#### Get started

Samples & tutorials

#### <u>Development</u>

▼ User interface

Introduction to widgets

▼ Building layouts

Layouts in Flutter

**Tutorial** 

Creating responsive apps

<u>Understanding constraints [NEW]</u>

Box constraints

Adding interactivity

Assets and images

Navigation & routing

- Animations
- Advanced UI

Widget catalog

- Data & backend
- Accessibility & internationalization
- ▶ <u>Platform integration</u>
- ▶ Packages & plugins
- ▶ Add Flutter to existing app
- ▶ Tools & techniques
- Migration notes

Testing & debugging

# **Building layouts**

Docs > Development > UI > Layout > Tutorial

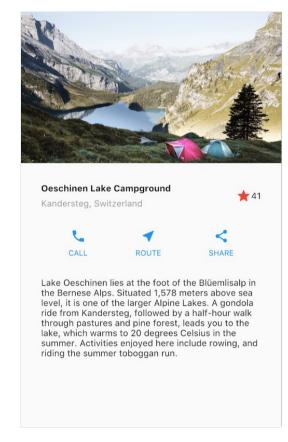
### Contents

- Step 0: Create the app base code
- Step 1: Diagram the layout
- Step 2: Implement the title row
- Step 3: Implement the button row
- Step 4: Implement the text section
- Step 5: Implement the image section
- Step 6: Final touch

### What you'll learn

- · How Flutter's layout mechanism works.
- How to lay out widgets vertically and horizontally.
- How to build a Flutter layout.

This is a guide to building layouts in Flutter. You'll build the layout for the following app:



The finished app

This guide then takes a step back to explain Flutter's approach to layout, and shows how to place a single widget on the screen. A discussion of how to lay widgets out horizontally and vertically, some of the most common layout widgets are covered.

If you want a "big picture" understanding of the layout mechanism, start with Flutter's approach to layout.

# Step 0: Create the app base code

Make sure to set up your environment, then do the following:

- 1. Create a basic "Hello World" Flutter app.
- 2. Change the app bar title and the app title as follows:

```
\blacksquare {codelabs/startup_namer/step1_base \rightarrow layout/base}/lib/main.dart
          @@ -10,10 +10,10 @@
               @override
  10
       10
  11
       11
               Widget build(BuildContext context) {
  12
       12
                 return MaterialApp(
  13
                   title: 'Welcome to Flutter',
                   title: 'Flutter layout demo',
                   home: Scaffold(
  14
      15
                     appBar: AppBar(
  15
                       title: Text('Welcome to Flutter'),
16
```

**=** 

<u>Get started</u>

Samples & tutorials

### <u>Development</u>

▼ <u>User interface</u>
Introduction to widgets

▼ Building layouts

Layouts in Flutter

**Tutorial** 

Creating responsive apps

<u>Understanding constraints [NEW]</u>

Box constraints

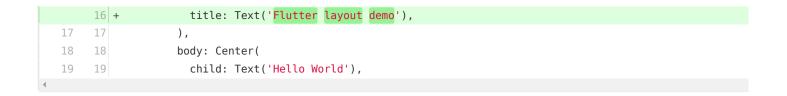
Adding interactivity

<u>Assets and images</u>

Navigation & routing

- Animations
- Advanced UIWidget catalog
- Data & backend
- ▶ Accessibility & internationalization
- ▶ Platform integration
- ▶ Packages & plugins
- ▶ Add Flutter to existing app
- ▶ Tools & techniques
- ▶ <u>Migration notes</u>

Testing & debugging

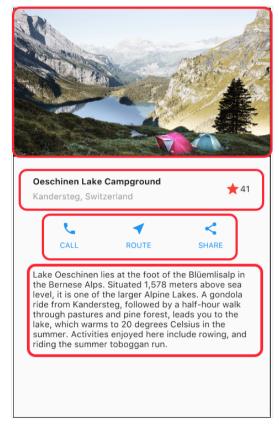


# Step 1: Diagram the layout

The first step is to break the layout down to its basic elements:

- · Identify the rows and columns.
- Does the layout include a grid?
- Are there overlapping elements?
- Does the UI need tabs?
- · Notice areas that require alignment, padding, or borders.

First, identify the larger elements. In this example, four elements are arranged into a column: an image, two rows, and a block of t

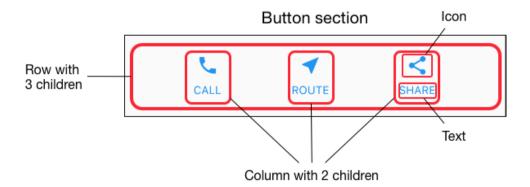


Column elements (circled in red)

Next, diagram each row. The first row, called the Title section, has 3 children: a column of text, a star icon, and a number. Its first child, the column, contains 2 lines of text. That first column takes a lot of space, so it must be wrapped in an Expanded widget.



The second row, called the Button section, also has 3 children: each child is a column that contains an icon and text.



Once the layout has been diagrammed, it's easiest to take a bottom-up approach to implementing it. To minimize the visual confu of deeply nested layout code, place some of the implementation in variables and functions.

### Step 2: Implement the title row

First, you'll build the left column in the title section. Add the following code at the top of the build() method of the MyApp class:

lib/main.dart (titleSection)

#### Get started

Samples & tutorials

#### <u>Development</u>

▼ User interface

Introduction to widgets

▼ Building layouts

Layouts in Flutter

**Tutorial** 

Creating responsive apps

<u>Understanding constraints [NEW]</u>

Box constraints

Adding interactivity

Assets and images

Navigation & routing

- Animations
- Advanced UI

Widget catalog

- Data & backend
- ▶ Accessibility & internationalization
- ▶ Platform integration
- ▶ Packages & plugins
- ▶ Add Flutter to existing app
- ▶ Tools & techniques
- Migration notes

Testing & debugging

```
Widget titleSection = Container(
 padding: const EdgeInsets.all(32),
 child: Row(
    children: [
      Expanded(
        /*1*/
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            /*2*/
            Container(
              padding: const EdgeInsets.only(bottom: 8),
              child: Text(
                'Oeschinen Lake Campground',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
              ),
            ),
            Text(
              'Kandersteg, Switzerland',
              style: TextStyle(
                color: Colors.grey[500],
              ),
            ),
          ],
        ),
      /*3*/
      Icon(
        Icons.star,
        color: Colors.red[500],
      Text('41'),
   ],
 ),
);
```

- /\*1\*/ Putting a Column inside an Expanded widget stretches the column to use all remaining free space in the row. Setting the crossAxisAlignment property to CrossAxisAlignment.start positions the column at the start of the row.
- /\*2\*/ Putting the first row of text inside a Container enables you to add padding. The second child in the Column, also text, displays as grey.
- /\*3\*/ The last two items in the title row are a star icon, painted red, and the text "41". The entire row is in a Container and pade along each edge by 32 pixels. Add the title section to the app body like this:

```
    {../base → step2}/lib/main.dart

          @ -12,11 +46,13 @@
  12
       46
                 return MaterialApp(
  13
       47
                  title: 'Flutter layout demo',
                  home: Scaffold(
  14
       48
  15
       49
                     appBar: AppBar(
                       title: Text('Flutter layout demo'),
  16
       50
  17
       51
  18
                     body: Center(
                       child: Text('Hello World'),
  19
       52 +
                     body: Column(
                       children: [
       54 +
                         titleSection,
       56
                    ),
       57
                  ),
  22
       58
                );
```

### Ţip:

- When pasting code into your app, indentation can become skewed. You can fix this in your Flutter editor using the automatic reformatting support.
- For a faster development experience, try Flutter's hot reload feature.
- If you have problems, compare your code to <a href="lib/main.dart">lib/main.dart</a>.

### Step 3: Implement the button row

The button section contains 3 columns that use the same layout—an icon over a row of text. The columns in this row are evenly spaced, and the text and icons are painted with the primary color.

Get started Samples & tutorials <u>Development</u>

▼ User interface

Introduction to widgets

▼ Building layouts

Layouts in Flutter

**Tutorial** 

Creating responsive apps

<u>Understanding constraints [NEW]</u>

Box constraints

Adding interactivity

<u>Assets and images</u>

Navigation & routing

- Animations
- ▶ Advanced UI

Widget catalog

- Data & backend
- ▶ Accessibility & internationalization
- ▶ Platform integration
- Packages & plugins
- ▶ Add Flutter to existing app
- ▶ Tools & techniques
- Migration notes

Testing & debugging

Since the code for building each column is almost identical, create a private helper method named buildButtonColumn(), which takes a color, an Icon and Text, and returns a column with its widgets painted in the given color.

```
lib/main.dart (_buildButtonColumn)
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
 Column _buildButtonColumn(Color color, IconData icon, String label) {
    return Column(
      mainAxisSize: MainAxisSize.min,
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(icon, color: color),
        Container(
          margin: const EdgeInsets.only(top: 8),
          child: Text(
            label,
            style: TextStyle(
              fontSize: 12,
              fontWeight: FontWeight.w400,
              color: color,
      ],
   );
```

The function adds the icon directly to the column. The text is inside a Container with a top-only margin, separating the text from

Build the row containing these columns by calling the function and passing the color, Icon, and text specific to that column. Align columns along the main axis using MainAxisAlignment.spaceEvenly to arrange the free space evenly before, between, and after each column. Add the following code just below the titleSection declaration inside the build() method:

```
lib/main.dart (buttonSection)
Color color = Theme.of(context).primaryColor;
Widget buttonSection = Container(
 child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      _buildButtonColumn(color, Icons.call, 'CALL'),
      _buildButtonColumn(color, Icons.near_me, 'ROUTE'),
      _buildButtonColumn(color, Icons.share, 'SHARE'),
   ],
 ),
);
```

Add the button section to the body:

```
{step2 → step3}/lib/main.dart
         @@ -46,3 +59,3 @@
  46
                return MaterialApp(
  47
      60
                 title: 'Flutter layout demo',
                  home: Scaffold(
          @@ -52,8 +65,9 @@
  52
                   body: Column(
                     children: [
                       titleSection,
                        buttonSection,
                      ],
  56
       70
                    ),
                  ),
                );
              }
```

# Step 4: Implement the text section

Define the text section as a variable. Put the text in a Container and add padding along each edge. Add the following code just be the buttonSection declaration:

<u>Get started</u>

<u>~</u>

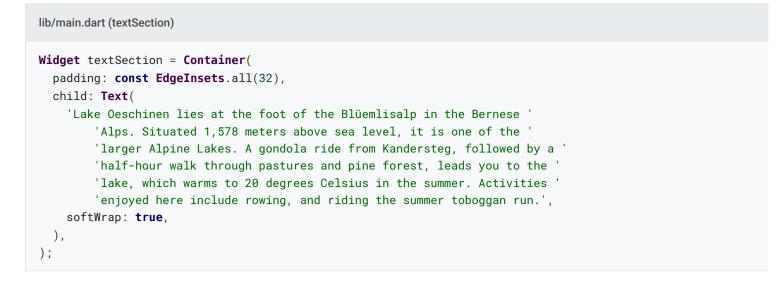
#### <u>Development</u>

▼ <u>User interface</u>

Samples & tutorials

- Introduction to widgets
- ▼ Building layouts
  - Layouts in Flutter
  - **Tutorial**
  - Creating responsive apps
  - <u>Understanding constraints [NEW]</u>
  - Box constraints
  - Adding interactivity
- Assets and images
- Navigation & routing
- Animations
- Advanced UI
  - Widget catalog
- Data & backend
- Accessibility & internationalization
- ▶ Platform integration
- Packages & plugins
- ▶ Add Flutter to existing app
- ▶ Tools & techniques
- ▶ Migration notes

Testing & debugging



By setting softwrap to true, text lines will fill the column width before wrapping at a word boundary.

Add the text section to the body:

```
a {step3 → step4}/lib/main.dart
      @@ -59,3 +72,3 @@
 59
     72
              return MaterialApp(
      73
                title: 'Flutter layout demo',
  61 74
                home: Scaffold(
     @@ -66,6 +79,7 @@
 66
                    children: [
                      titleSection,
 68 81
                      buttonSection,
                      textSection,
 69 83
                   ],
      84
                  ),
  71 85
                ),
```

# Step 5: Implement the image section

Three of the four column elements are now complete, leaving only the image. Add the image file to the example:

- Create an images directory at the top of the project.
- Add <u>lake.jpg</u>.

• Note that wget doesn't work for saving this binary file. The original image is <u>available online</u> under a Creative Commons license, but it's large and slow to fetch.

• Update the pubspec.yaml file to include an assets tag. This makes the image available to your code.

#### ♥ Tip:

- Note that pubspec.yaml is case sensitive, so write assets: and the image URL as shown above.
- The pubspec file is also sensitive to white space, so use proper indentation.

Now you can reference the image from your code:

```
a {step4 → step5}/lib/main.dart
          @a -77,6 +77,12 @a
  77
                    ),
  78
                    body: Column(
  79
                      children: [
       79
                         Image.asset(
       81 +
                           'images/lake.jpg',
                           width: 600,
                          height: 240,
                           fit: BoxFit.cover,
                         titleSection,
```

#### **Get started**

Samples & tutorials

### <u>Development</u>

▼ <u>User interface</u>

Introduction to widgets

▼ Building layouts

Layouts in Flutter

**Tutorial** 

<u>Creating responsive apps</u>

<u>Understanding constraints [NEW]</u>

Box constraints

Adding interactivity

Assets and images

Navigation & routing

- Animations
- Advanced UI

Widget catalog

- ▶ Data & backend
- ▶ <u>Accessibility & internationalization</u>
- ▶ Platform integration
- ▶ Packages & plugins
- Add Flutter to existing app
- ▶ Tools & techniques
- ▶ <u>Migration notes</u>

Testing & debugging



BoxFit.cover tells the framework that the image should be as small as possible but cover its entire render box.

## Step 6: Final touch

In this final step, arrange all of the elements in a ListView, rather than a Column, because a ListView supports app body scrolling when the app is run on a small device.

```
a {step5 → step6}/lib/main.dart
          @@ -72,13 +77,13 @@
       77
                return MaterialApp(
                  title: 'Flutter layout demo',
  73
       78
       79
                  home: Scaffold(
  75
       80
                    appBar: AppBar(
       81
                      title: Text('Flutter layout demo'),
  76
  77
       82
                    ),
  78
                    body: Column(
                    body: ListView(
       83 +
                      children: [
  79
       84
  80
       85
                        Image.asset(
                          'images/lake.jpg',
  81
       86
                          width: 600,
  82
       87
  83
       88
                          height: 240,
                          fit: BoxFit.cover,
  84
       89
```

Dart code: main.dart
Image: images

Pubspec: pubspec.yaml

That's it! When you hot reload the app, you should see the same app layout as the screenshot at the top of this page.

You can add interactivity to this layout by following Adding Interactivity to Your Flutter App.



<u>flutter-dev@・terms・security・privacy・español</u>・社区中文资源

Except as otherwise noted, this work is licensed under a Creative Commons Attribution 4.0 International License, and code samples are licensed under the BSD License.