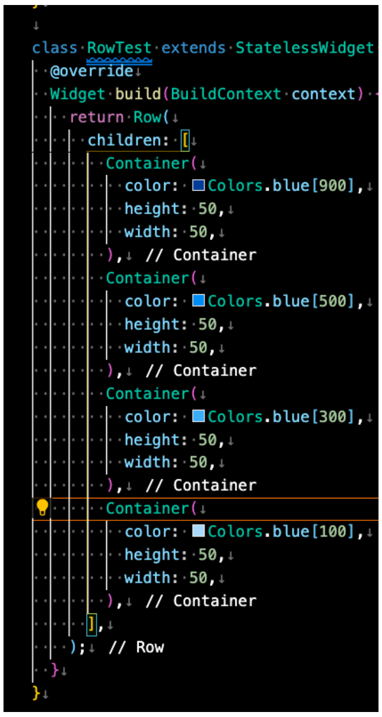
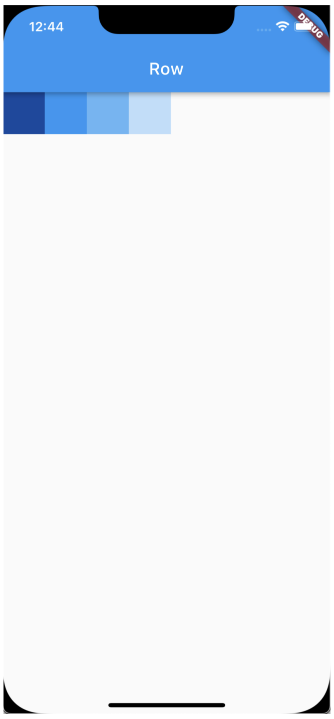
Lab Two Code Examples

Rows/Columns:
Rows and columns are important widgets for laying out widgets vertically (Columns), and horizontally (Rows). There are two important properties to Rows and Columns, the MainAxis and the CrossAxis. The MainAxis is vertical for Columns and is horizontal for Rows, you can think of it as the main axis along which widgets will be positioned. Then there is also the CrossAxis which is the axis that widgets will not be placed along. We will further look at rows and the MainAxis and CrossAxis below (note Columns will not be looked at as they are identical to Rows just having the MainAxis and CrossAxis swapped).

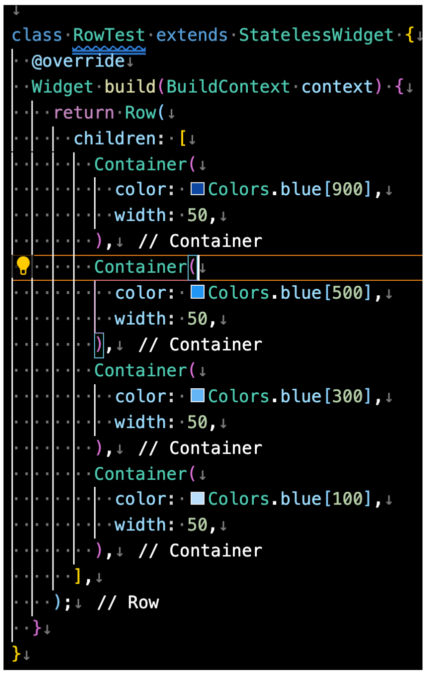
Example One: Using Row and giving explicit sizes for the height & width of the containers.



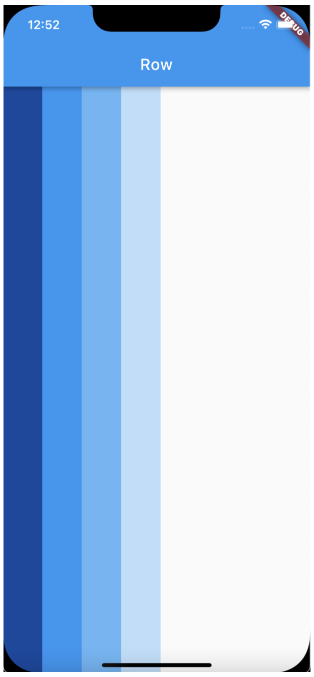
```dart
class RowTest extends StatelessWidget
  @override
  Widget build(BuildContext context)
    return Row(
      children: [
        Container(
          color: Colors.blue[900],
          height: 50,
          width: 50,
        ), // Container
        Container(
          color: Colors.blue[500],
          height: 50,
          width: 50,
        ), // Container
        Container(
          color: Colors.blue[300],
          height: 50,
          width: 50,
        ), // Container
        Container(
          color: Colors.blue[100],
          height: 50,
          width: 50,
        ), // Container
      ],
    ); // Row
  }
}
```



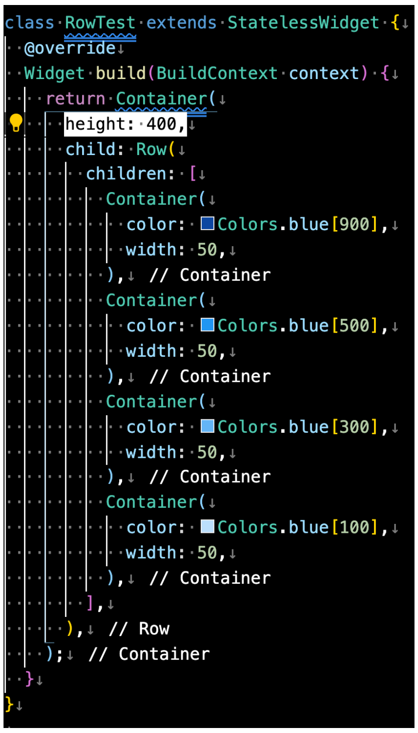
Example Two: Importantly, width for each child widget needs to be given when using a row, however, in the case that the CrossAxisSize (or Height) is not given, this will be taken as the maximum value for height its parent widget has. If the children widgets all have defined heights then the rows height is determined to be the largest height of all the children widgets.



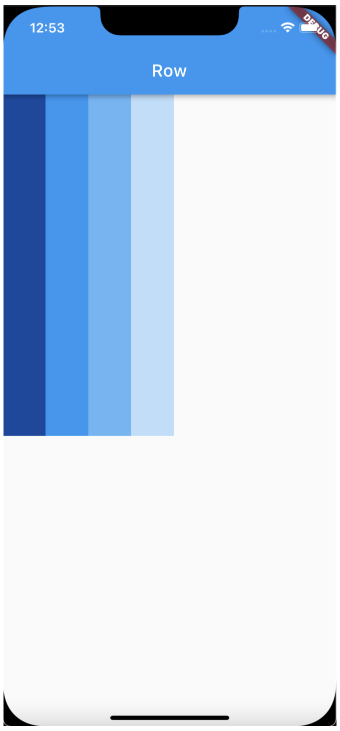
```
class RowTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Container(
          color: Colors.blue[900],
          width: 50,
        ), // Container
        Container(
          color: Colors.blue[500],
          width: 50,
        ), // Container
        Container(
          color: Colors.blue[300],
          width: 50,
        ), // Container
        Container(
          color: Colors.blue[100],
          width: 50,
        ), // Container
      ],
    ); // Row
  }
}
```



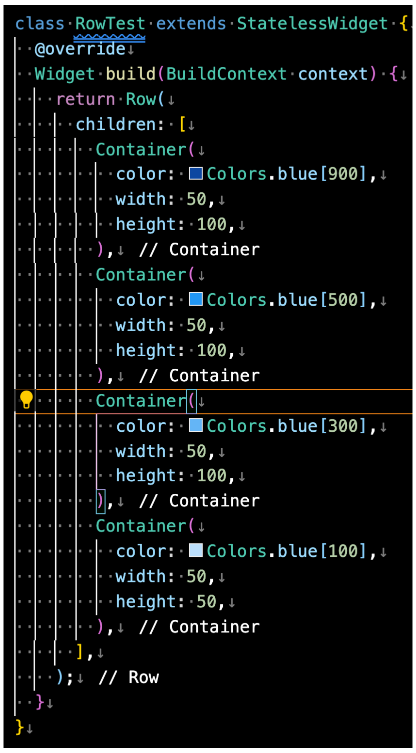
In this example, since no height is provided the height defaults to take all the available space it can.



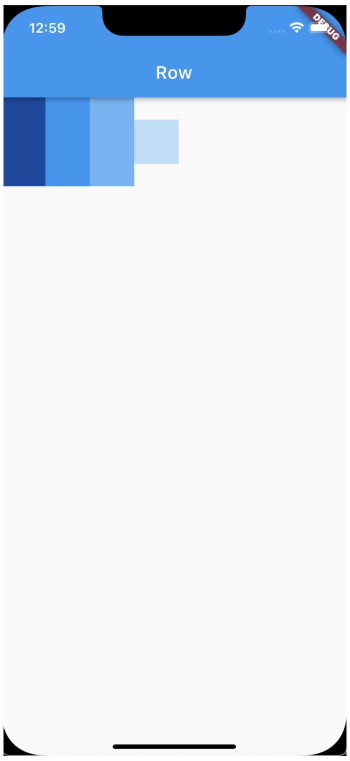
```
class RowTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      height: 400,
      child: Row(
        children: [
          Container(
            color: Colors.blue[900],
            width: 50,
          ), // Container
          Container(
            color: Colors.blue[500],
            width: 50,
          ), // Container
          Container(
            color: Colors.blue[300],
            width: 50,
          ), // Container
          Container(
            color: Colors.blue[100],
            width: 50,
          ), // Container
        ],
      ), // Row
    ); // Container
  }
}
```



In this example the parent of the Row is given an explicit height of 400, so importantly the CrossAxisSize will be 400. Even if heights are provided in all the containers (children of Row), the CrossAxisSize will still be 400. The only difference that will be seen (if all children have heights) is that the size of the Containers will change, provided they are smaller than 400, otherwise they will be treated in the case above as if no height is provided (i.e., larger values then the given CrossAxisSize will be capped to the maximum size).

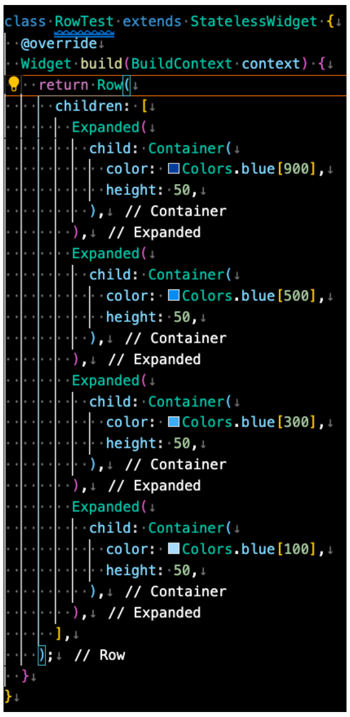


```dart
class RowTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Container(
          color: Colors.blue[900],
          width: 50,
          height: 100,
        ), // Container
        Container(
          color: Colors.blue[500],
          width: 50,
          height: 100,
        ), // Container
        Container(
          color: Colors.blue[300],
          width: 50,
          height: 100,
        ), // Container
        Container(
          color: Colors.blue[100],
          width: 50,
          height: 50,
        ), // Container
      ],
    ); // Row
  }
}
```

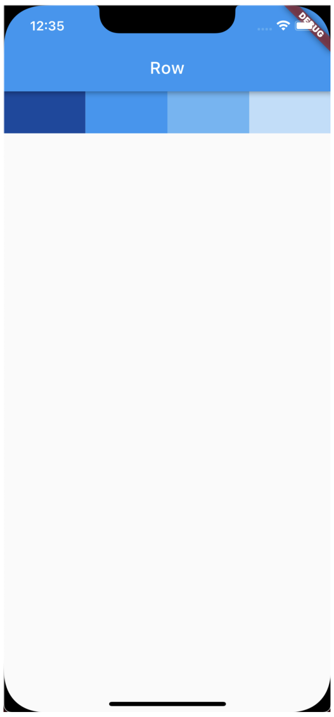


In this example the last container has a height of 50 (all others have 100), since the largest widget is 100, the widget of size 50 is centered (Rows have a predefined CrossAxisAlignment to center the row along the cross axis) within the rows defined CrossAxisSize (height) of 100.

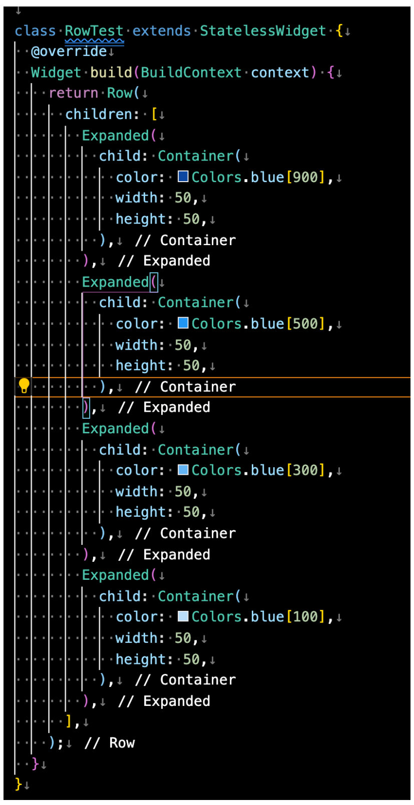
Example Three: In this example we are using a Row and giving explicit sizes for the height of the containers, then we are wrapping the containers with Expanded so when in a row they expand (or take all available space) along the main axis of the Row (horizontal).



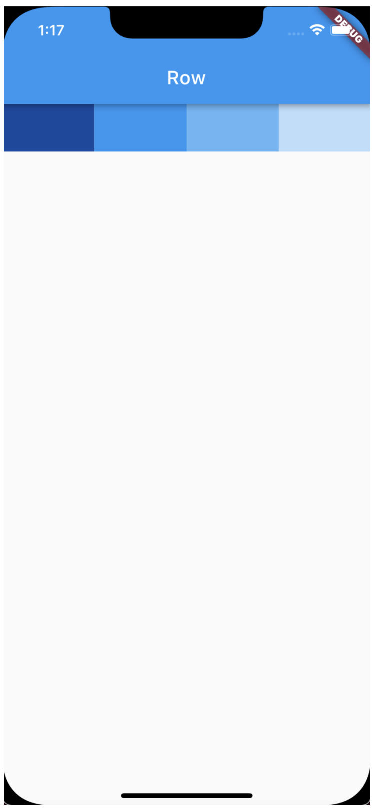
```dart
class RowTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Expanded(
          child: Container(
            color: Colors.blue[900],
            height: 50,
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            color: Colors.blue[500],
            height: 50,
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            color: Colors.blue[300],
            height: 50,
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            color: Colors.blue[100],
            height: 50,
          ), // Container
        ), // Expanded
      ],
    ); // Row
  }
}
```



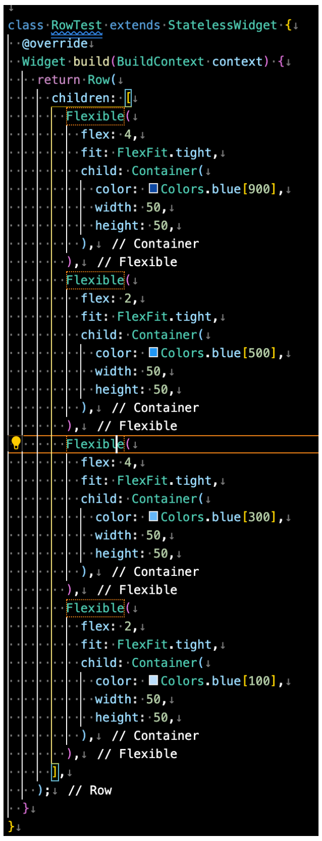
(IMPORTANT CONCEPT!)
Example Four: When you use Expanded but still provide explicit sizes to children widget along the MainAxis, the explicit sizes given to children will be ignored.



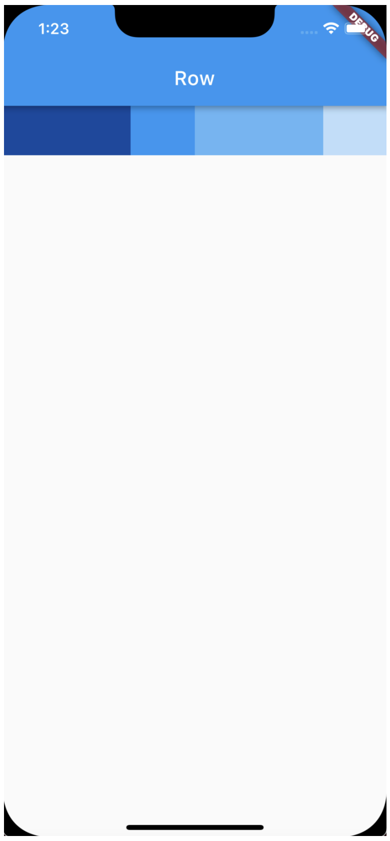
```
class RowTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Expanded(
          child: Container(
            color: Colors.blue[900],
            width: 50,
            height: 50,
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            color: Colors.blue[500],
            width: 50,
            height: 50,
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            color: Colors.blue[300],
            width: 50,
            height: 50,
          ), // Container
        ), // Expanded
        Expanded(
          child: Container(
            color: Colors.blue[100],
            width: 50,
            height: 50,
          ), // Container
        ), // Expanded
      ],
    ); // Row
  }
}
```



Example Five: Flexible is another widget very similar Expanded, however, there is alot more you can do with a Flexible widget. Flexible allows you to determine what portion of the remaining available screen each widget wrapped with should get.



```
class RowTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Flexible(
          flex: 4,
          fit: FlexFit.tight,
          child: Container(
            color: Colors.blue[900],
            width: 50,
            height: 50,
          ), // Container
        ), // Flexible
        Flexible(
          flex: 2,
          fit: FlexFit.tight,
          child: Container(
            color: Colors.blue[500],
            width: 50,
            height: 50,
          ), // Container
        ), // Flexible
        Flexible(
          flex: 4,
          fit: FlexFit.tight,
          child: Container(
            color: Colors.blue[300],
            width: 50,
            height: 50,
          ), // Container
        ), // Flexible
        Flexible(
          flex: 2,
          fit: FlexFit.tight,
          child: Container(
            color: Colors.blue[100],
            width: 50,
            height: 50,
          ), // Container
        ), // Flexible
      ],
    ); // Row
  }
}
```



(IMPORTANT CONCEPT!)
To summarize, for a row widget the CrossAxisSize (Height) is determined in a couple of ways:

1) The first most important determinant is whether there is an explicit height provided by the parent widget, if there is then the Row's CrossAxisSize will be based on that.
2) If there is no height provided by the parent widget to the Row, then if all children of the row have a height provided, the CrossAxisSize will be based on the child with the largest height provided.
3) Finally, if no height is provided by the parent widget to the Row, and not all children have a defined height, then the CrossAxisSize will be based on how much space is available to display the widget in the current context.

Finally, for a row width the MainAxisSize (Width) must be explicitly defined in some way, here are some ways you can do this:

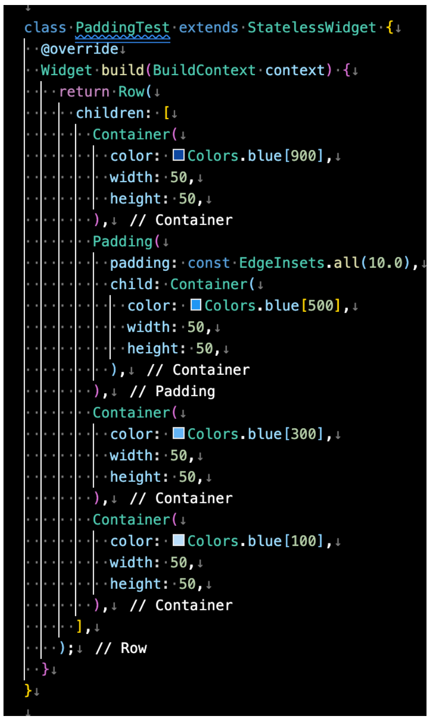
1) Give explicit sizes for the widths of children widgets
2) Expand each or apply a flexible and give a unique flex value to size it.

Note: Again we will not go over Columns in this document, as they work identically to Rows, only the CrossAxis and MainAxis have been swapped.

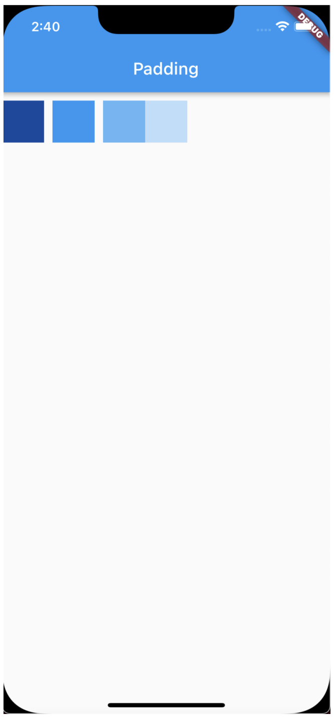
Padding:

A way to add spacing around widgets. There is three main ways to provide padding to a widget we will go over those in three examples:

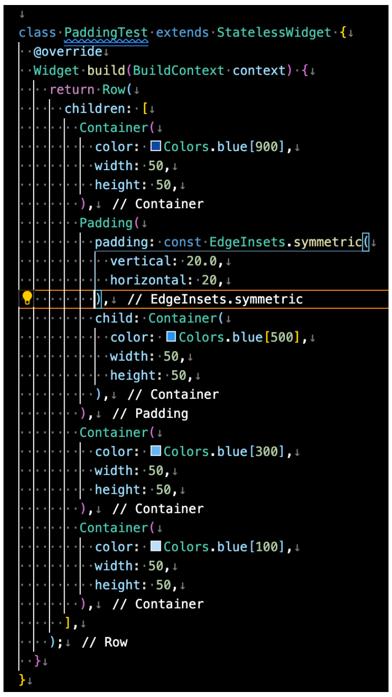
Example One: You can use EdgeInsets.all(double value) to provide a specific padding value around the whole widget (top, bottom, left and right).



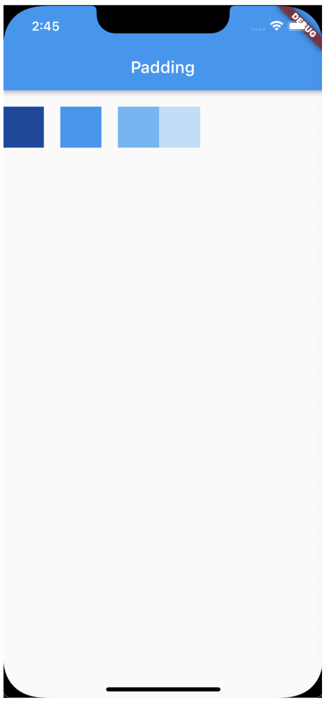
```dart
class PaddingTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Container(
          color: Colors.blue[900],
          width: 50,
          height: 50,
        ), // Container
        Padding(
          padding: const EdgeInsets.all(10.0),
          child: Container(
            color: Colors.blue[500],
            width: 50,
            height: 50,
          ), // Container
        ), // Padding
        Container(
          color: Colors.blue[300],
          width: 50,
          height: 50,
        ), // Container
        Container(
          color: Colors.blue[100],
          width: 50,
          height: 50,
        ), // Container
      ],
    ); // Row
  }
}
```



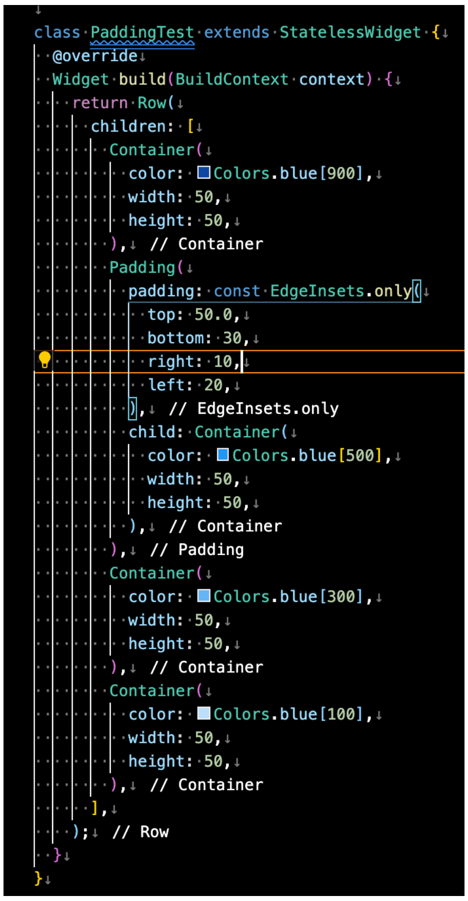
Example Two: You can use EdgeInsets.symmetric(vertical: double value, horizontal: double value) to provide specific padding values vertically (top and bottom) or horizontally (left and right). Note importantly you can also choose to only apply padding to one of vertical or horizontal (e.g., EdgeInsets.only(horizontal: double value), this only applies padding to the left and right of a widget).



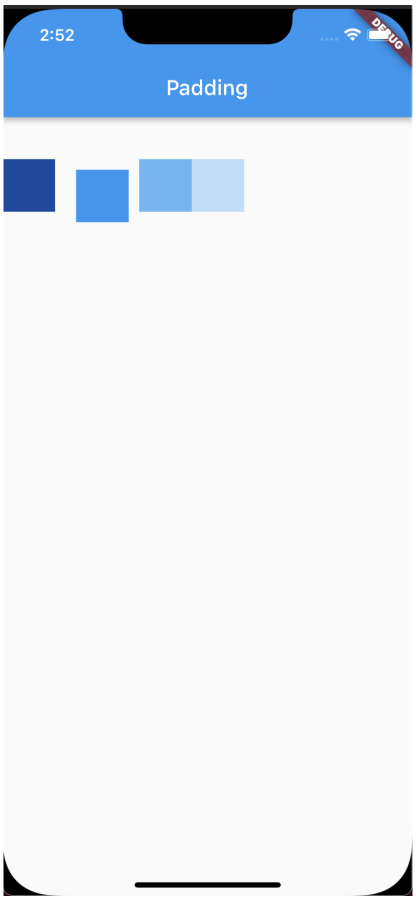
```dart
class PaddingTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Container(
          color: Colors.blue[900],
          width: 50,
          height: 50,
        ), // Container
        Padding(
          padding: const EdgeInsets.symmetric(
            vertical: 20.0,
            horizontal: 20,
          ), // EdgeInsets.symmetric
          child: Container(
            color: Colors.blue[500],
            width: 50,
            height: 50,
          ), // Container
        ), // Padding
        Container(
          color: Colors.blue[300],
          width: 50,
          height: 50,
        ), // Container
        Container(
          color: Colors.blue[100],
          width: 50,
          height: 50,
        ), // Container
      ],
    ); // Row
  }
}
```



Example Three: You can use EdgeInsets.only(top: double value, bottom: double value, left: double value, right: double value,) to provide specific padding values individually to each specific value around an object. Note importantly you can also opt to only to apply padding to one position (e.g., EdgeInsets.only(top: double val), this only applies padding to the top of a widget).

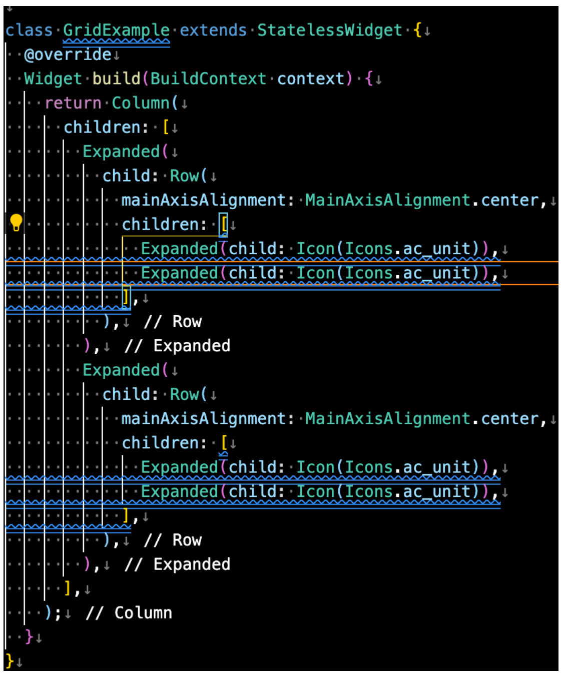


```
class PaddingTest extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      children: [
        Container(
          color: Colors.blue[900],
          width: 50,
          height: 50,
        ), // Container
        Padding(
          padding: const EdgeInsets.only(
            top: 50.0,
            bottom: 30,
            right: 10,
            left: 20,
          ), // EdgeInsets.only
          child: Container(
            color: Colors.blue[500],
            width: 50,
            height: 50,
          ), // Container
        ), // Padding
        Container(
          color: Colors.blue[300],
          width: 50,
          height: 50,
        ), // Container
        Container(
          color: Colors.blue[100],
          width: 50,
          height: 50,
        ), // Container
      ],
    ); // Row
  }
}
```

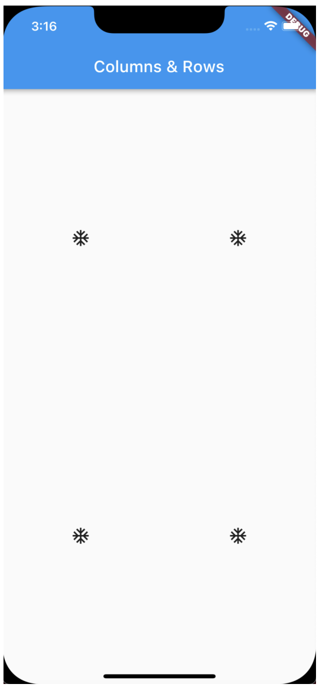


Using Rows and Columns together:

As a reminder, Rows and Columns both take a list of widgets with their parameter called Children, and importantly Rows and Columns are Widgets!!! What does this mean, well if you wanted to or needed to, you could have a Column with as many rows as you wanted and if you wanted those rows could also have Columns and so on. When you think of Rows and Columns, I want you to think of them as widgets that place other widgets horizontally, and vertically respectively, and start by breaking down an interface into layout components. Below I show an example where I place snowflake icons in a 2x2 grid.



```
class GridExample extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Expanded(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Expanded(child: Icon(Icons.ac_unit)),
              Expanded(child: Icon(Icons.ac_unit)),
            ],
          ), // Row
        ), // Expanded
        Expanded(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Expanded(child: Icon(Icons.ac_unit)),
              Expanded(child: Icon(Icons.ac_unit)),
            ],
          ), // Row
        ), // Expanded
      ],
    ); // Column
  }
}
```


3:16
DEBUG
Columns & Rows