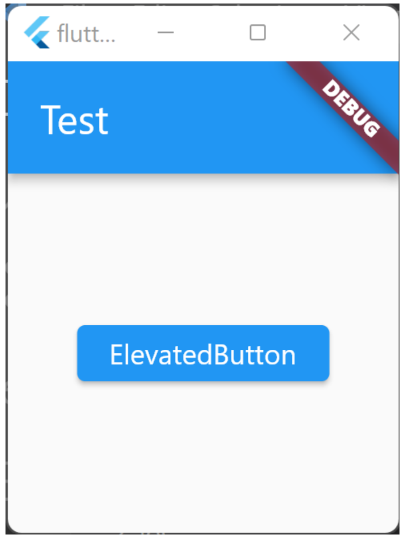
Buttons are an integrated part of any flutter application. There are 3 types of widgets defined in Flutter: **ElevatedButton**, **TextButton** and **OutlinedButton**

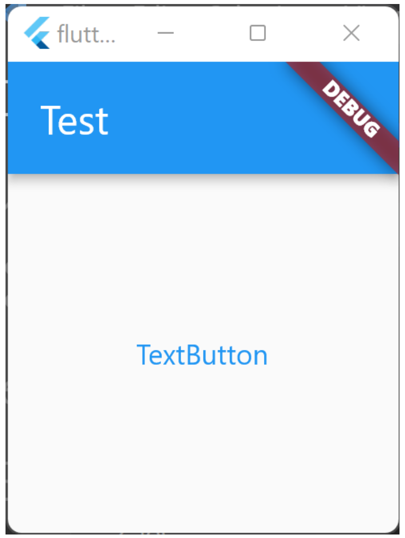
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ElevatedButton(
                    onPressed: () => {},
                    child: Text("Press me")))));
  }
}
```

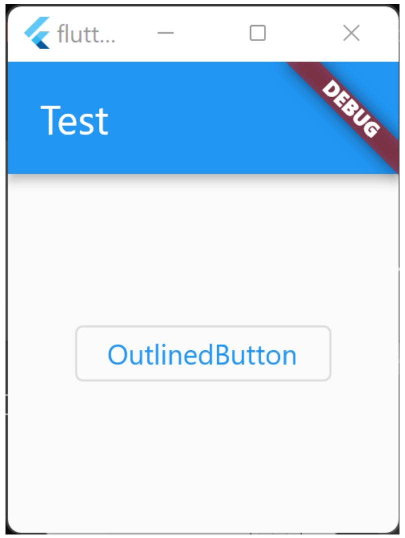


Buttons

Buttons are an integrated part of any flutter application. There are 3 types of widgets defined in Flutter: **ElevatedButton**, **TextButton** and **OutlinedButton.** The main difference lies in how these type of Buttons are painted.



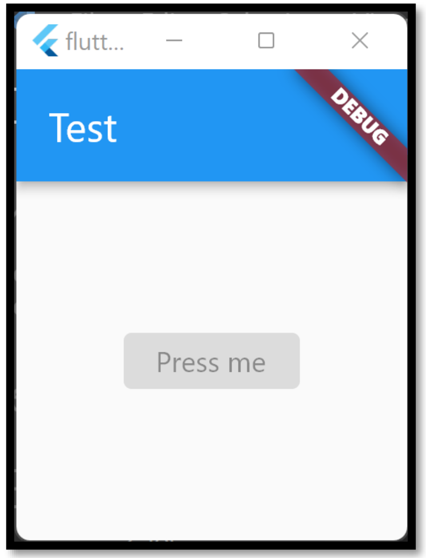




Buttons

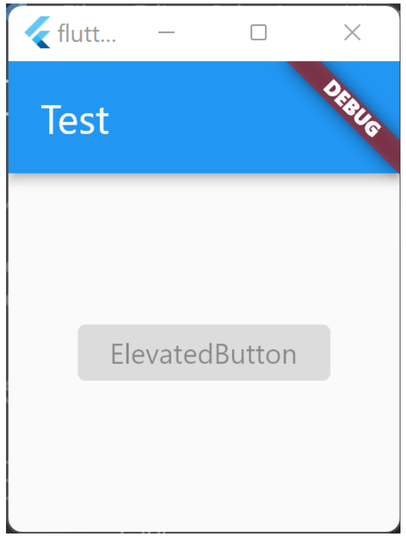
To disable a button , just set the onPressed property to null.

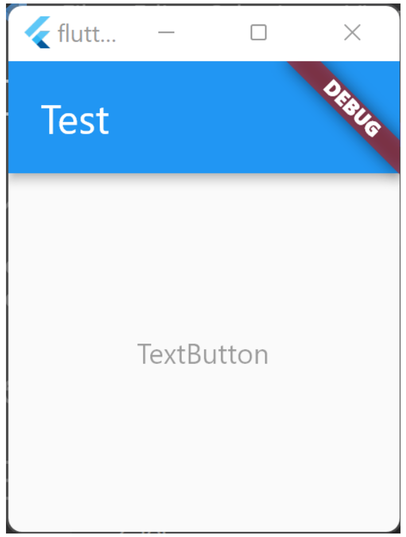
```
class MyAppStateextends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ElevatedButton(
                  onPressed: null ,
                  child: Text("Press me")))));
  }
}
```

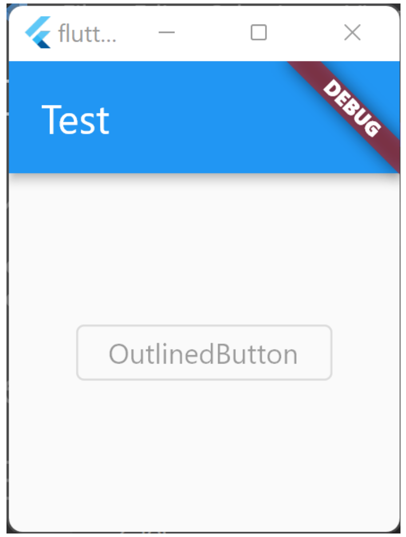


Buttons

To disable a button , just set the onPressed property to null.





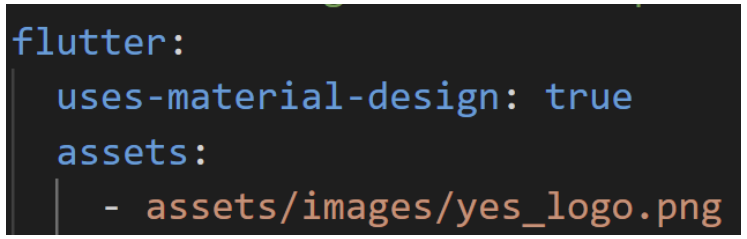


Buttons

What if we want to create a button that just shows an image.

1. **First**, we need to add an image to our assets folder (let’s name it ‘yes_logo.png’). Make sure that the image has transparency.

2. **Second**, add the new image to the pubspec.yaml file

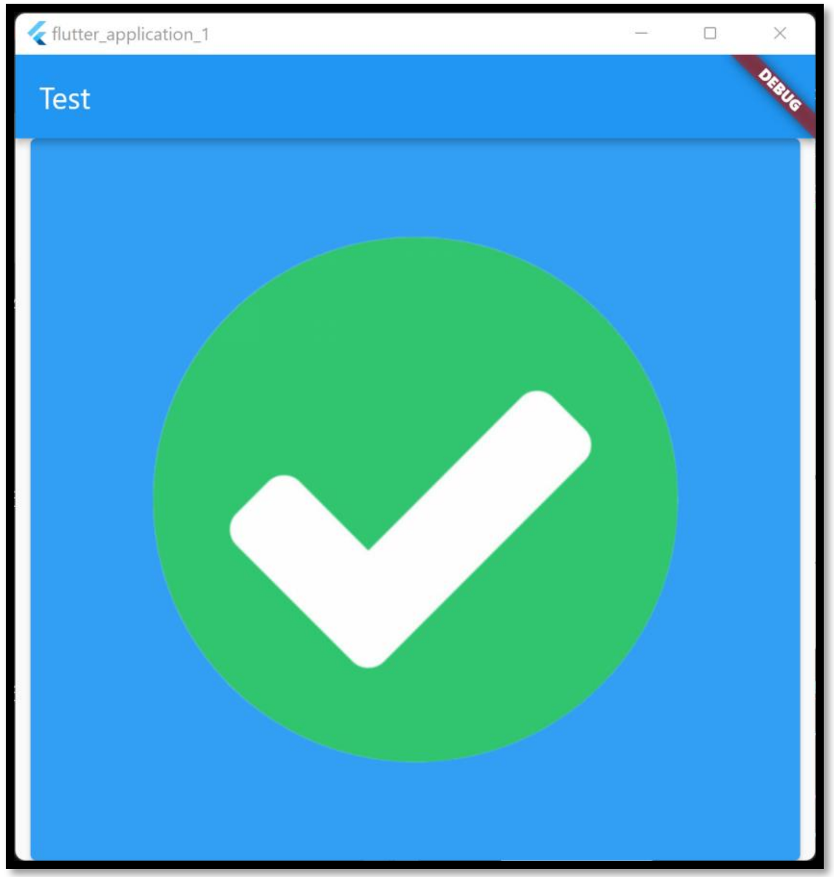


```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/images/yes_logo.png
```

Buttons

What if we want to create a button that just shows an image.

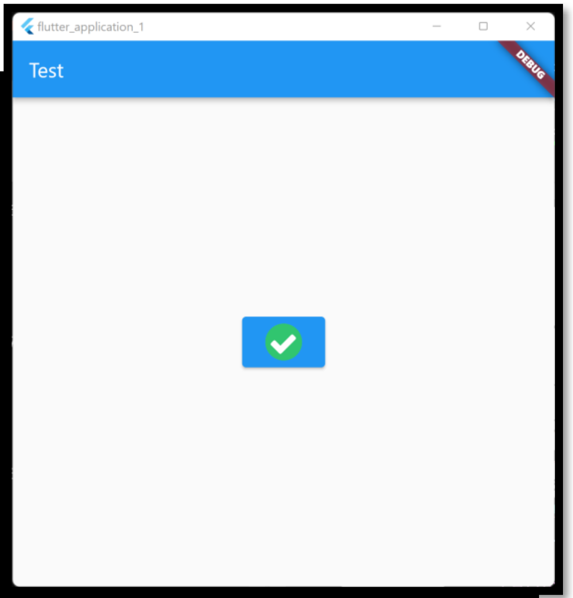
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ElevatedButton(
                    onPressed: () => {},
                    child:
Image.asset("assets/images/yes_logo.png")))));
  }
}
```



Buttons

In this particular case, the image size was 700 x 700 pixels and as such it increased the size of the button to almost the entire app screen. To solve this, we can use the .width and .height parameters in the Image constructor.

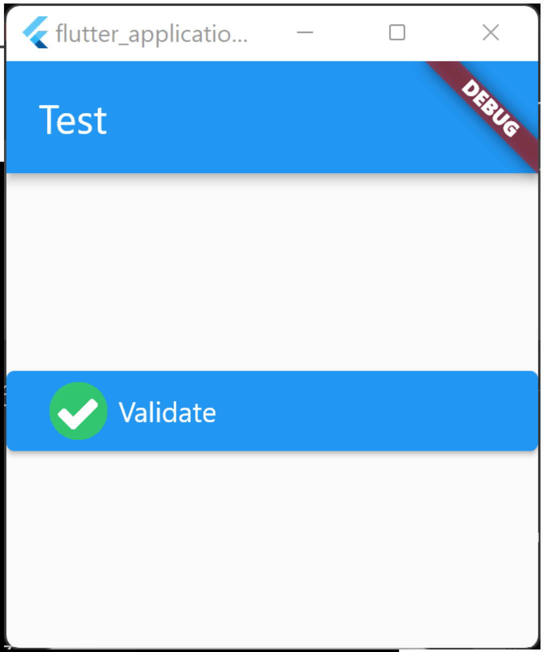
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ElevatedButton(
                    onPressed: () => {},
                    child: Image.asset("assets/images/yes_logo.png",
                                          height: 50, width: 50)))));}
}
```



Buttons

But what if we want to add both an image and a text to a button ?

```
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: ElevatedButton(
                    onPressed: () => {},
                    child: Row(children: [
                      Image.asset("assets/images/yes_logo.png",
                                   height: 40, width: 40),
                      Text("Validate")
                    ])))));
}
```

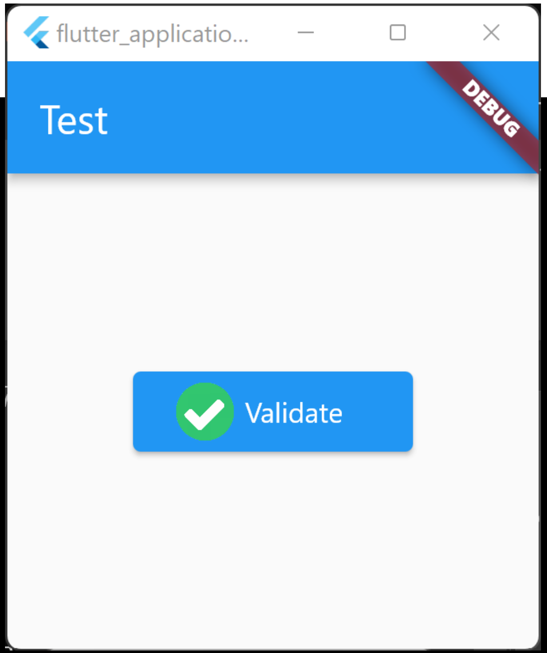


The main issue here is that the Row widget extends to the entire app space

Buttons

One way around the Row widget problem is to use a Container:

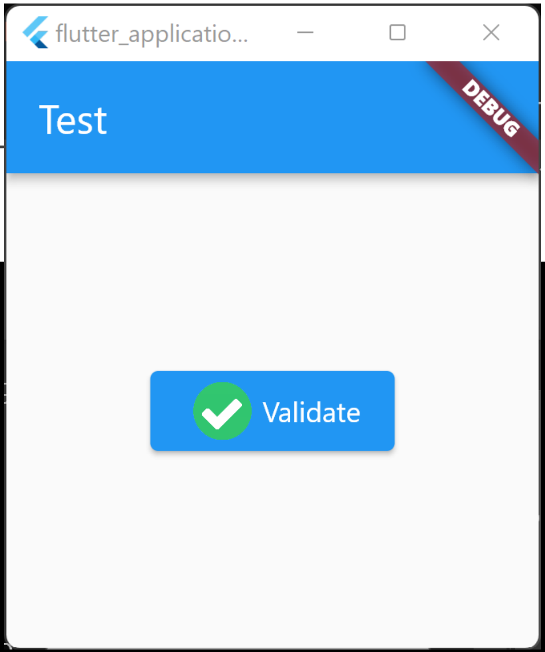
```
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
        appBar: AppBar(title: Text("Test")),
        body: Center(
          child: Container(
              width: 140,
              height: 40,
              child: ElevatedButton(
                 onPressed: () => {},
                 child: Row(children: [
                    Image.asset("assets/images/yes_logo.png", height: 40, width: 40),
                    Text("Validate")]))))));
  }
```



Buttons

Another solution is to set the property mainAxisSize to MainAxisSize.min

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(
              child: ElevatedButton(
                  onPressed: () => {},
                  child: Row(mainAxisSize: MainAxisSize.min,
                      children: [
                          Image.asset("assets/images/yes_logo.png",
                              height: 40, width: 40),
                          Text("Validate")
                      ])))));
}
```



Buttons

However, this approach comes with other issues as well (for example, one needs to compute the size of the Container so that it fits the button).

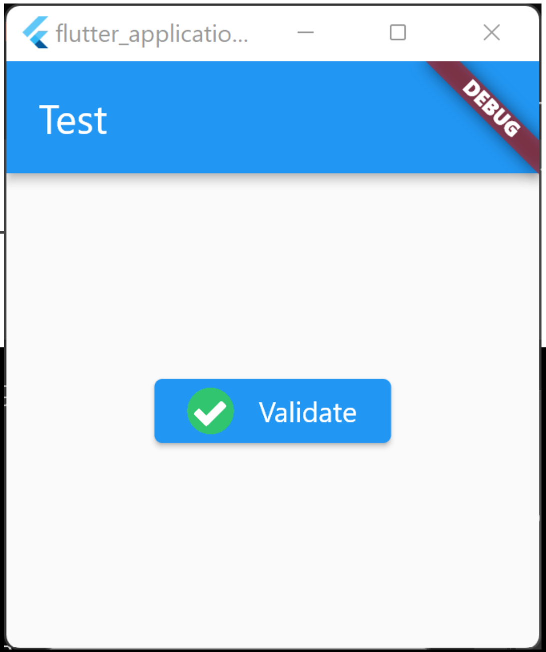
Since this type of buttons (with an icon and a text) are very common, a named constructor (.icon) was added to all three forms of buttons to easily describe a button with an image and a text.

```
ElevatedButton.icon (required Widget icon, required Widget label, […])
TextButton.icon (required Widget icon, required Widget label, […])
OutlinedButton.icon (required Widget icon, required Widget label, […])
```

Buttons

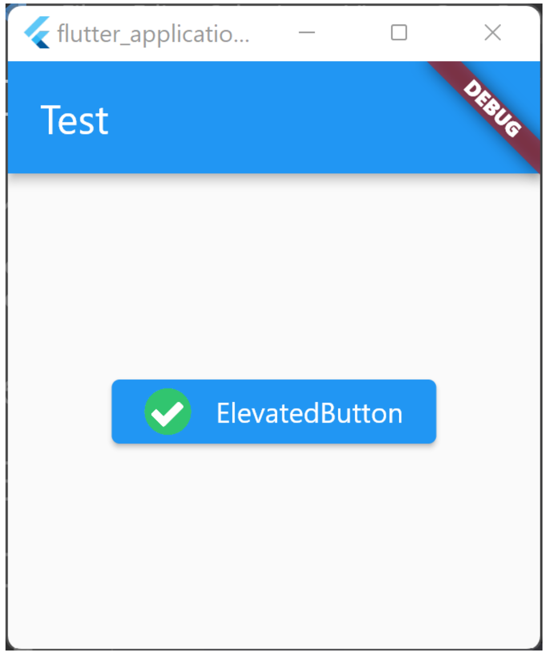
As such, the previous code will now look like this:

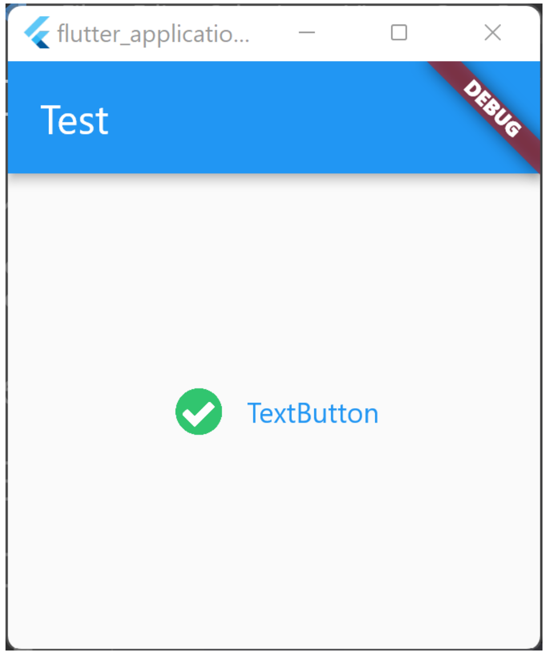
```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(
                child: ElevatedButton.icon(
                       onPressed: () => {},
                       icon: Image.asset("assets/images/yes_logo.png",
                                         width: 32, height: 32),
                       label: Text("Validate"),
          ))));
}
```

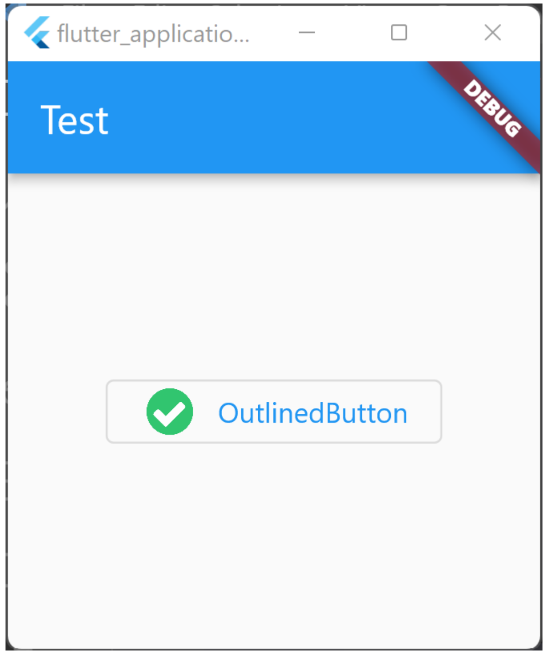


Buttons

As such, the previous code will now look like this:



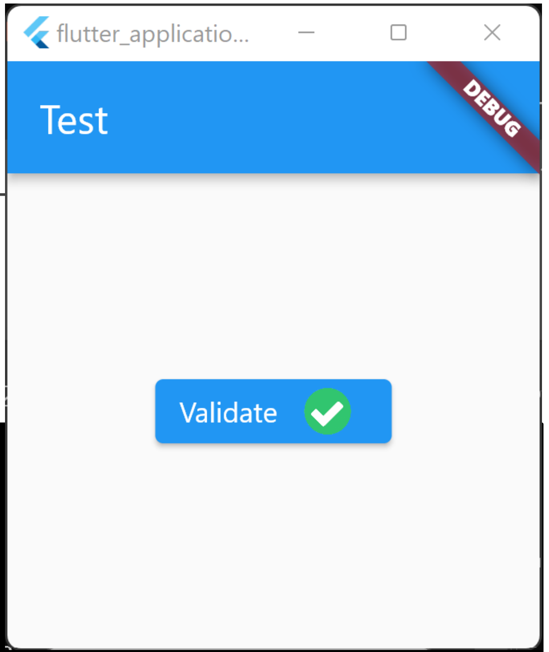




Buttons

Since both label and icon properties are in fact Widgets, one ca replace them with any type of widget. One simple effect being that we can create a right-side icon with the text on the left side).

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(
                child: ElevatedButton.icon(
                       onPressed: () => {},
                       label: Image.asset("assets/images/yes_logo.png",
                                          width: 32, height: 32),
                       icon: Text("Validate"),
          ))));
}
```



Buttons

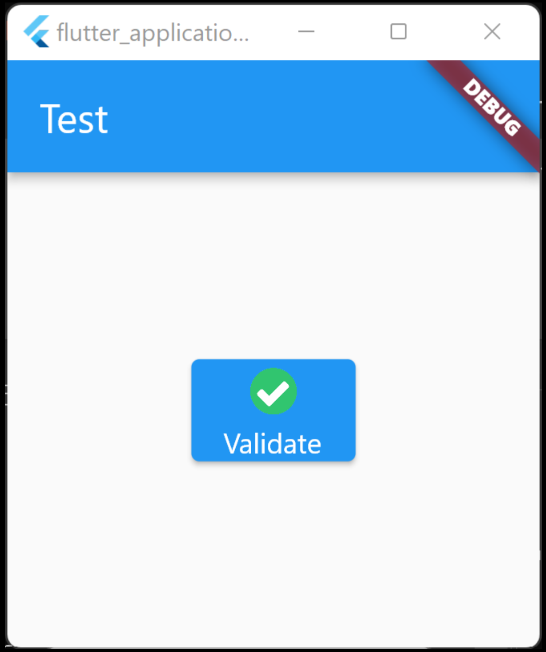
But what if we want to create an even more custom button (something that we can re-use). We can try to extend something from the existing classes.

```
class MyButton extends ElevatedButton {
  MyButton(String iconAsset, String label, Function()? onPressedCallback)
      : super( onPressed: onPressedCallback,
               child: Column(children: [
                                  Image.asset(iconAsset,
                                              width: 32,
                                              height: 32),
                                  Text(label)
                                ],
                           mainAxisSize: MainAxisSize.min)
                          )
         {}
}
```

Buttons

But what if we want to create an even more custom button (something that we can re-use). We can try to extend something from the existing classes.

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: MyButton(
                    "assets/images/yes_logo.png",
                    "Validate",
                    () => {})))));
  }
}
```



Buttons

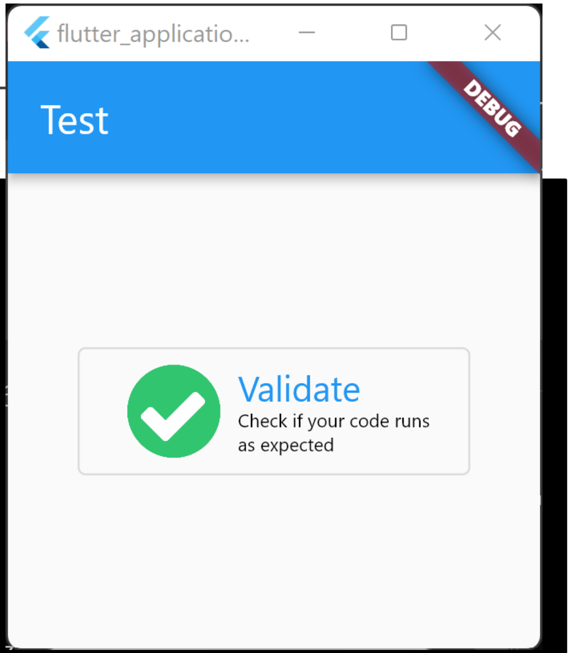
Using this technique, complex widgets can be created and reuse.

```dart
class MyButton extends OutlinedButton {
  MyButton(String iconAsset, String label, String label2, Function()? fnc)
      : super( onPressed: fnc,
            child: Row(children: [
              Image.asset(iconAsset, width: 64, height: 64),
              Container(
                  width: 100, height: 48,
                  child: Column(children: [
                    Container(width: 100, height: 22,
                              child: Text(label, textScaleFactor: 1.25)),
                    Text(label2, textScaleFactor: 0.66,
                         style: TextStyle(color: Colors.black))
                  ]))
            ], mainAxisSize: MainAxisSize.min)) {}
}
```

Buttons

Using this technique, complex widgets can be created and reuse.

```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: MyButton("assets/images/yes_logo.png",
                                "Validate",
                                "Check if your code runs as expected",
                                () => {})
            )));
  }
}
```



Buttons

Besides these three type of buttons, flutter framework also provides a round button (based on icon) called FloatingActionButton (that is more easily customizable).

```
FloatingActionButton ({Key? key,
                        Widget? child,
                        String? tooltip,
                        Color? foregroundColor,
                        Color? backgroundColor,
                        Color? focusColor,
                        Color? hoverColor,
                        Color? splashColor,
                        required VoidCallback? onPressed,…})
```

```
FloatingActionButton.extended(…)
```

```
FloatingActionButton.large(…)
```

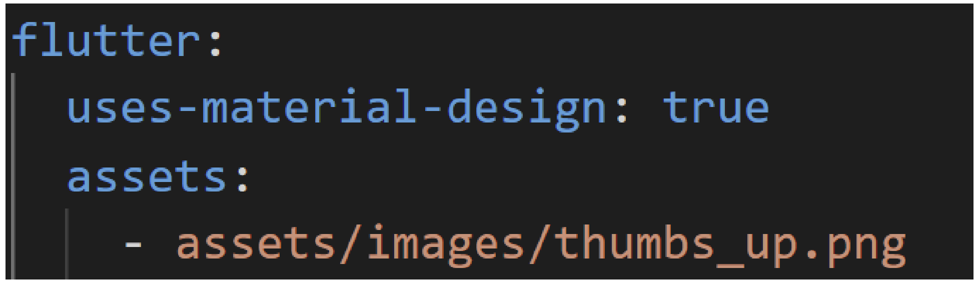
```
FloatingActionButton.small(…)
```

Buttons

Just like in the previous cases, let's add an image (a black and white icon) to our assets.

1. **First**, we need to add an image to our assets folder (let's name it 'thumbs_up.png'). Make sure that the image has transparency.

2. **Second**, add the new image to the pubspec.yaml file

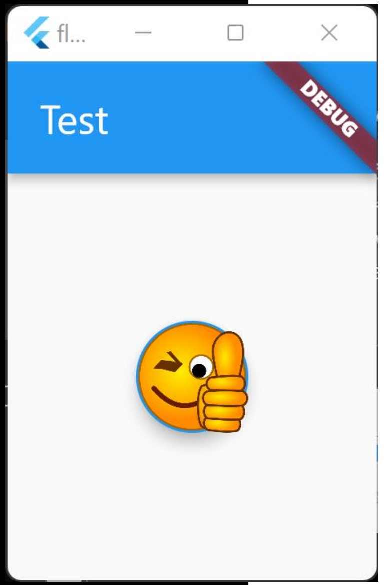


```yaml
flutter:
  uses-material-design: true
  assets:
    - assets/images/thumbs_up.png
```

Buttons

Then, let's create a simple FloatingActionButton using that icon.
If the image is bigger than the actual button, the image will be automatically scaled down to fit the button.

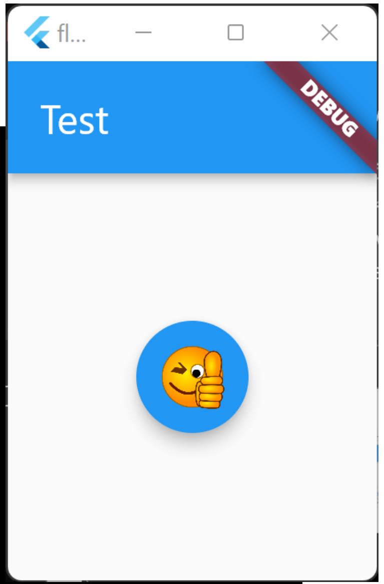
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: FloatingActionButton(
                    child: Image.asset("assets/images/thumbs_up.png"),
                    onPressed: () => {}))));
  }
}
```



Buttons

Then, let's create a simple FloatingActionButton using that icon. We can however, programmatically resize the icon to be smaller than the actual button:

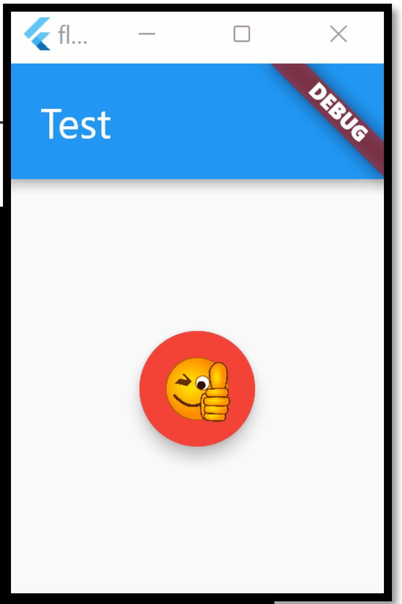
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: FloatingActionButton(
                    child: Image.asset("assets/images/thumbs_up.png",
                                        width: 32, height: 32),
                    onPressed: () => {}))));
  }
}
```



Buttons

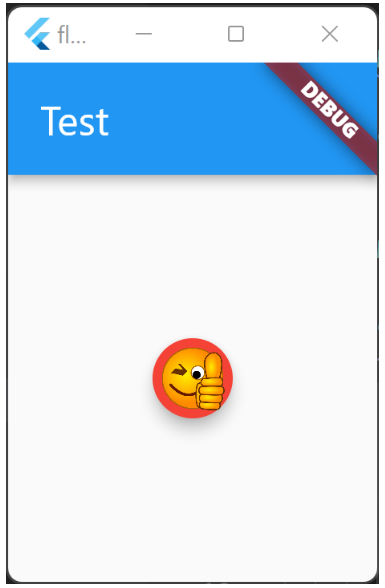
We can also customize the colors used for background or hover …

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(
              child: FloatingActionButton(
                  child: Image.asset("assets/images/thumbs_up.png",
                                      width: 32, height: 32),
                  onPressed: () => {},
                  backgroundColor: Colors.red,
                  hoverColor: Colors.orange,
            ))));
}
```

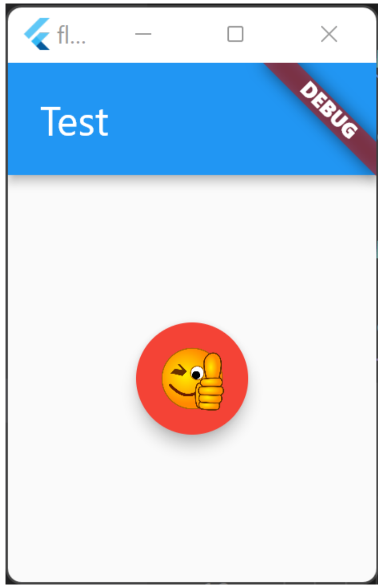


Buttons

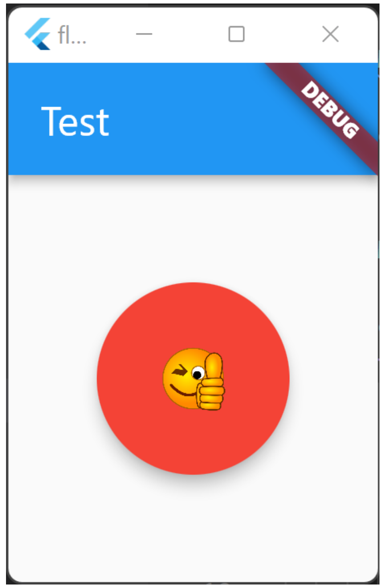
.small and .large named constructors make the button size smaller or bigger.



FloatingActionButton.**small**(...)



FloatingActionButton(...)

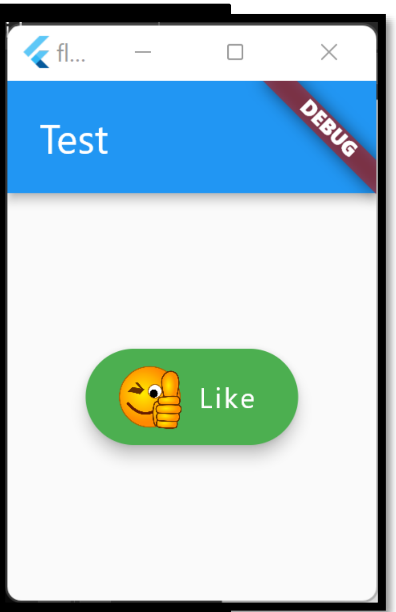


FloatingActionButton.**large**(...)

Buttons

The .extended named constructor can be used to create a button with text and icon.

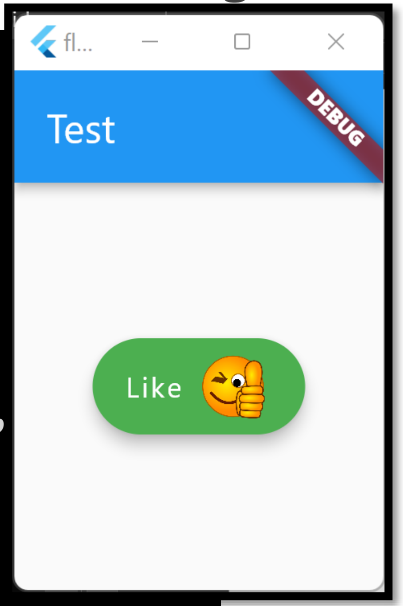
```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(
              child: FloatingActionButton.extended(
                  icon: Image.asset("assets/images/thumbs_up.png",
                                    width: 32, height: 32),
                  label: Text("Like"),
                  onPressed: () => {},
                  backgroundColor: Colors.green,
                  hoverColor: Colors.orange,
          ))));
}
```



Buttons

And as expected, since label and icon properties are widgets, they can be used with other widgets (in this example we've just reversed them to create a button where text is on the left and icon on the right

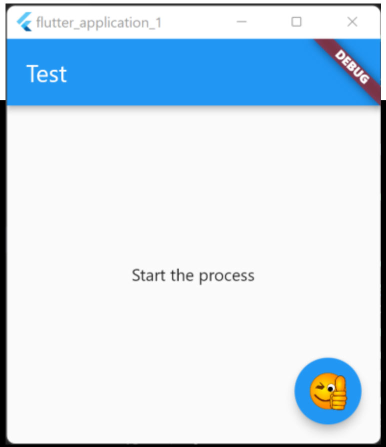
```
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: FloatingActionButton.extended(
                    label: Image.asset("assets/images/thumbs_up.png",
                                           width: 32, height: 32),
                    icon: Text("Like"),
                    onPressed: () => {},
                    backgroundColor: Colors.green,
                    hoverColor: Colors.orange))));
  }
```



Buttons

Finally, the Scaffold widget has a property called floatingActionButton that can be set up with such a bottom (that will be located at the bottom-right side of the application).

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(child: Text("Start the process")),
          floatingActionButton: FloatingActionButton(
                          child: Image.asset("assets/images/thumbs_up.png",
                                             width: 32, height: 32),
                           onPressed: () => {},
                           backgroundColor: Colors.blue,
          )));
}
```



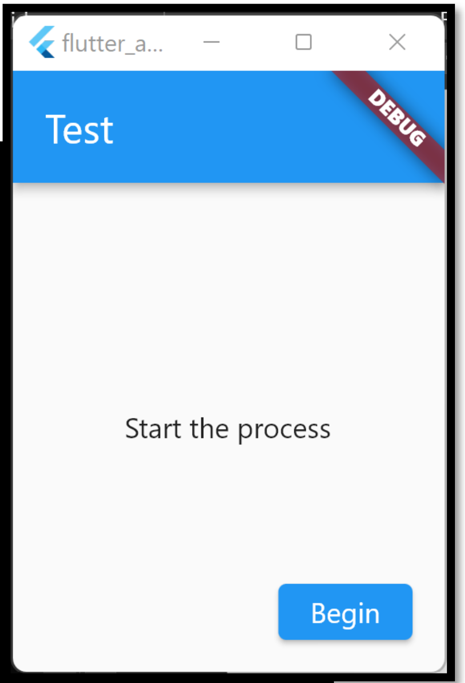
Start the process

Buttons

The floatingActionButton property is also a Widget. This mean that even if the name suggests that the widget used in this case should be a FloatingActionButton, in reality it could be any other widget. For example, the next example uses an ElevatedButton widget as the value of property floatingActionButton.

```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(child: Text("Start the process")),
          floatingActionButton: ElevatedButton(
                                      child: Text("Begin"),
                                      onPressed: () => {},

          )));
}
```



Buttons

Another type of button available on Flutter framework is IconButton. This type of button is similar to the previous ones, with the exception that it uses icons (glyphs) instead of images.

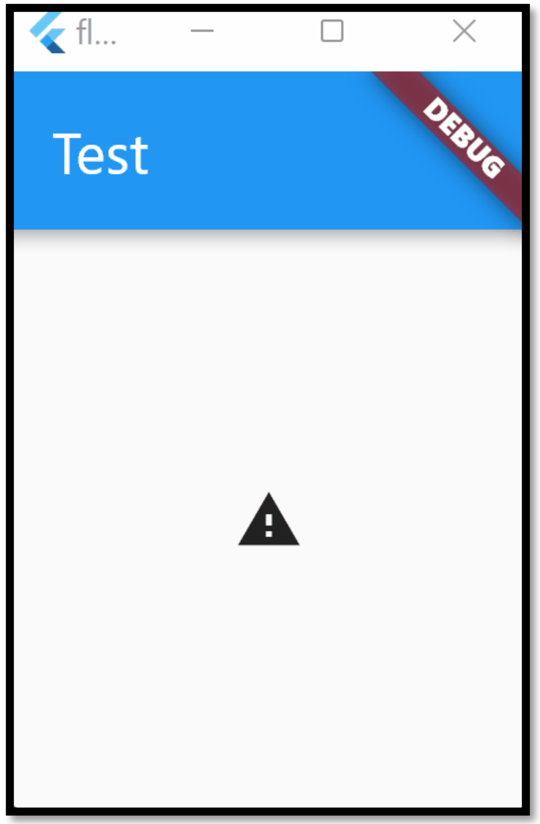
```
IconButton ({Key? key,
             required VoidCallback? onPressed,
             required Widget? Icon,
             double? iconSize,
             Color? color,
             Color? disabledColor,
             Color? focusColor,
             Color? hoverColor,
             Color? splashColor,
            ,…})
```

Just like previous cases, this type of button is more customizable, and it is designed to work with existing icons.

Buttons

The next example creates a very simple (centered) icon button with the alert sign.

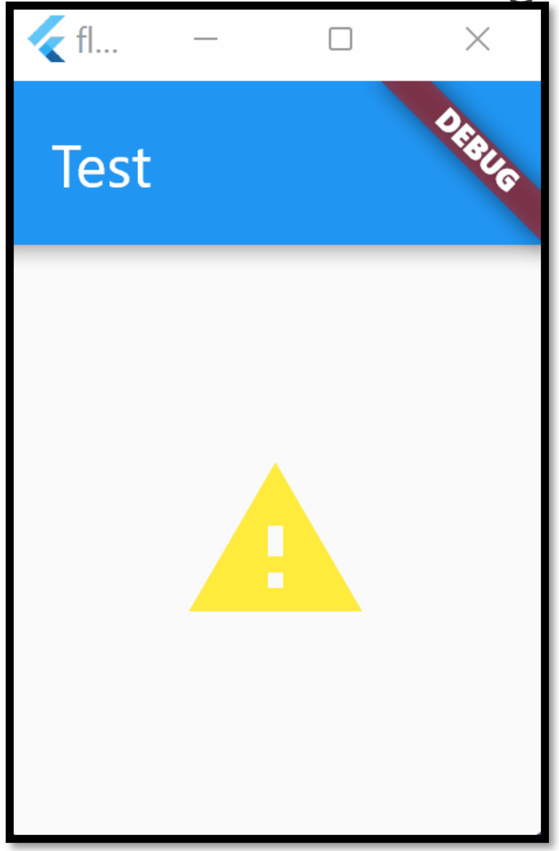
```
class MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                child: IconButton(
                    icon: Icon(Icons.warning),
                    onPressed: () => {}))));
  }
}
```



Buttons

The next example creates a customized icon button with a larger icon and red hovered background.

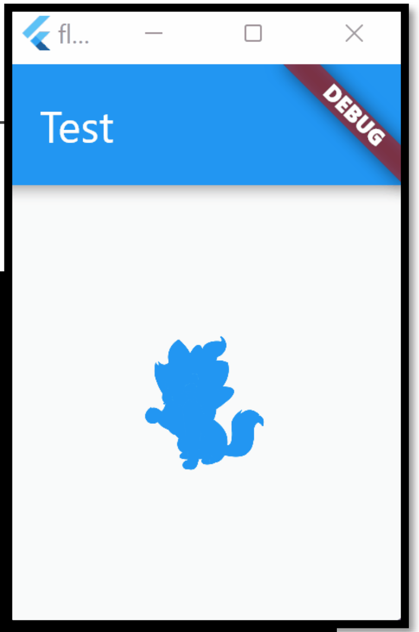
```
Widget build(BuildContext context) {
  return MaterialApp(
      home: Scaffold(
          appBar: AppBar(title: Text("Test")),
          body: Center(
              child: IconButton(
                    icon: Icon(Icons.warning),
                    onPressed: () => {},
                    iconSize: 64,
                    color: Colors.yellow,
                    hoverColor: Colors.red,
          ))));
}
```



Buttons

We cam also use a custom icon (via ImageIcon and AssetImage classes), with a size of 64 logical pixels, blue color and yellow hover color.

```
Widget build(BuildContext context) {
  return MaterialApp(
       home: Scaffold(
            appBar: AppBar(title: Text("Test")),
            body: Center(
                 child: IconButton(
                        icon: ImageIcon(AssetImage("assets/icons/cat.png")),
                        onPressed: () => {},
                        iconSize: 64,
                        color: Colors.blue,
                        hoverColor: Colors.yellow,
            ))));
}
```



Buttons

For convenience, there two extra buttons defined in Flutter:

1. CloseButton (a class that simulates an IconButton with the close icon)
2. BackButton (a class that simulates an IconButton with the back button)

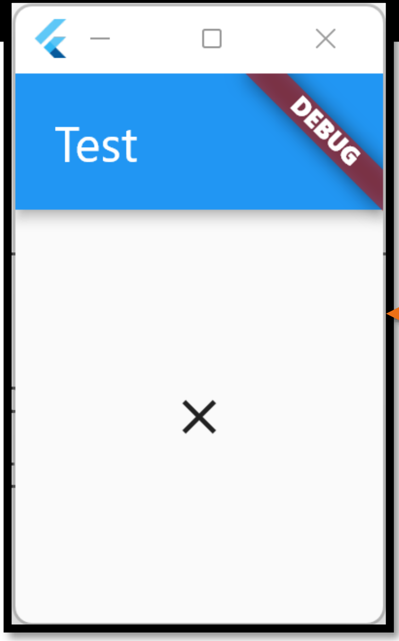
```
CloseButton ({Key? key, Color? color, void Function? onPressed})
BackButton ({Key? key, Color? color, void Function? onPressed})
```

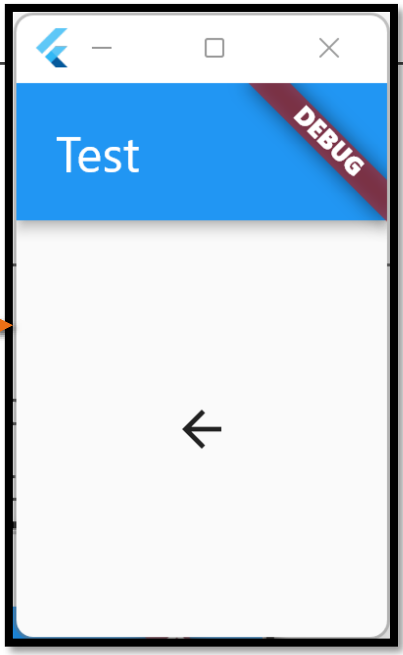
These types of button don't have so many customization → just the color and the onPressed callback.

However, since these types of buttons are often use, and they do have a clear usage and icon defined in material view, its easier to use this way.

Buttons

```
Widget build(BuildContext context) {
  return MaterialApp( home: Scaffold(
        appBar: AppBar(title: Text("Test")),
        body: Center(child: CloseButton())));
}
```





```
Widget build(BuildContext context) {
  return MaterialApp( home: Scaffold(
        appBar: AppBar(title: Text("Test")),
        body: Center(child: BackButton())));
}
```

Q & A