Extracted from:

Programming Flutter

Native, Cross-Platform Apps the Easy Way

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Programming Flutter

Native, Cross-Platform Apps the Easy Way

Carmine Zaccagnino
edited by Michael Swaine

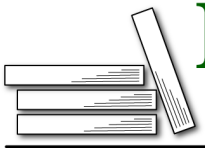
Programming Flutter

Native, Cross-Platform Apps the Easy Way

Carmine Zaccagnino

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

The team that produced this book includes:

Publisher: Andy Hunt
VP of Operations: Janet Furlow
Executive Editor: Dave Rankin
Development Editor: Michael Swaine
Copy Editor: Jasmine Kwityn
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-695-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2020

Implement the Basic Layout

Inside _CalculatorHomePageState's curly braces, we'll define a build method with the usual Scaffold and AppBar:

layout/calculator_part1/lib/main.dart

```
@override
Widget build(BuildContext context) {
```

```
return Scaffold(
  appBar: AppBar(
    title: Text(widget.title),
  ),

  );
}
```

The Scaffold's body will be the following:

```
body: Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: <Widget>[
    calculatorScreen,
    Row(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: <Widget>[

      ],
    ),
    Row(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: <Widget>[
        Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[
            Row(
              crossAxisAlignment:  CrossAxisAlignment.stretch,
              children: <Widget>[

              ],
            ),
            Row(
              crossAxisAlignment:  CrossAxisAlignment.stretch,
              children:<Widget>[

              ],
            ),
            Row(
              crossAxisAlignment:  CrossAxisAlignment.stretch,
              children: <Widget>[

              ],
            ),
            Row(
              crossAxisAlignment:  CrossAxisAlignment.stretch,
              children: <Widget>[

              ],
            ),
          ],
        ),
```

```
        Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: <Widget>[

          ],
        ),
      ],
    ),
  ],
),
```

The CrossAxisAlignment options are meant to stretch the widgets out wherever possible.

What that alone can't do will be done by Expanded (we'll talk about Expanded after we have defined the layout elements).

Let's Give the Calculator a Screen

Obviously the calculatorScreen at the top of the Column is just a placeholder; we need to replace it with a real, working screen that will be composed of the following: a wrapping and a Card and, inside it, a padded Text displaying a String, which represents the expression or result to be displayed.

But, before adding that, we need to declare that String inside the _CalculatorHomePageState declaration, above the @override keyword preceding the build method declaration:

layout/calculator_part1/lib/main.dart

```
String _str = "0";
```

After you've done that, replace calculatorScreen (from the code on page 6) with:

layout/calculator_part1/lib/main.dart

```
Card(
  color: Colors.lightGreen[50],
  child: Padding(
    padding: EdgeInsets.all(15.0),
    child: Text(
      _str,
      textScaleFactor: 2.0,
    ),
  ),
),
```

Where Colors.lightGreen[50] is a very low intensity light green.

Color intensity (50, in this case) is specified on a scale where 500 is normal intensity (like using Colors.lightGreen) and higher numbers are higher intensity colors, usually used for accent color purposes.

Add the Deletion Row

The first Row widget is going to contain the button used to delete a single character and the button used to delete the entire expression.

All of the buttons in the app are going to call _CalculatorHomePageState methods, so we need to define them. So, directly below code on page 7, add the following empty method definitions, which we will fill up after we define the class to handle the calculations in Use the Calculation Inside the App, on page ?:

layout/calculator_part1/lib/main.dart

```
void add(String a) {
}

void deleteAll() {
}

void deleteOne() {
}

void getResult() {
}
```

Going back to app layout definition, the buttons will be FlatButtons, which you'll add to the first Row widget you can see in the code on page 6 by inserting, inside the square brackets of that Row's children attribute, the following code:

layout/calculator_part1/lib/main.dart

```
FlatButton(
  child: Text(
    'C',
    style: TextStyle(color: Colors.white),
  ),
  onPressed: (){deleteAll();},
  color: Colors.black54,
),
FlatButton(
  child: Text(
    '<-',
    style: TextStyle(color: Colors.white)
  ),
  onPressed: (){deleteOne();},
  color: Colors.black87,
),
```

Using this code will result in the button on the left (delete all) being gray and the one on the right (delete one character) being black.

That is because the number after the color name (which is used in this manner only with black and white) specifies how transparent they are; a low

number means that they are very transparent and almost invisible and a high number is used to make something very opaque.

The Third Row

We have one more Row, which might sound strange until you look back at the structure of the app in the figure on page ?. That row will have two children:

- On the left, the numbers and the *dot* and *equals* buttons.
- On the right, the operator buttons.

Add the Number Buttons

The number buttons will be spread out across four rows:

- The first will contain the numbers 7, 8, and 9.
- The second will contain the numbers 4, 5, and 6.
- The third will contain the numbers 1, 2, and 3.
- The fourth will contain the number 0, the *dot*, and the button to calculate the result.

This means that, inside this Row we'll need another Column with four Rows as its children just to display the number buttons. The buttons will have white text on an intense blue background, except for the = button, which will have black text on a very light blue background. So, as the first child of the third Row of our app, we'll have the following:

layout/calculator_part1/lib/main.dart

```
Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: <Widget>[
    Row(
      crossAxisAlignment:  CrossAxisAlignment.stretch,
      children: <Widget>[
        FlatButton(
          child: Text(
            '7',
            style: TextStyle(color: Colors.white),
          ),
          onPressed: () {add('7');},
          color: Colors.blueAccent,
        ),
        FlatButton(
          child: Text(
            '8',
            style: TextStyle(color: Colors.white),
          ),
          onPressed: () {add('8');},
```

```
      color: Colors.blueAccent,
    ),
    FlatButton(
      child: Text(
        '9',
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {add('9');},
      color: Colors.blueAccent,
    ),
  ],
),
Row(
  crossAxisAlignment:  CrossAxisAlignment.stretch,
  children:<Widget>[
    FlatButton(
      child: Text(
        '4',
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {add('4');},
      color: Colors.blueAccent,
    ),
    FlatButton(
      child: Text(
        '5',
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {add('5');},
      color: Colors.blueAccent,
    ),
    FlatButton(
      child: Text(
        '6',
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {add('6');},
      color: Colors.blueAccent,
    ),
  ],
),
Row(
  crossAxisAlignment:  CrossAxisAlignment.stretch,
  children: <Widget>[
    FlatButton(
      child: Text(
        '1',
        style: TextStyle(color: Colors.white),
      ),
      onPressed: () {add('1');},
```

```
          color: Colors.blueAccent,
        ),
        FlatButton(
          child: Text(
            '2',
            style: TextStyle(color: Colors.white),
          ),
          onPressed: () {add('2');},
          color: Colors.blueAccent,
        ),
        FlatButton(
          child: Text(
            '3',
            style: TextStyle(color: Colors.white)
          ),
          onPressed: () {add('3');},
          color: Colors.blueAccent,
        ),
      ],
    ),
    Row(
      crossAxisAlignment:  CrossAxisAlignment.stretch,
      children: <Widget>[
        FlatButton(
          child: Text(
            '0',
            style: TextStyle(color: Colors.white),
          ),
          onPressed: () {add('0');},
          color: Colors.blueAccent,
        ),
        FlatButton(
          child: Text(
            '.',
            style: TextStyle(color: Colors.white),
          ),
          onPressed: () {add('.');},
          color: Colors.blueAccent,
        ),
        FlatButton(
          child: Text('='),
          onPressed: () {
            getResult();
          },
          color: Colors.blue[50],
        ),
      ],
    ),
  ],
),
```

As you can see, = calls getResult() and the numbers call the add(number) method.

Add the Operators

To the right of the numbers, we'll add the operator buttons.

As an example, we'll implement the *divide* button as a FlatButton with an image from the assets on it.

To start, create a directory in the root of your project called icons.

Inside it, create or copy a file called divide.png, which will be the icon used for the divide button. You can find a divide.png icon suitable for this in the book's source code or on my website.[1]

After you've done that, edit pubspec.yaml and add the following at the bottom of it:

layout/calculator/pubspec.yaml

```
flutter:
  assets:
    - icons/
```

Now we can use that image in our code.

To use it, we need to use Image.asset as explained in Displaying Images from the Assets, on page ?.

More specifically for this example, the divide button will be implemented as follows:

layout/calculator_part1/lib/main.dart

```
FlatButton(
  child: Image.asset(
    "icons/divide.png",
    width: 10.0,
    height: 10.0,
  ),
  onPressed: () {add('÷');},
  color: Colors.blue[50],
),
```

The entire Column of operator buttons (with the other ones being normal Text-based buttons) will therefore be:

1. http://www.carminezacc.com/divide.png

layout/calculator_part1/lib/main.dart

```
Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: <Widget>[
    FlatButton(
      child: Image.asset(
        "icons/divide.png",
        width: 10.0,
        height: 10.0,
      ),
      onPressed: () {add('÷');},
      color: Colors.blue[50],
    ),
    FlatButton(
      child: Text('x'),
      onPressed: () {add('x');},
      color: Colors.blue[50],
    ),
    FlatButton(
      child: Text('-'),
      onPressed: () {add('-');},
      color: Colors.blue[50],
    ),
    FlatButton(
      child: Text('+'),
      onPressed: () {add('+');},
      color: Colors.blue[50],
    ),
  ],
),
```

The buttons are of the same color as the = button (a very light blue).

Why a GridView Wouldn't Work

GridViews are meant for something completely different: building scrollable, probably variable grids that usually have more widgets than could be displayed comfortably in a non-scrollable grid.

What we are doing here is simply displaying a small, fixed amount of I/O widgets in both the vertical and horizontal axis, which you should do using nested Rows and Columns.

Make the App Look and Work Right with Expanded

Currently, the app wouldn't look right if you ran it on a device.

To make the buttons of the right shape and size, we need to wrap Rows, Columns, and the buttons themselves in Expanded.

Expand Elements to Divide the Space Inside the Main Column

We'll expand some elements to make them exactly as big as we want them to be.

In this case, the main `Column` will be divided in the following way:

- 2/7 of the space will be taken up by the calculator screen.
- 1/7 of the space will be taken up by the deletion row.
- The remaining 4/7 of the space (four times as much as the deletion row) will be taken up by the numbers and operators, so that every button (including the delete buttons) is of the same height.

As was explained in Fill the Space Available in the View Using Expanded, on page ?, the `Expanded` widget makes this really simple, requiring you to replace the calculator screen shown in the code on page 7 with this:

layout/calculator_part2/lib/main.dart

```
Expanded(
  flex: 2,
  child: Card(
    color: Colors.lightGreen[50],
    child: Padding(
      padding: EdgeInsets.all(15.0),
      child: Text(
        _str,
        textScaleFactor: 2.0,
      ),
    ),
  ),
),
```

Similarly, the first `Row` widget (the one containing the delete buttons, shown in the code on page 8) will have to be replaced with:

```
Expanded(
  flex: 1,
  child: Row(
    ...
  )
)
```

and the one containing the number buttons (shown in the code on page 9) will similarly have to be replaced with:

```
Expanded(
  flex: 4,
  child: Row(
    ...
  )
)
```

Expand Elements to Divide the Space Inside the Deletion Row

Three quarters of the space will be taken up by the *delete all* button and one quarter by the *delete one* button.

This means that those two FlatButtons will have to be expanded by replacing them with FlatButtons wrapped in two Expanded widgets with flex values of, respectively, 3 and 1:

layout/calculator_part2/lib/main.dart

```
Expanded(
  flex: 3,
  child: FlatButton(
    child: Text(
      'C',
      style: TextStyle(color: Colors.white),
    ),
    onPressed: (){deleteAll();},
    color: Colors.black54,
  ),
),
Expanded(
  flex: 1,
  child: FlatButton(
    child: Text(
      '<-',
      style: TextStyle(color: Colors.white)
    ),
    onPressed: (){deleteOne();},
    color: Colors.black87,
  ),
),
```

Expand Elements to Divide the Space Inside the Number and Operator Grid

Similarly, you will need to expand the numbers' Column with a flex of 3 and the operators' Column with a flex of 1 to make the buttons all of the same size.

Create ExpandedRow and ExpandedButton for the Rest of the Buttons

The number and operator buttons should all be of the same size, so they should all be wrapped in Expanded widgets with the same flex which, for simplicity, we'll set to the value 1.

The same is true for the Rows of number buttons: they all need to be expanded with the same flex.

To avoid repetition and to speed up replacement of many widgets, we'll build expanded FlatButton and Row classes, which will be called ExpandedButton and ExpandedRow.

They need to take all of the arguments and options we currently use, and build the corresponding widgets wrapped in Expanded.

They are defined, very simply, above main(), in the following way:

layout/calculator_part2/lib/main.dart

```
class ExpandedButton extends StatelessWidget {

  ExpandedButton({this.onPressed, this.child, this.color});

  final Widget child;
  final VoidCallback onPressed;
  final Color color;

  @override
  Widget build(BuildContext context) =>
    Expanded(
      flex:1,
      child: FlatButton(
        onPressed: onPressed,
        child: child,
        color: color,
      ),
    );
}

class ExpandedRow extends StatelessWidget {

  ExpandedRow({this.children, this.crossAxisAlignment});

  final List<Widget> children;
  final CrossAxisAlignment crossAxisAlignment;

  @override
  Widget build(BuildContext context) =>
    Expanded(
      flex:1,
      child: Row(
        children: children,
        crossAxisAlignment: crossAxisAlignment,
      ),
    );
}
```

After defining them, simply replace all of the operator button's Row widgets with ExpandedRow widgets and all of the operator and number FlatButtons and the calculator's layout should be complete.