ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

502071

Cross-Platform Mobile Application Development

Layout Widgets

1

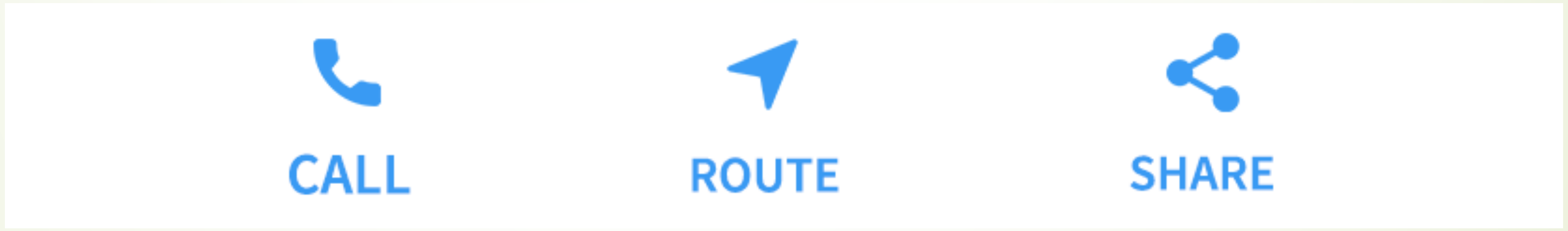
Flutter Layout

Flutter Layout

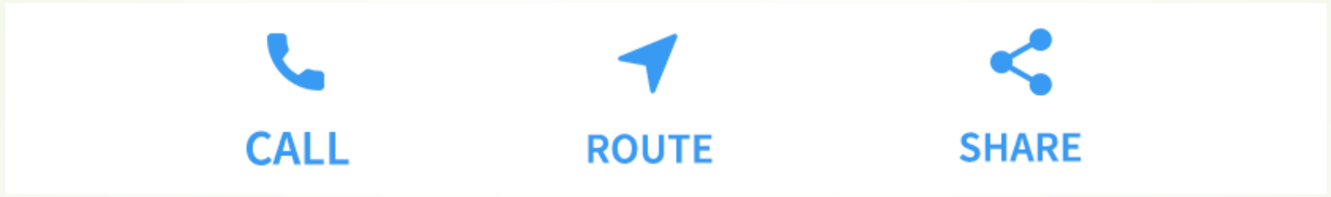
- Flutter layout refers to the way elements in a Flutter app are arranged and organized on the screen. It involves determining the size, position, and arrangement of widgets within a user interface, with the goal of creating a visually appealing and functional app.
- In Flutter, the layout is built using widgets, and it's a central aspect of the framework. By providing a wide range of pre-built widgets and flexible layout options, Flutter makes it easy for developers to create beautiful and functional user interfaces.
- A well-designed layout can improve user experience, enhance the usability of the app, and make the app visually appealing. On the other hand, a poorly designed layout can lead to confusion, frustration, and a poor overall experience for the user.
- Therefore, it's important to give proper consideration to the layout in Flutter app development, as it can have a significant impact on the success of the app.

Flutter Layout

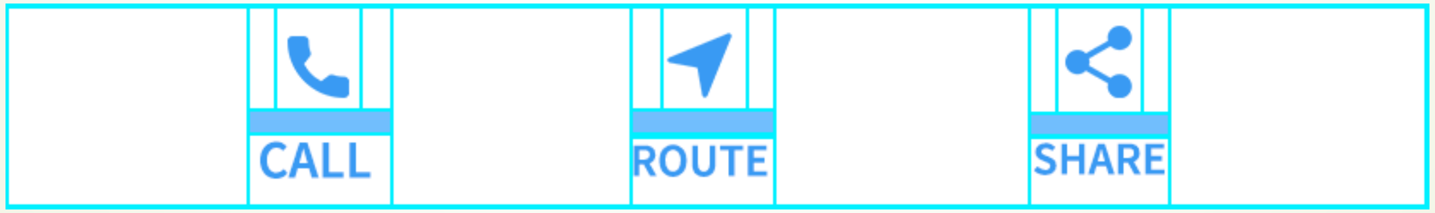
- The Flutter framework provides a rich set of widgets and tools that make it easy to create complex and dynamic layouts: Container, Row, Column, Expanded, Wrap, Flow, Stack, Card, ListView, GridView.
- For a better understanding of the concept let's take a single example and break down those components for better understanding.



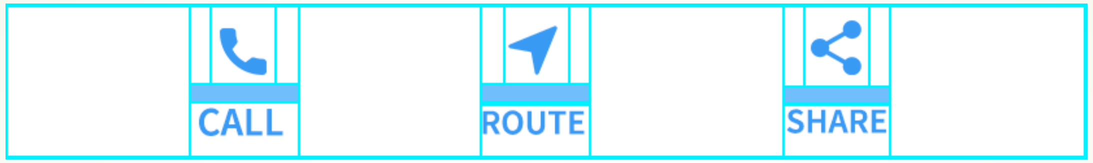
Flutter Layout

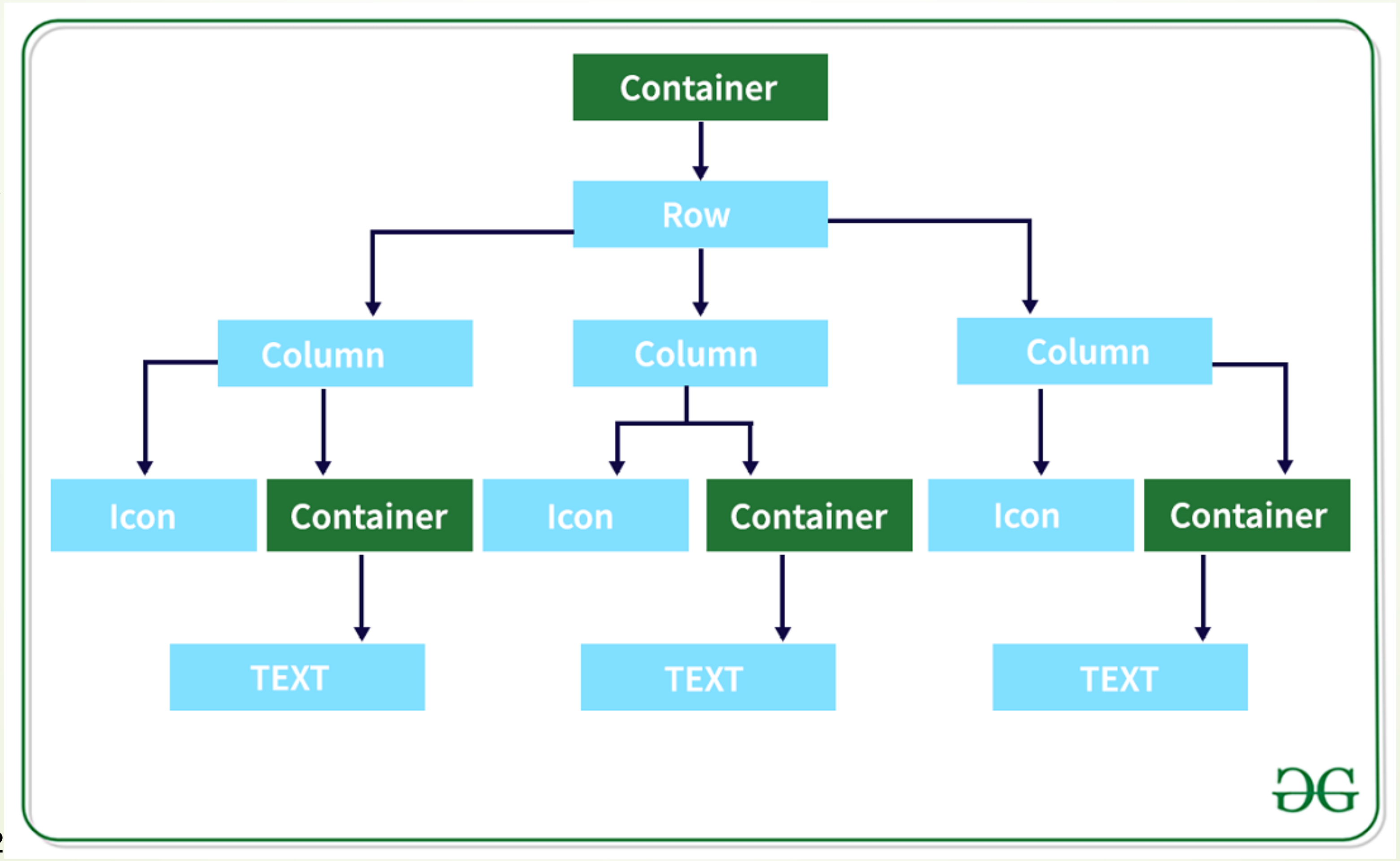


- In the above image, you just saw a layout that is nothing but just a composition of few basic widgets.
- Now look at the above image here we just outline the layouts, look closely you can see inside a row widget there are 3 column widgets and each column contains an icon and a label. Take a look at the below widget tree diagram.



Flutter Layout





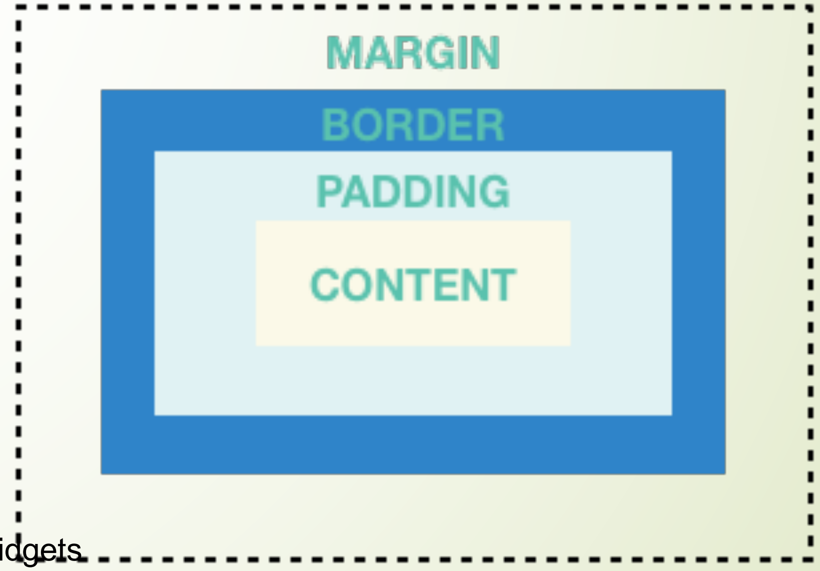
Flutter Layout

- The common layout widget can be divided into two categories: Standard Widgets and Material Widgets.
- Standard widgets in flutter are Container, GridView, ListView, Stack.
- Material widgets in flutter are Card, ListTile, AppBar, TabBar, Drawere, BottomNavigationBar.

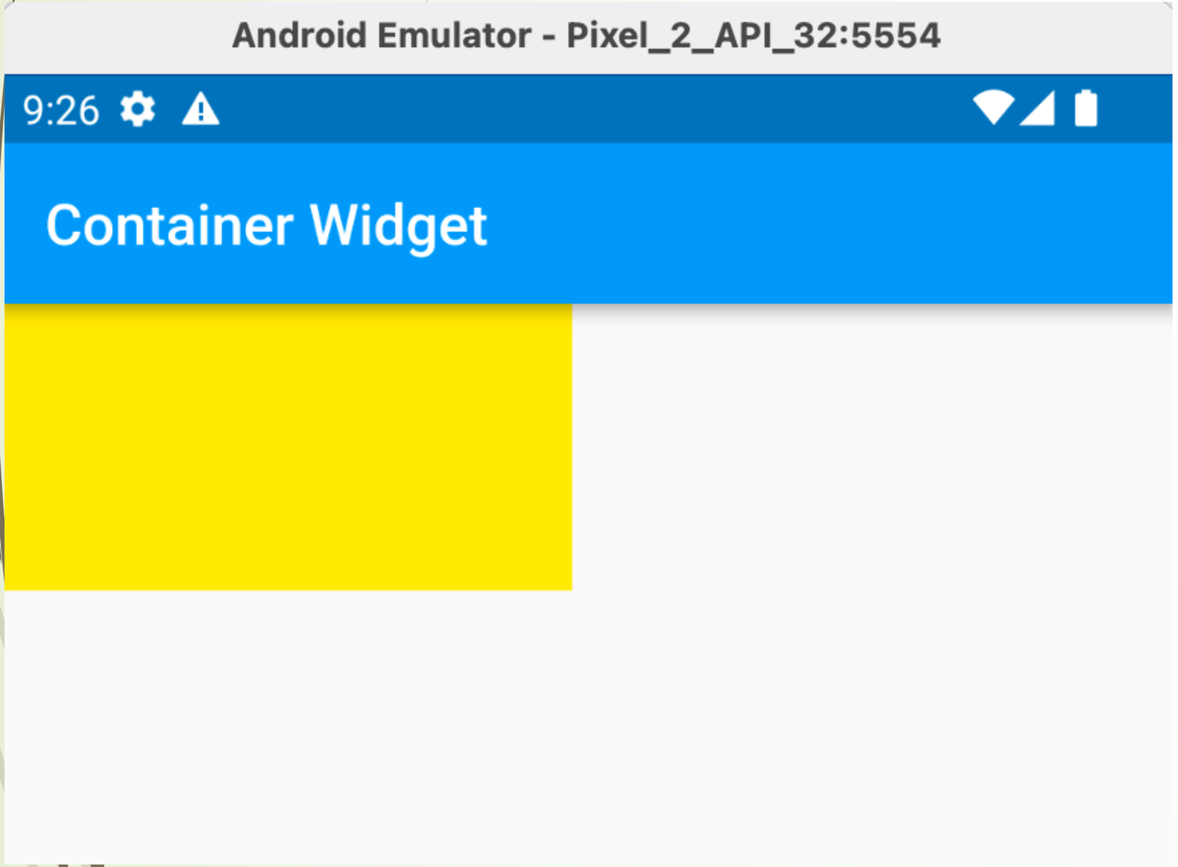
Common Layout Widgets

Container widget

- The container is the most used widget in flutter, It contains only a single widget or child.
- We can add padding, margin, border, and background color-like properties in this widget and we can customize it according to our requirement.
- The Container widget in Flutter and the div tag in HTML serve similar purposes. They are both used to group other widgets or elements together and provide a way to apply common styling and layout properties to those elements.

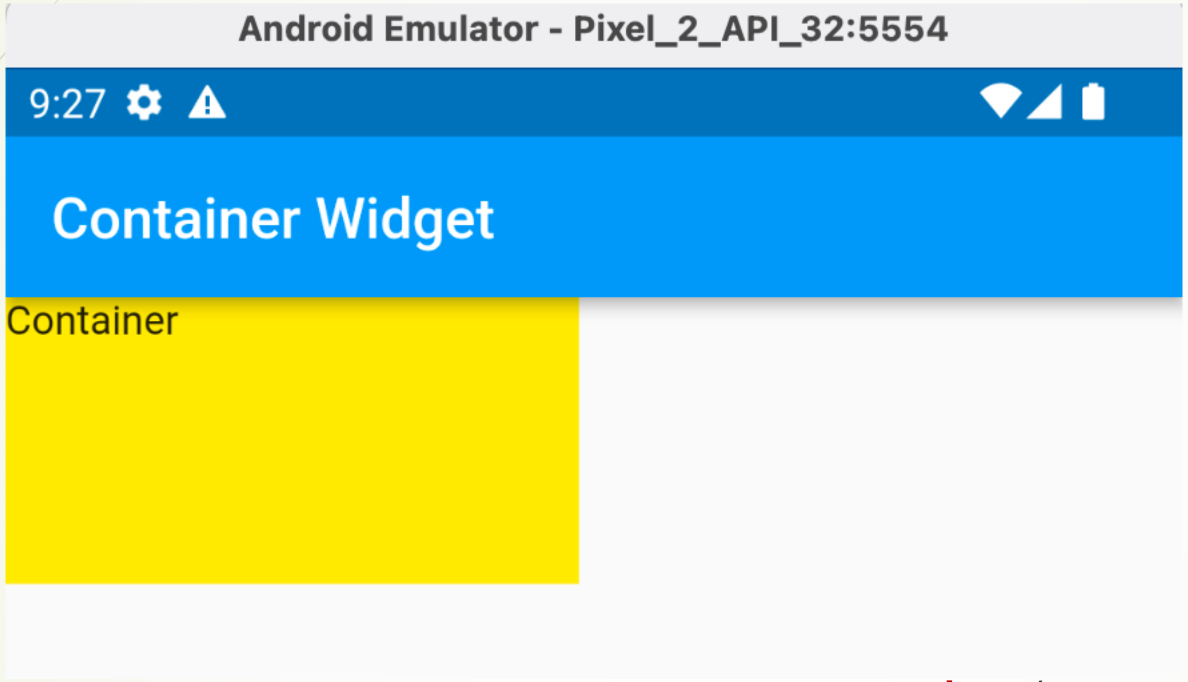


Container widget



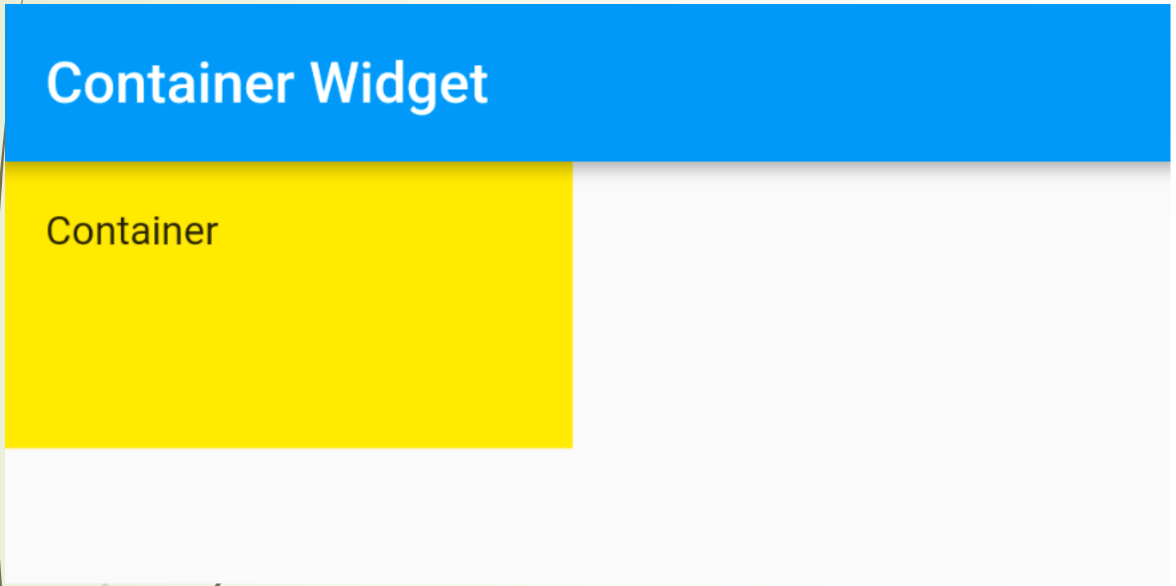
```
Scaffold(
    appBar: AppBar(title: Text('Container Widget')),
    body: Container(width: 150, height: 80,
            color: Colors.yellow)
),
```

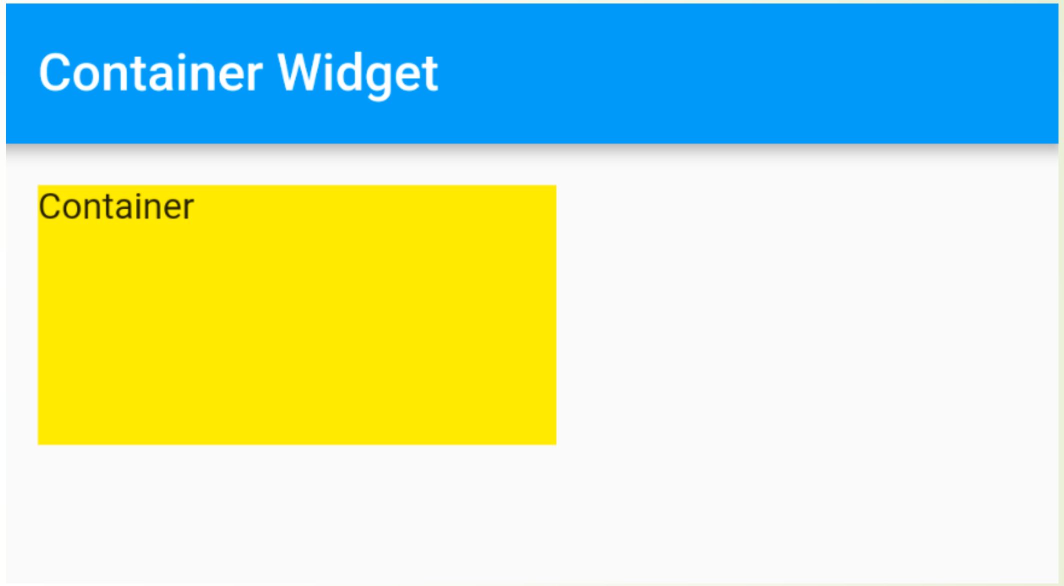
Container widget



```
Container(width: 200, height: 100,
    color: Colors.yellow,
    child: const Text('Container'),
)
```

Container widget



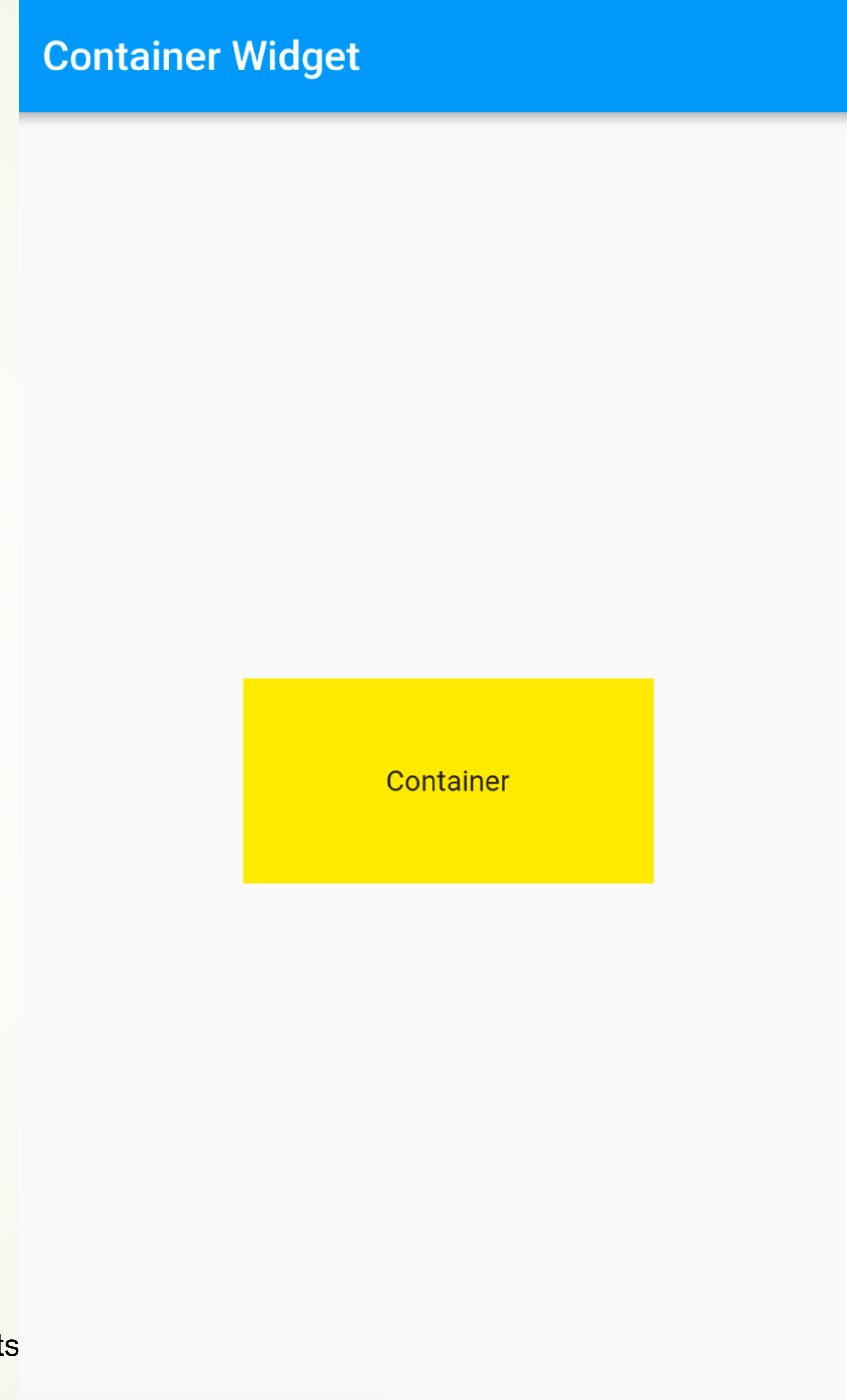


```
Container(width: 200, height: 100,
    padding: const EdgeInsets.all(16),
    color: Colors.yellow,
    child: const Text('Container'),
)
```

```
Container(width: 200, height: 100,
    color: Colors.yellow,
    margin: EdgeInsets.only(left: 16, top: 16),
    child: const Text('Container'),)
```

Container widget

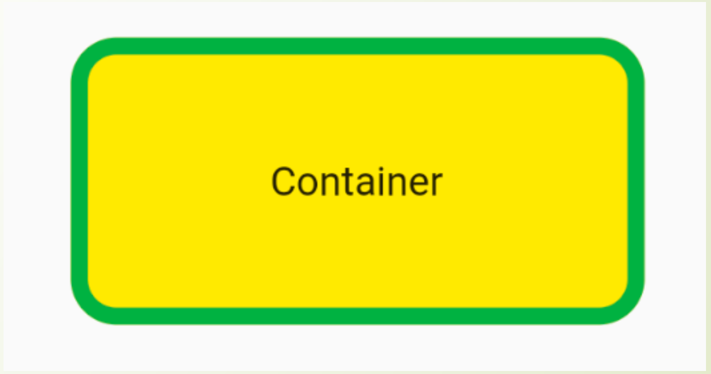
```
Center(
    child: Container(
        width: 200,
        height: 100,
        color: Colors.yellow,
        alignment: Alignment.center,
        child: const Text('Container’),
    ),
)
```


Container Widget

Container

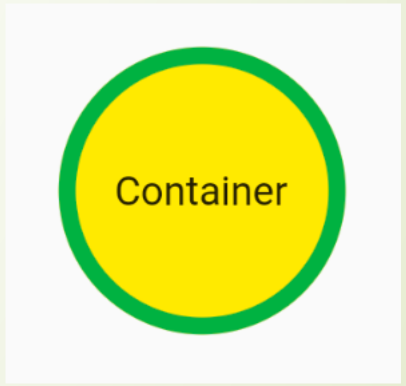
Container widget

```
Container(
    width: 200,
    height: 100,
    decoration: BoxDecoration(
        color: Colors.yellow,
        border: Border.all(width: 6, color: Colors.green),
        borderRadius: BorderRadius.circular(16)),
    alignment: Alignment.center,
    child: const Text('Container'),
)
```



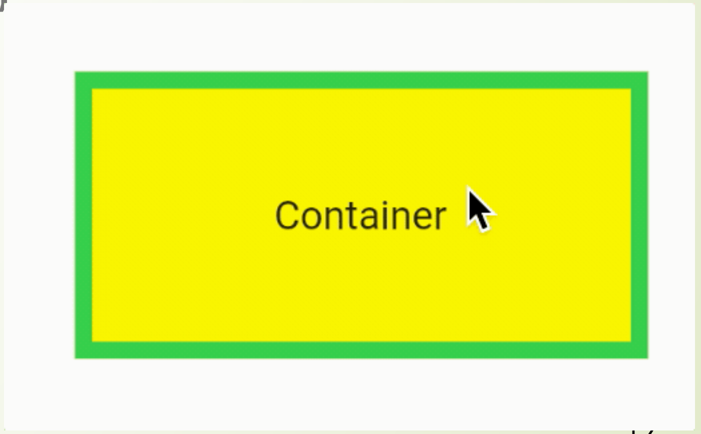
Container widget

```
Container(
     width: 200,
     height: 100,
     decoration: BoxDecoration(
          color: Colors.yellow,
          border: Border.all(width: 6, color: Colors.green),
          shape: BoxShape.circle,),
     alignment: Alignment.center,
     child: const Text('Container'),
)
```



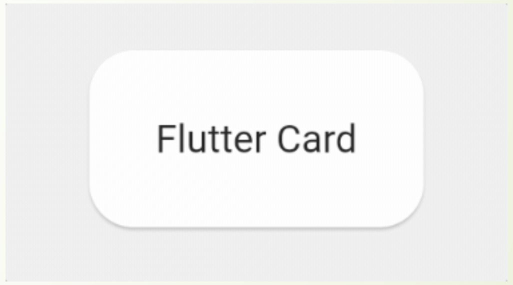
Container widget

```
Material(
    color: Colors.yellow,
    child: InkWell(
        onTap: () {},
        child: Container(
            width: 200,
            height: 100,
            decoration: BoxDecoration(
                border: Border.all(width: 6, color: Colors.green),
            ),
            alignment: Alignment.center,
            child: const Text('Container’),),
    ),
)
```



Container widget

```
Card(
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(16)),
    child: InkWell(
        onTap: () {},
        borderRadius: BorderRadius.circular(16),
        child: Container(
            padding: const EdgeInsets.all(24),
            child: Text('Flutter Card'),),
    ),
)
```



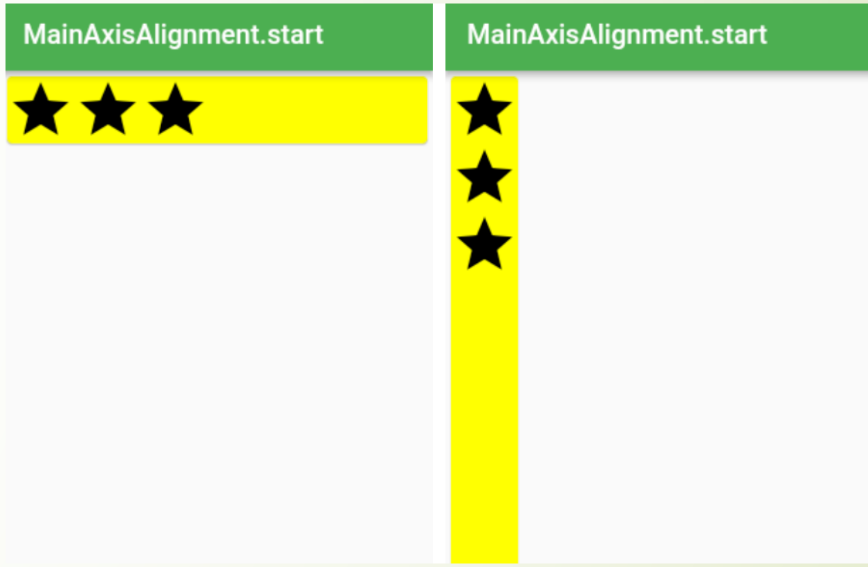
Container widget

- Some of the highlights of the Container widget are:
  - **Padding**: Container allows you to add padding to the widget, which creates space between the contents of the Container and its border.
  - **Decoration**: Container supports adding a decoration such as a background color, gradient, or image.
  - **Size**: Container supports setting the width, height, and constraints for the widget, allowing you to control the size and shape of the widget.
  - **Alignment**: Container allows you to align its child widgets using the alignment property.
  - **Margin**: Container also supports adding a margin, which creates space outside the Container's border.
  - **Flexibility**: Container can be combined with other layout widgets such as Row, Column, and Stack to create complex and flexible layouts.
  - **Reusability**: Container is a highly reusable widget, and it can be used multiple times in different parts of an app with different properties, making it an efficient widget to use in Flutter.

Row and Column

Row widget

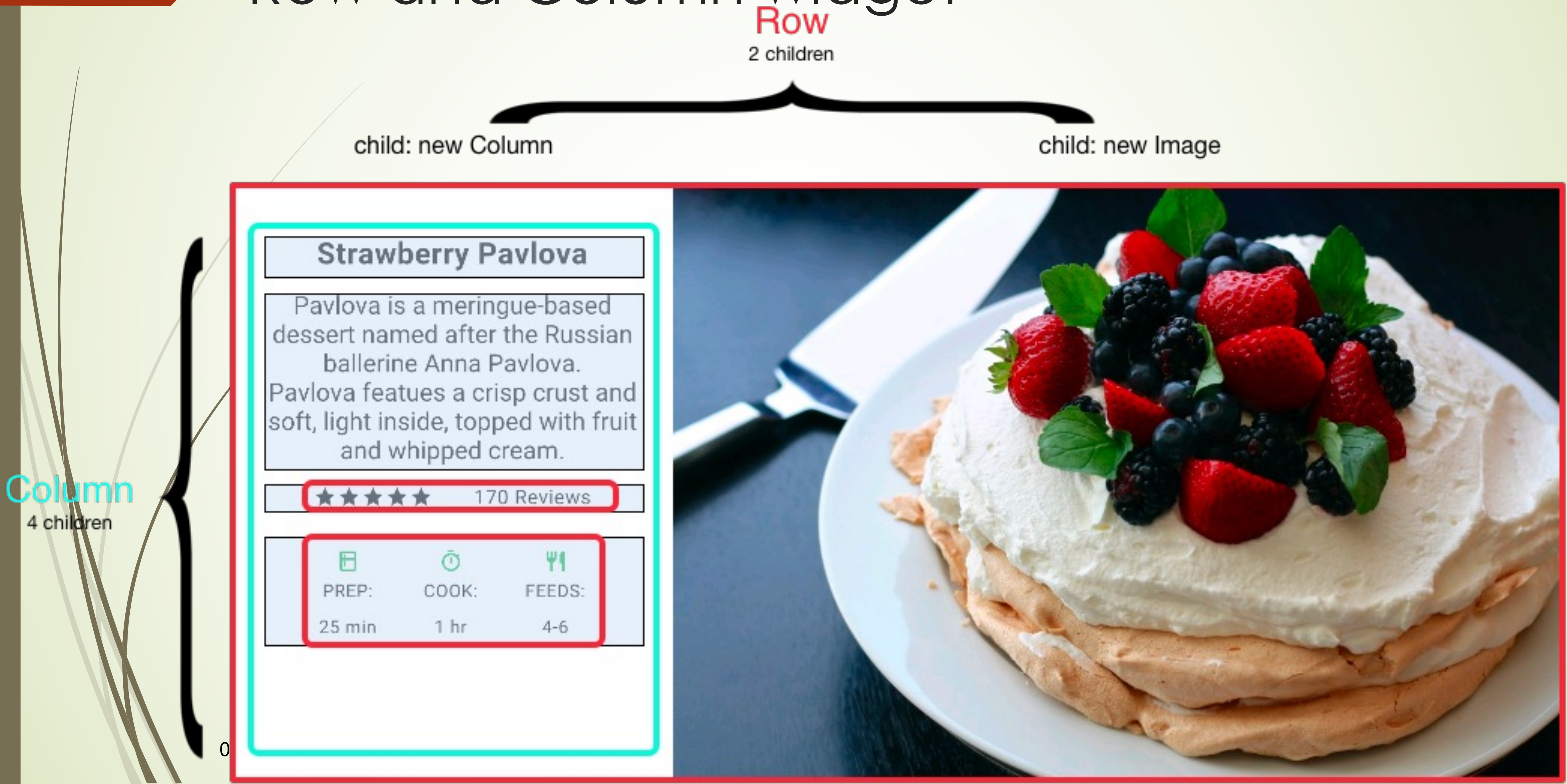
- The Flutter Row widget is a widget that displays its children in a horizontal line. It's often used to align widgets next to each other, such as text and icons, buttons, and images.
- One of the strengths of the Flutter Row widget is its ability to handle flexible and non-fixed-size widgets. For example, you can use the Expanded widget in combination with the Row widget to create a flexible layout that adjusts to different screen sizes.
- The Expanded widget allows you to specify how much space a widget should take up relative to the other widgets in the Row. This makes it possible to create responsive and adaptive layouts that work well on both large and small screens.



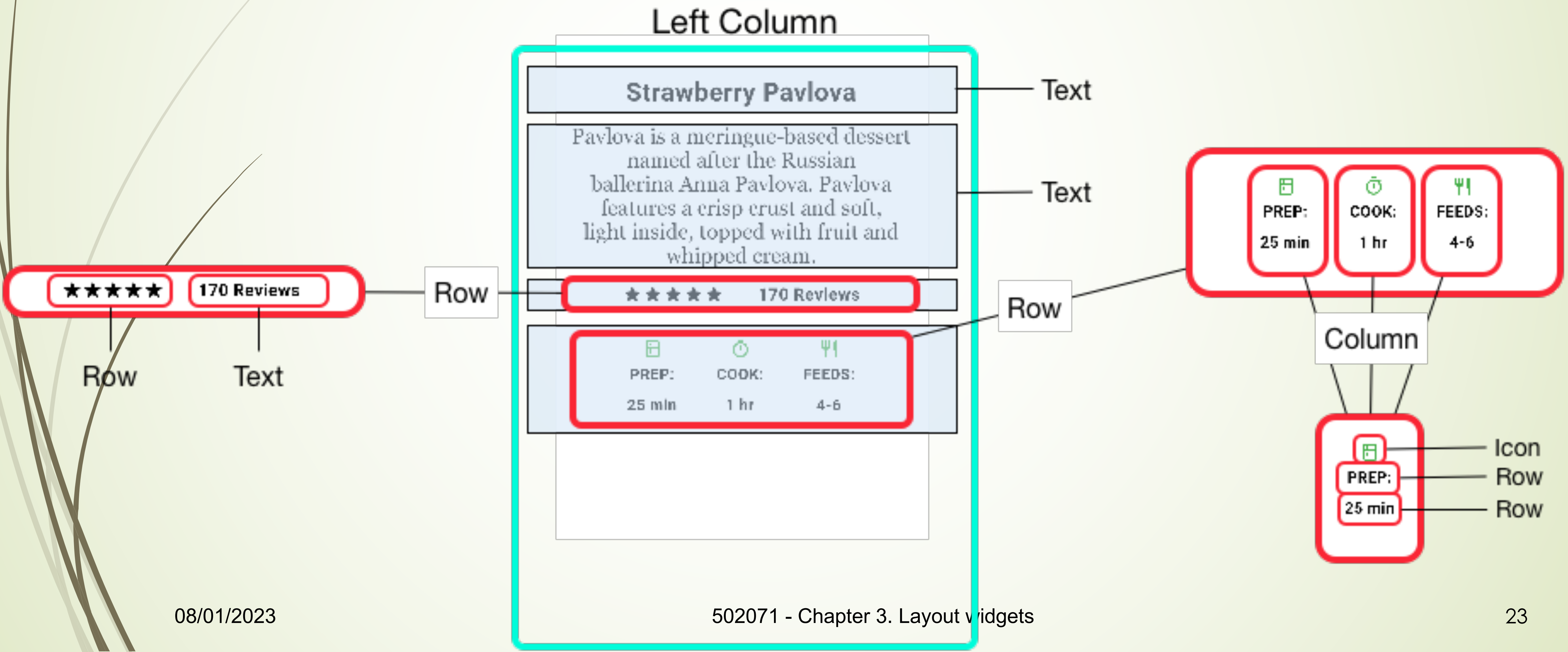
Row widget

- The Flutter Row widget allows developers to control the alignment, spacing, and size of its child widgets. For example, you can use the mainAxisAlignment and crossAxisAlignment properties to align the child widgets within the Row.
- The mainAxisAlignment property controls the alignment of the child widgets along the main axis, which is the horizontal axis, while the crossAxisAlignment property controls the alignment of the child widgets along the cross axis, which is the vertical axis.

Row and Column widget

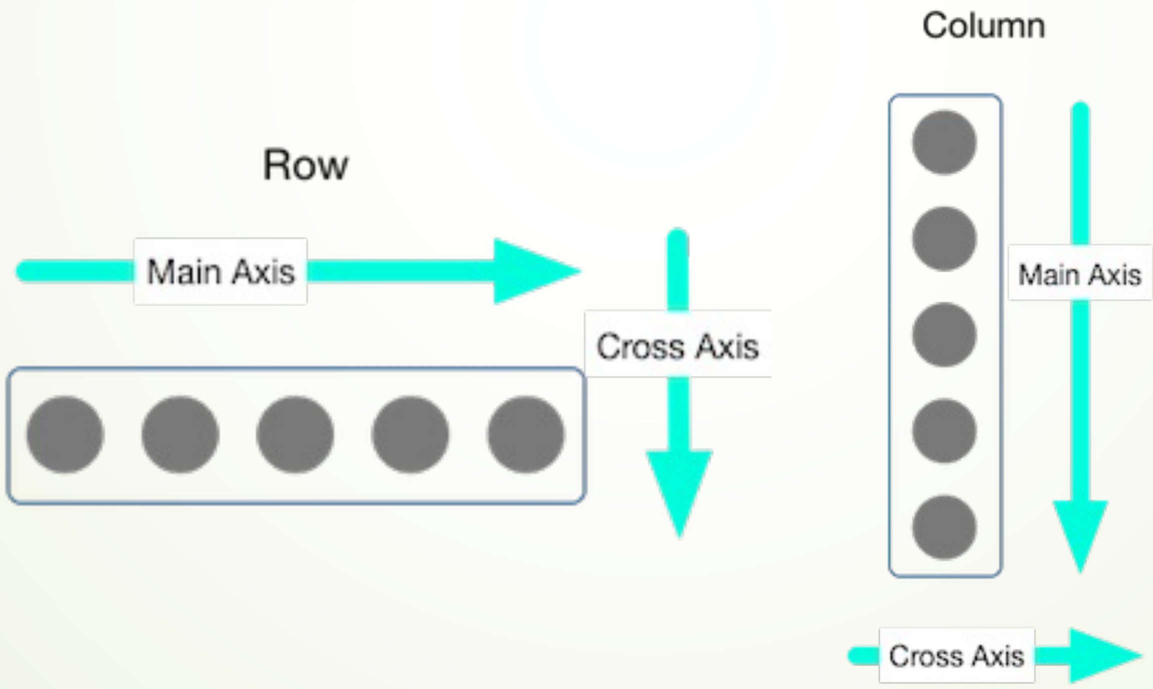

Row
2 children
child: new Column
child: new Image
Column
4 children
Strawberry Pavlova
Pavlova is a meringue-based dessert named after the Russian ballerine Anna Pavlova. Pavlova featues a crisp crust and soft, light inside, topped with fruit and whipped cream.
★★★★★ 170 Reviews
PREP: 25 min
COOK: 1 hr
FEEDS: 4-6
0

Row and Column widget


Left Column
Strawberry Pavlova
Pavlova is a meringue-based dessert named after the Russian ballerina Anna Pavlova. Pavlova features a crisp crust and soft, light inside, topped with fruit and whipped cream.
170 Reviews
PREP: 25 min
COOK: 1 hr
FEEDS: 4-6
Text
Text
Row
Row
Row
Text
Column
Icon
Row
Row

Aligning widgets

- You control how a row or column aligns its children using the mainAxisAlignment and crossAxisAlignment properties. For a row, the main axis runs horizontally and the cross axis runs vertically. For a column, the main axis runs vertically and the cross axis runs horizontally.

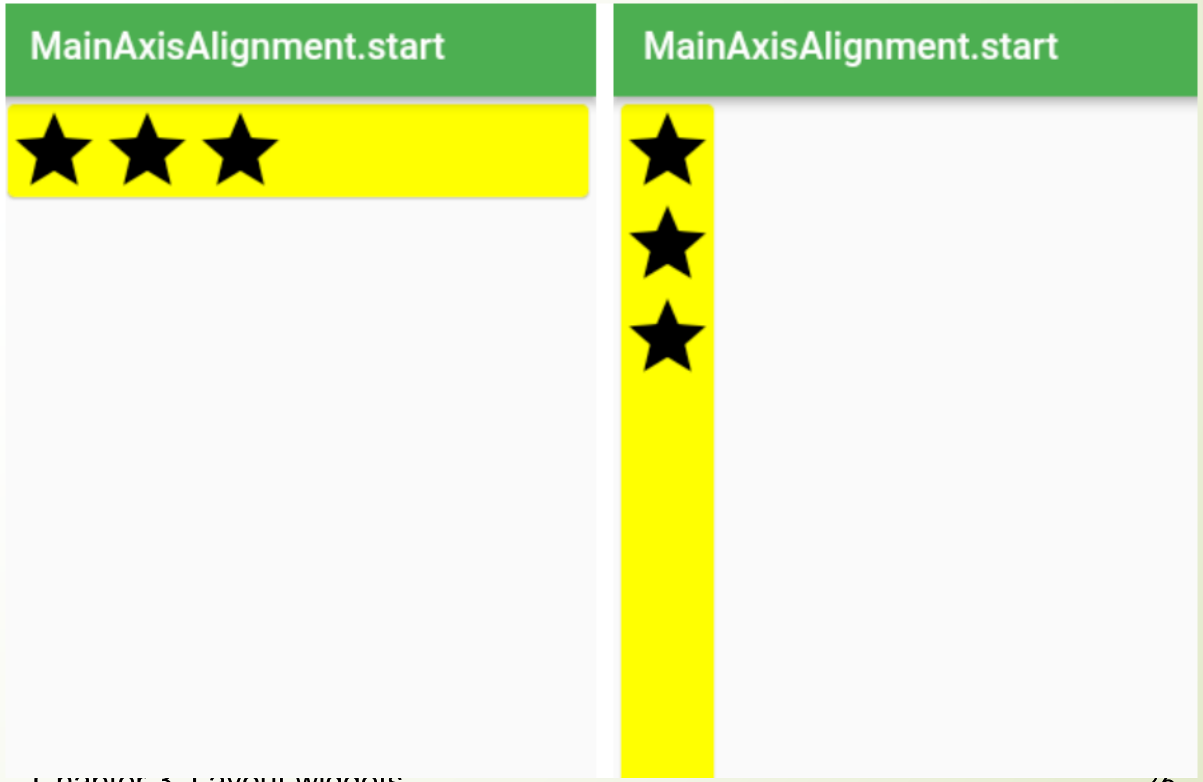


mainAxisAlignment

Row and Column widget

- **mainAxisAlignment** property

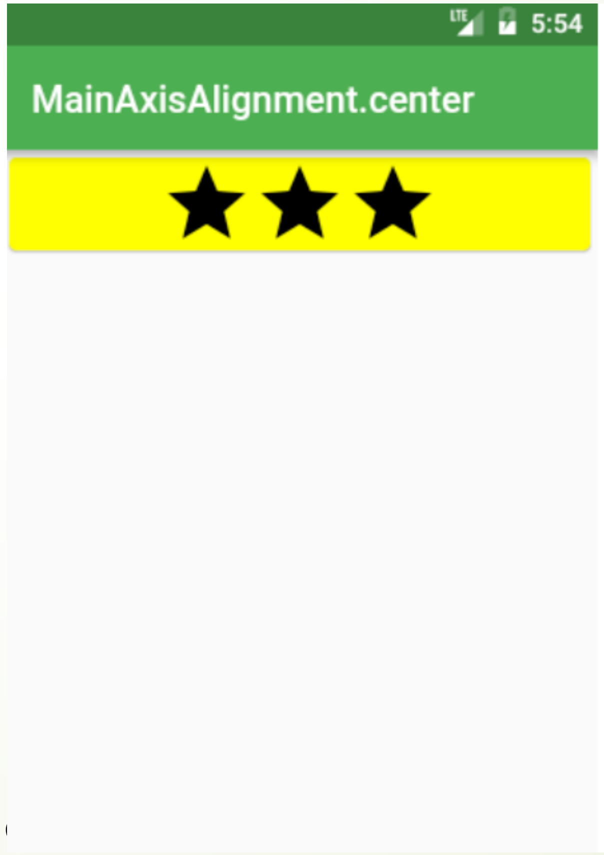
```
Row /*or Column*/(
    mainAxisAlignment: MainAxisAlignment.start,
    children: <Widget>[
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),],
),
```

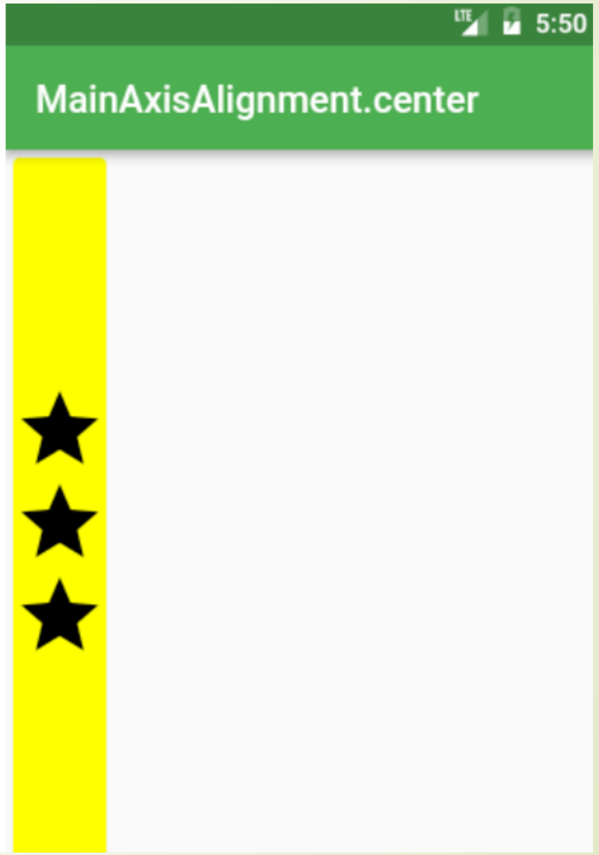


Row and Column widget

- **mainAxisAlignment** property

```
Row /*or Column*/(
      mainAxisAlignment: MainAxisAlignment.center,
      children: <Widget>[
      Icon(Icons.star, size: 50),
      Icon(Icons.star, size: 50),
      Icon(Icons.star, size: 50),],
),
```

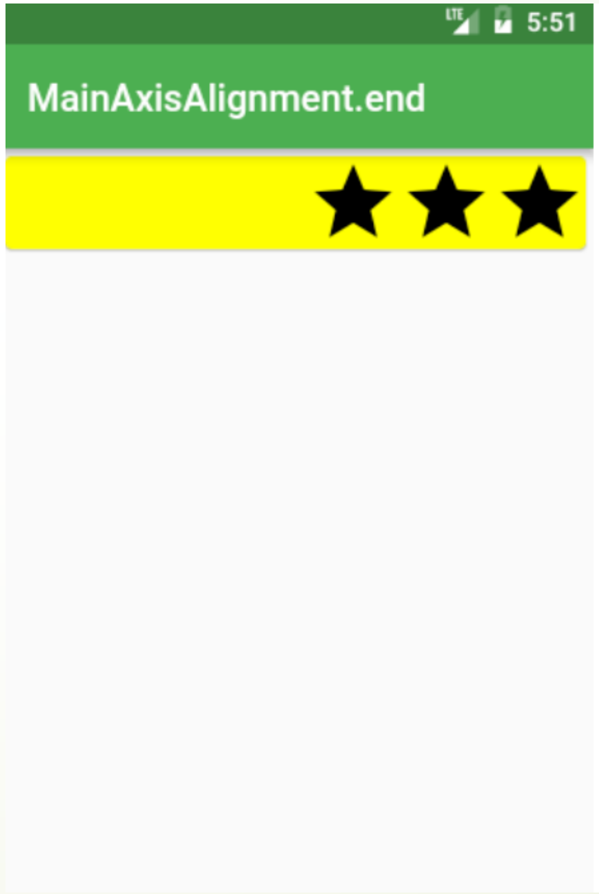


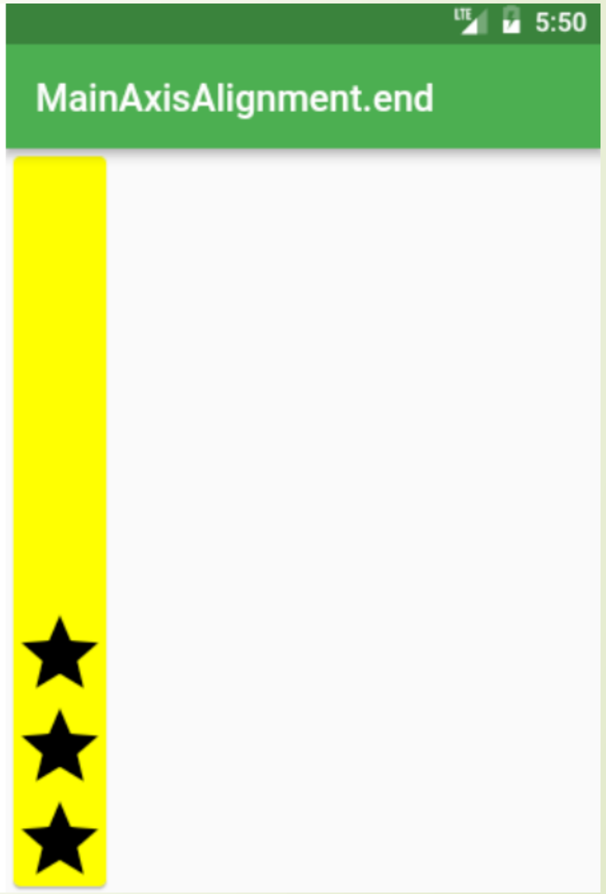


Row and Column widget

- **mainAxisAlignment** property

```
Row /*or Column*/(
    mainAxisAlignment: MainAxisAlignment.end,
    children: <Widget>[
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),],
),
```

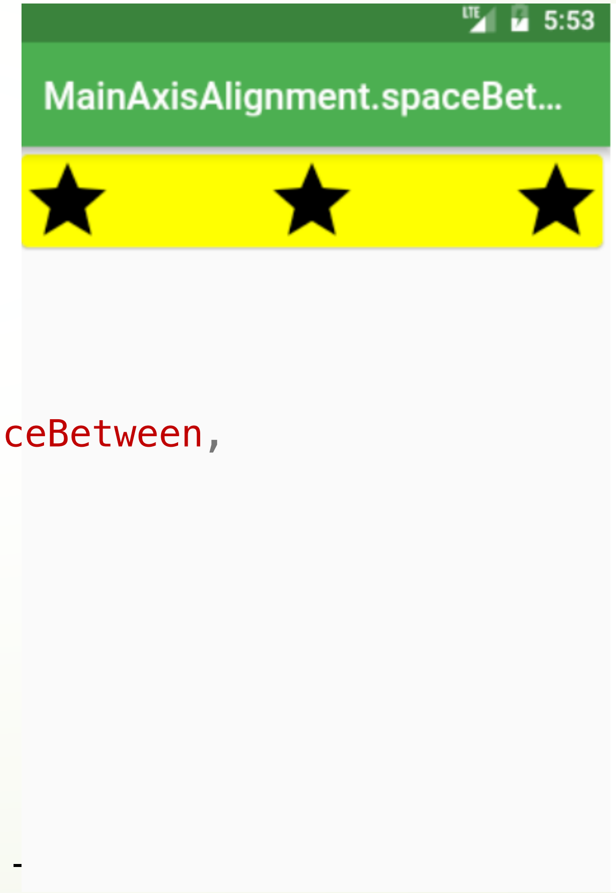


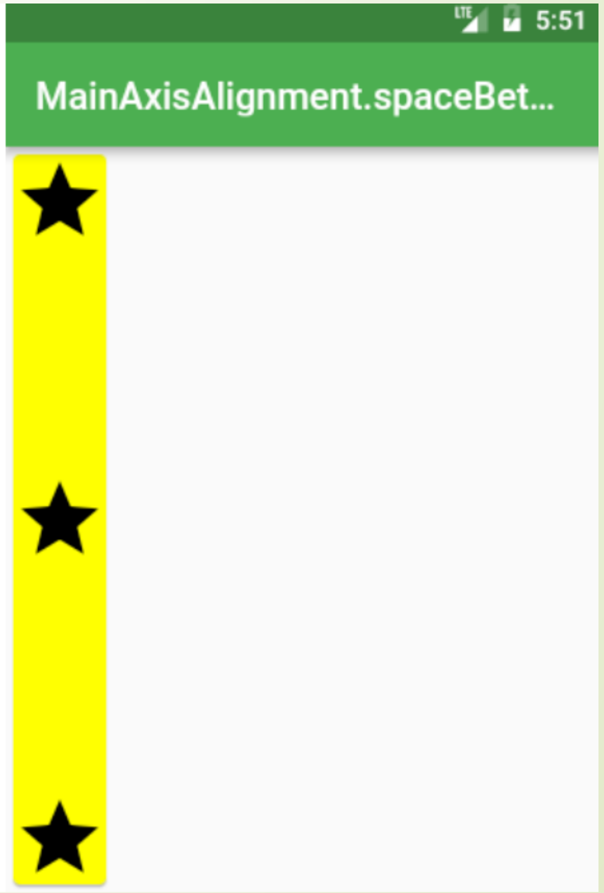


Row and Column widget

- **mainAxisAlignment** property

```
Row /*or Column*/(
    mainAxisAlignment: MainAxisAlignment. spaceBetween,
    children: <Widget>[
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),],
),
```

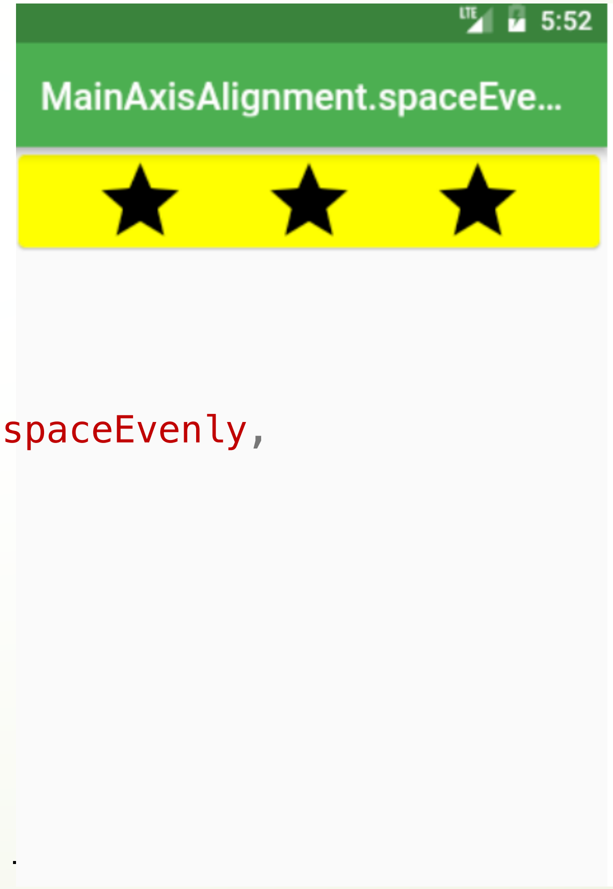


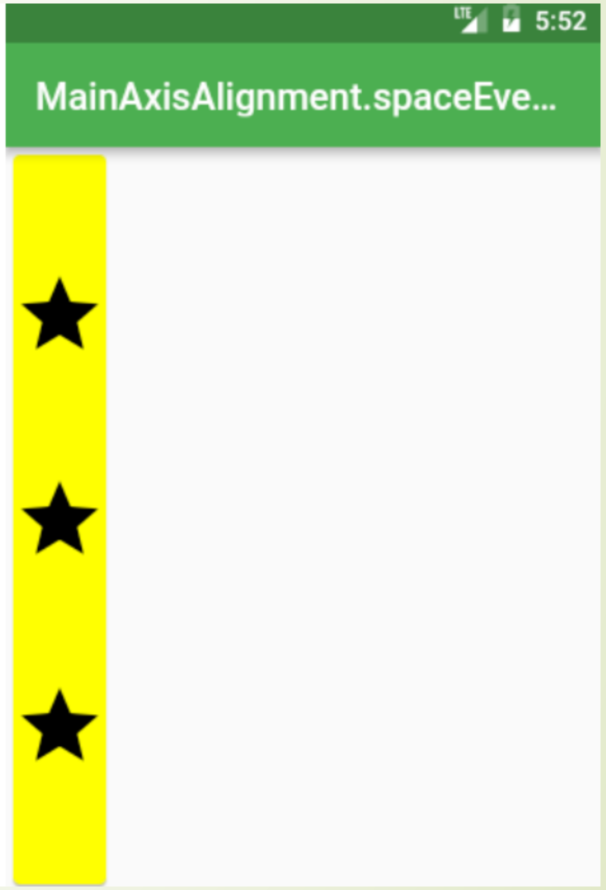


Row and Column widget

- **mainAxisAlignment** property

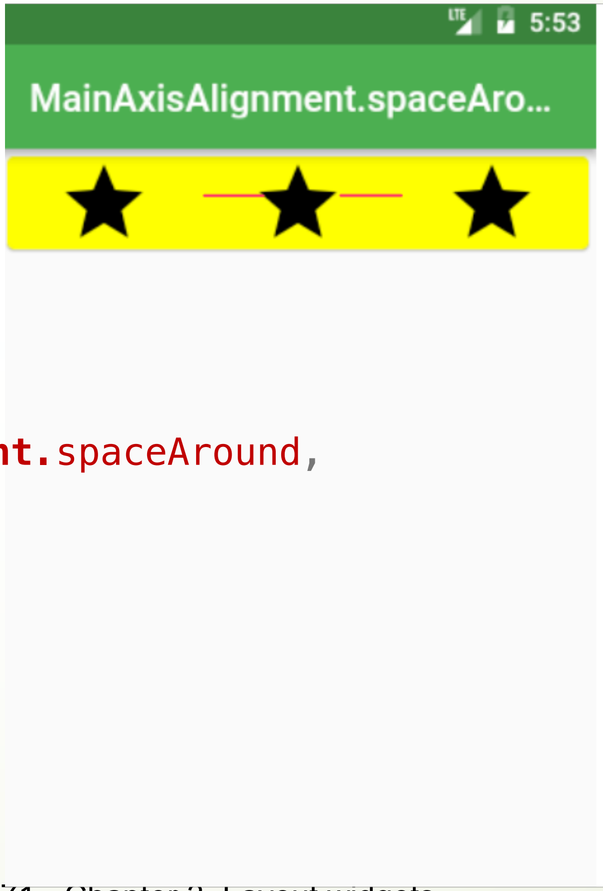
```
Row /*or Column*/(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: <Widget>[
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),],
),
```

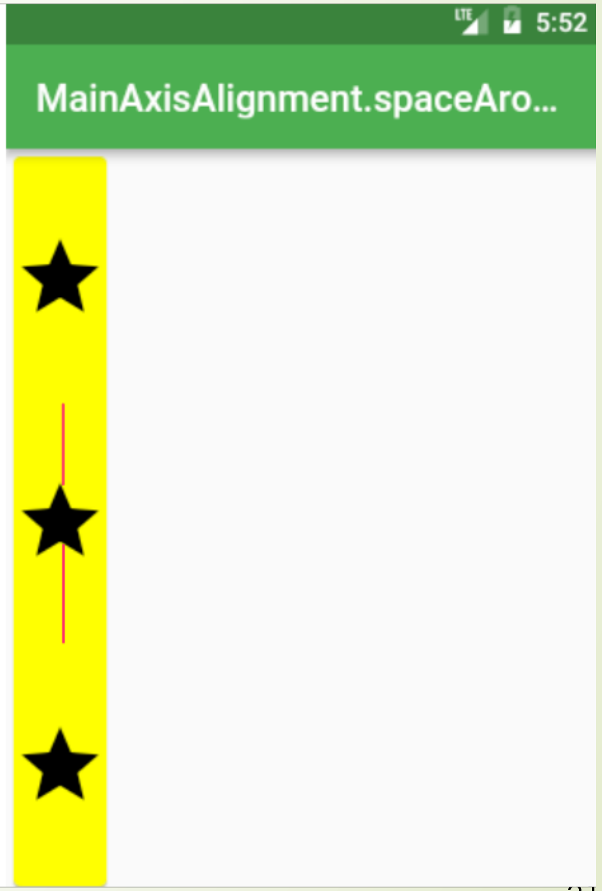




Row and Column widget

- **mainAxisAlignment** property



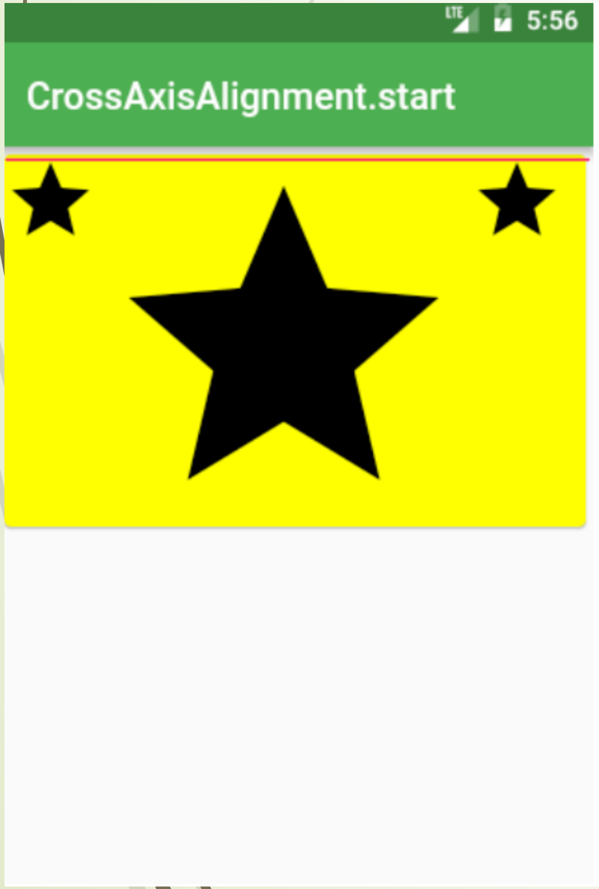


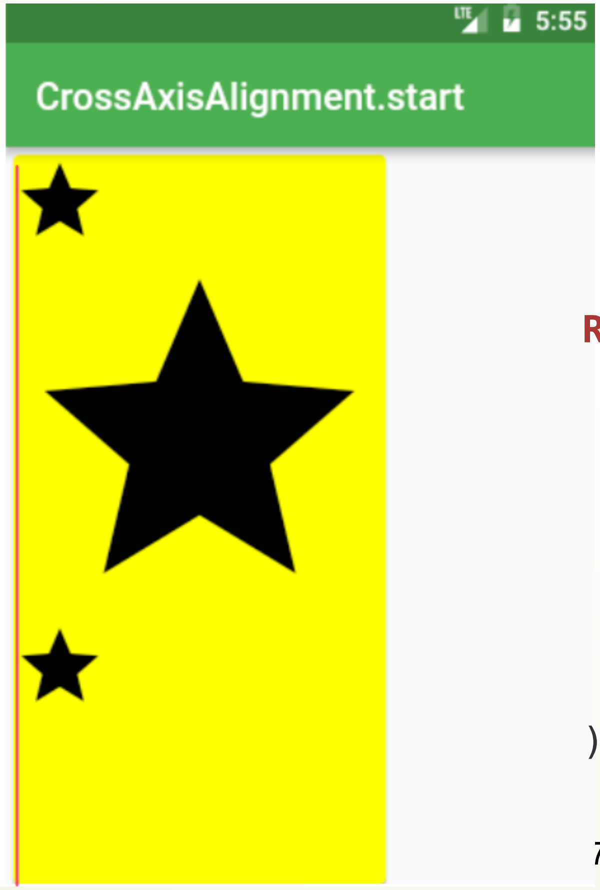
```
Row /*or Column*/(
    mainAxisAlignment: MainAxisAlignment.spaceAround,
    children: <Widget>[
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 50),],
),
```

crossAxisAlignment

Row and Column widget

- **crossAxisAlignment** property

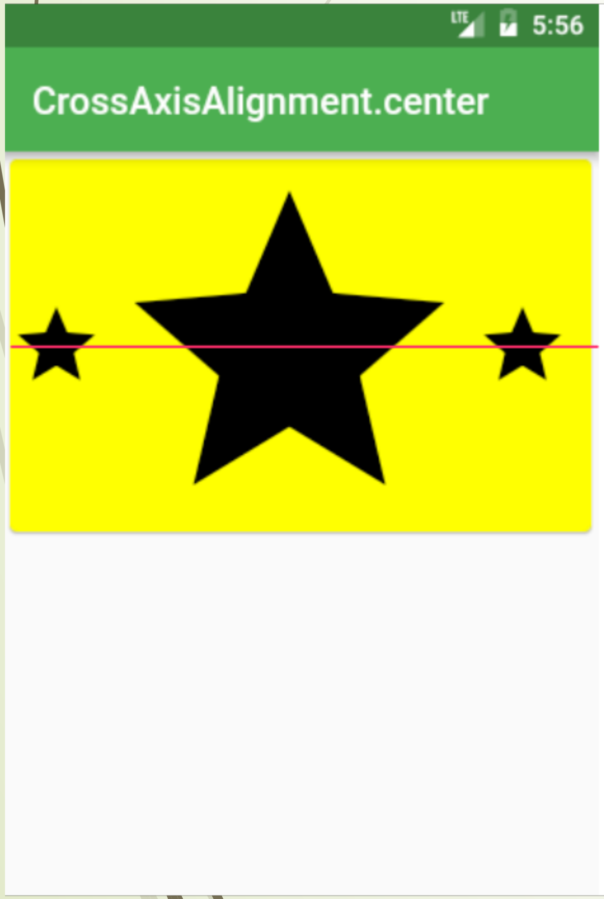


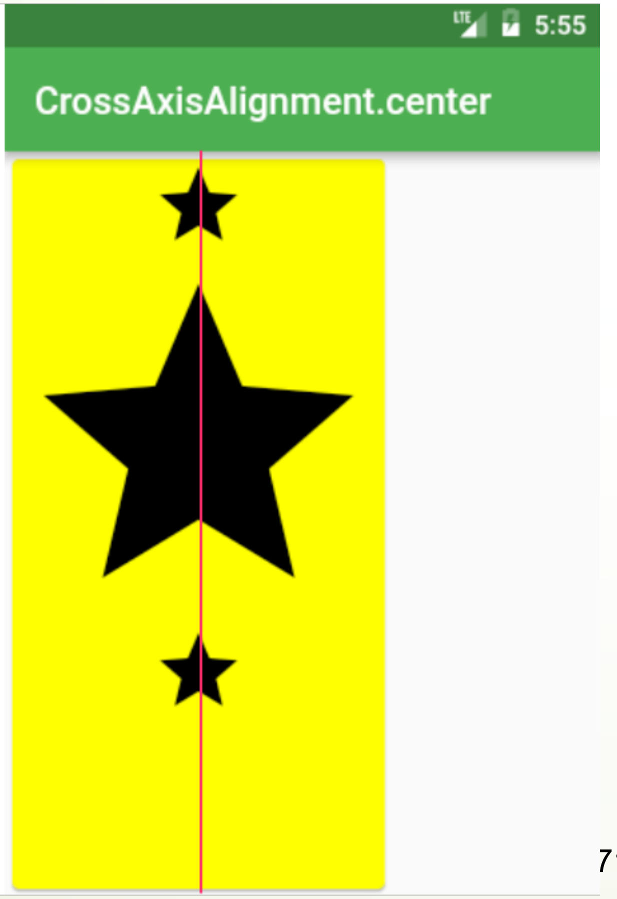


```
Row /*or Column*/(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: <Widget>[
    Icon(Icons.star, size: 50),
    Icon(Icons.star, size: 200),
    Icon(Icons.star, size: 50),],
),
```

Row and Column widget

- **crossAxisAlignment** property

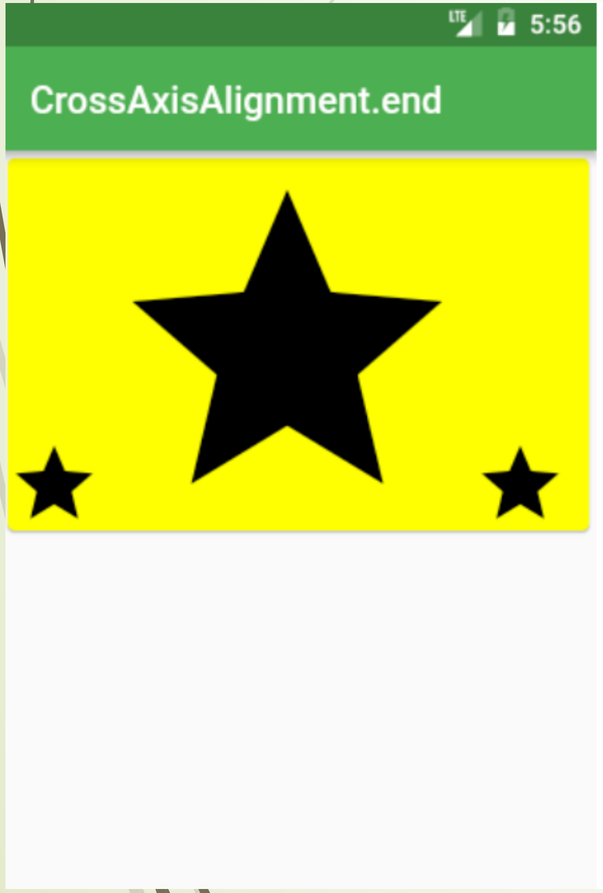


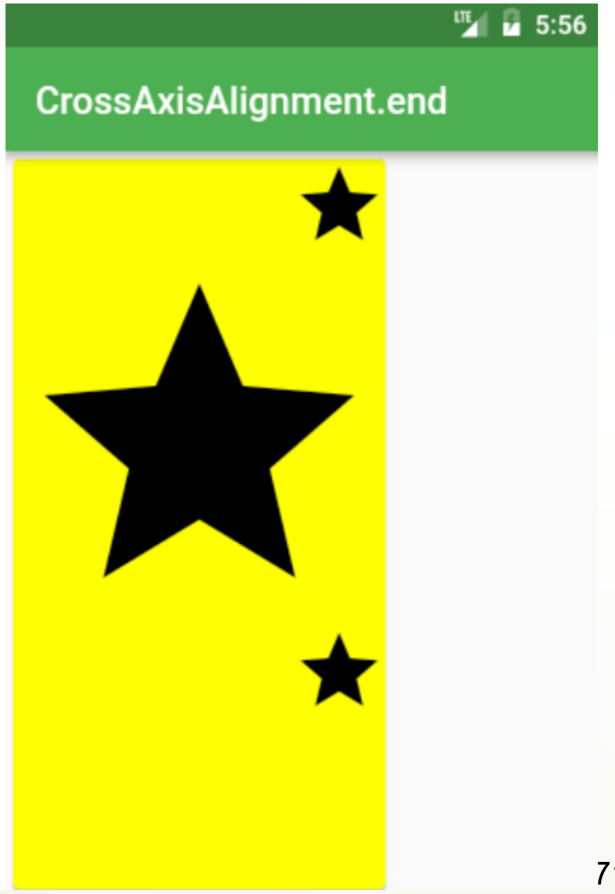


```
Row /*or Column*/(
      crossAxisAlignment: CrossAxisAlignment.center,
      children: <Widget>[
      Icon(Icons.star, size: 50),
      Icon(Icons.star, size: 200),
      Icon(Icons.star, size: 50),],
),
```

Row and Column widget

- **crossAxisAlignment** property



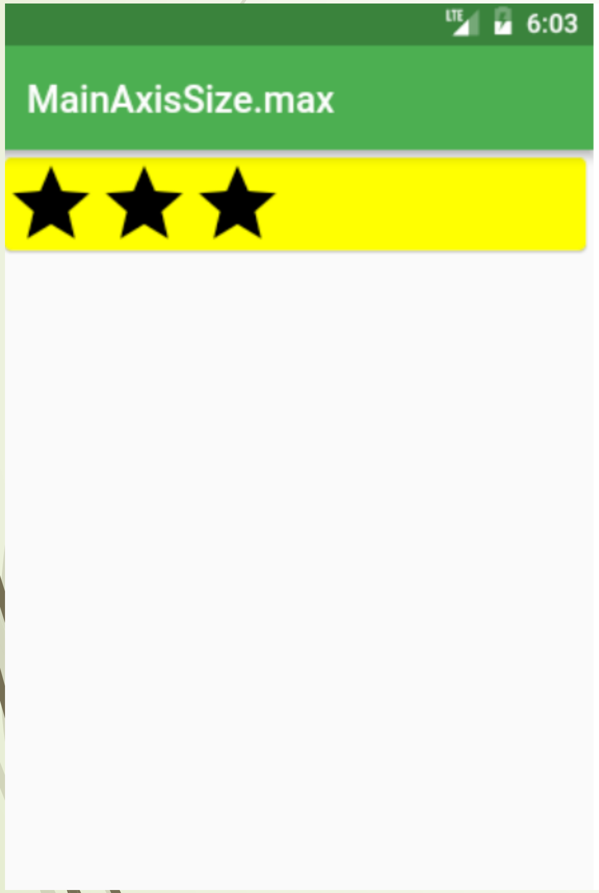


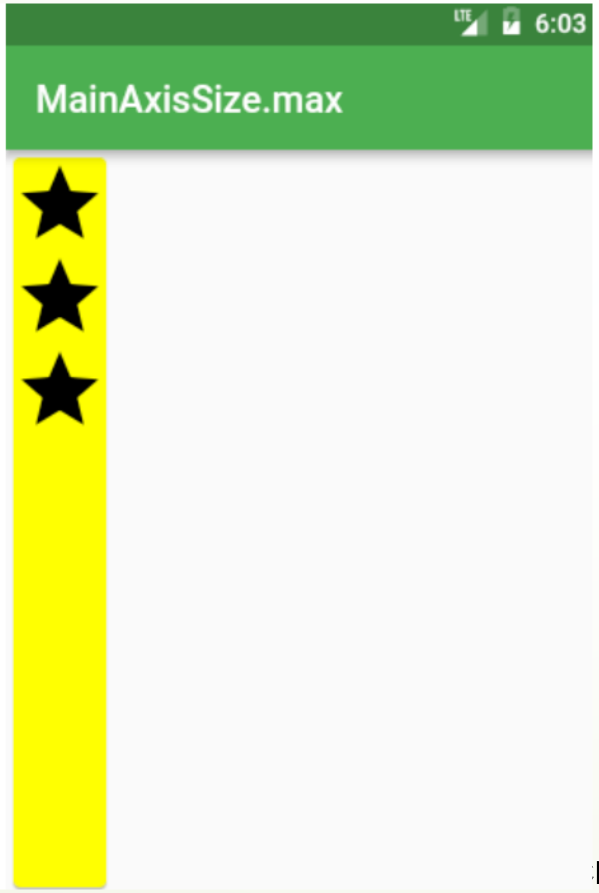
```
Row /*or Column*/(
     crossAxisAlignment: CrossAxisAlignment.end,
     children: <Widget>[
     Icon(Icons.star, size: 50),
     Icon(Icons.star, size: 200),
     Icon(Icons.star, size: 50),],
),
```

mainAxisSize

Row and Column widget

- **mainAxisSize** property

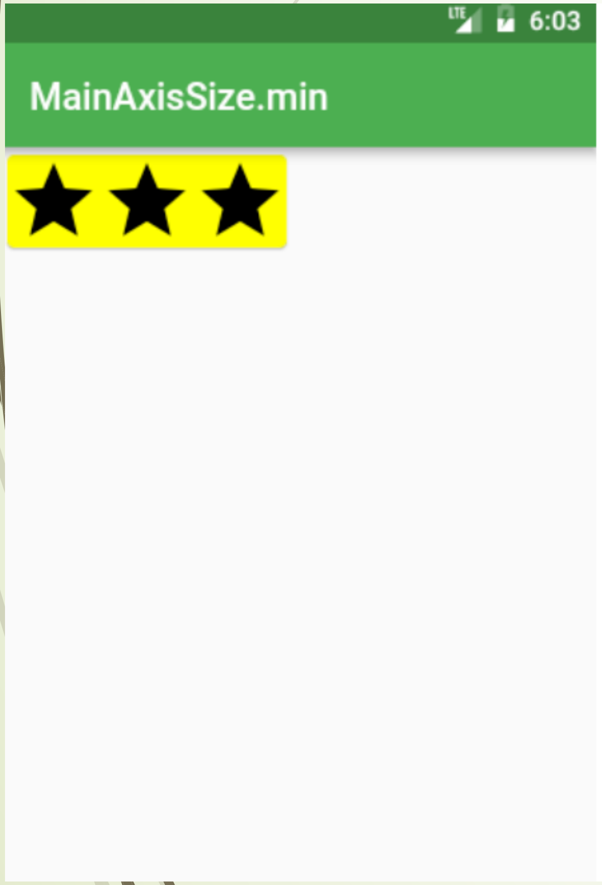


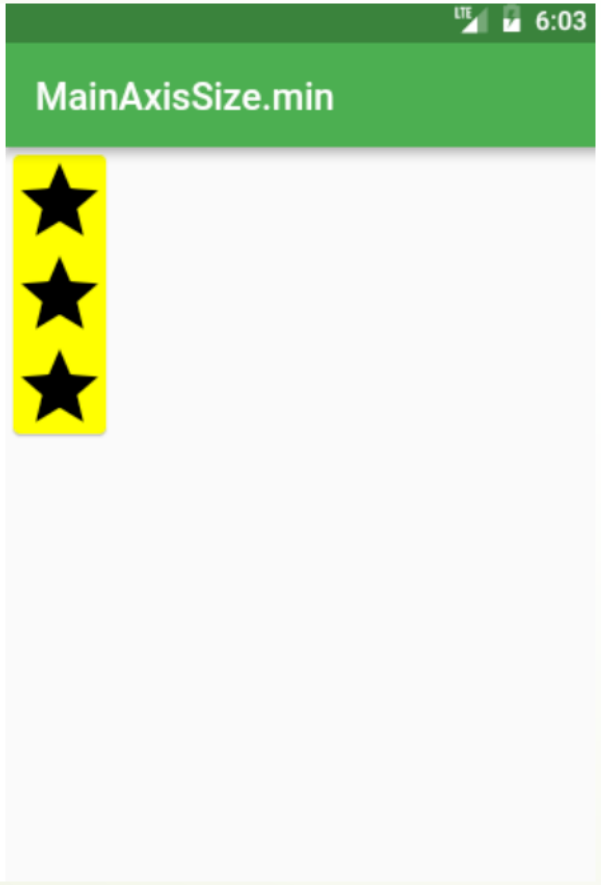


```
Row /*or Column*/(
    mainAxisSize: MainAxisSize.max,
    children: <Widget>[
        Icon(Icons.star, size: 50),
        Icon(Icons.star, size: 50),
        Icon(Icons.star, size: 50)]),
```

Row and Column widget

- **mainAxisSize** property

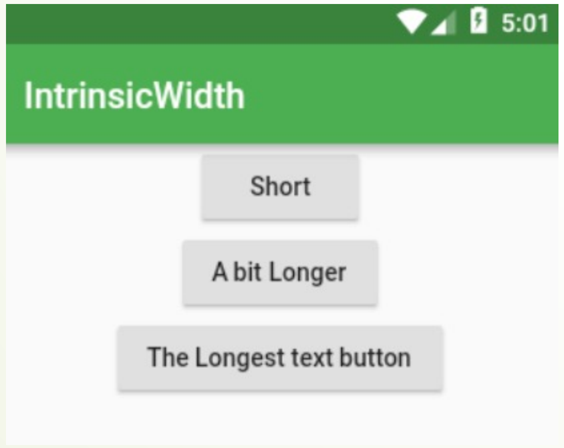


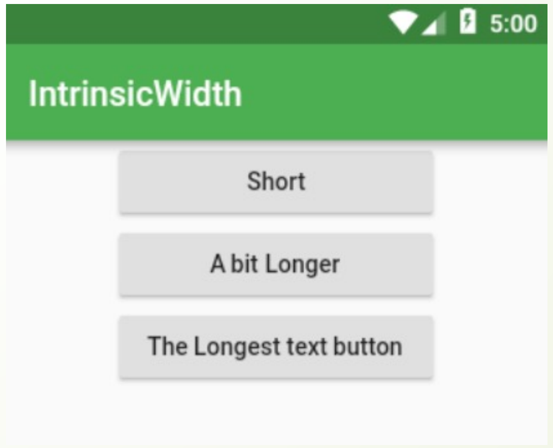


```
Row /*or Column*/(
    mainAxisSize: MainAxisSize.min,
    children: <Widget>[
        Icon(Icons.star, size: 50),
        Icon(Icons.star, size: 50),
        Icon(Icons.star, size: 50)]),
```

IntrinsicWidth and IntrinsicHeight

- IntrinsicWidth sets the width of its child to the child's maximum intrinsic width. This means that the width of the child will be based on its own internal constraints, such as the width of its text or images.
- IntrinsicHeight works similarly, but sets the height of its child to the child's maximum intrinsic height.





IntrinsicWidth and IntrinsicHeight

```
IntrinsicWidth(
    child: Column(
    crossAxisAlignment:
    CrossAxisAlignment.stretch,
    children: <Widget>[
    ElevatedButton(
        onPressed: () {},
        child: Text('Short')),
    ElevatedButton(
        onPressed: () {},
        child: Text('A bit Longer’)),
    ElevatedButton(
        onPressed: () {},
        child: Text('The Longest text button’))]
    ),
)
```

Stack widget

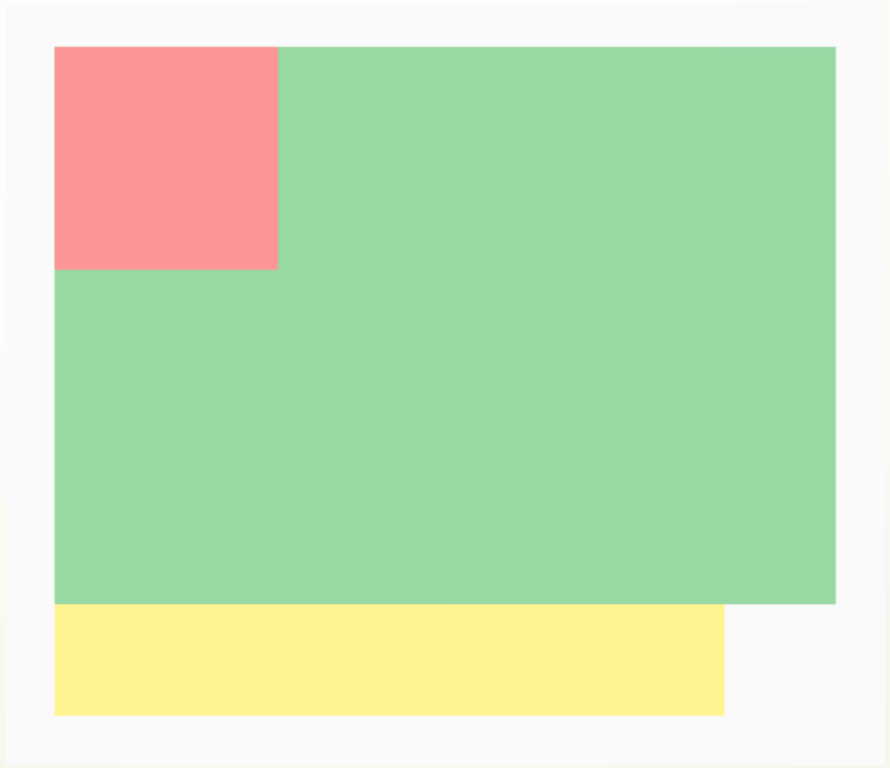
- The Flutter Stack widget is a layout widget in the Flutter framework that allows developers to overlap widgets on top of each other. It's one of the most commonly used layout widgets in Flutter and is used to create a stack of widgets, where each widget is positioned on top of the other in a z-order fashion.
- The Flutter Stack widget is a multi-child layout widget, meaning that it can have multiple child widgets. These child widgets are stacked on top of each other, with the first child widget being the bottommost widget in the stack and the last child widget being the topmost widget.

Stack widget

- The Flutter Stack widget provides several properties that allow developers to control the alignment, spacing, and size of its child widgets. For example, you can use the alignment property to specify the alignment of the child widgets within the Stack, and the fit property to control how the Stack should size its child widgets.
- One of the strengths of the Flutter Stack widget is its ability to handle transparent widgets. You can use the Stack widget to overlap widgets with transparent backgrounds, creating unique and beautiful user interfaces. This makes it possible to create layered and overlapping designs that stand out from the rest.

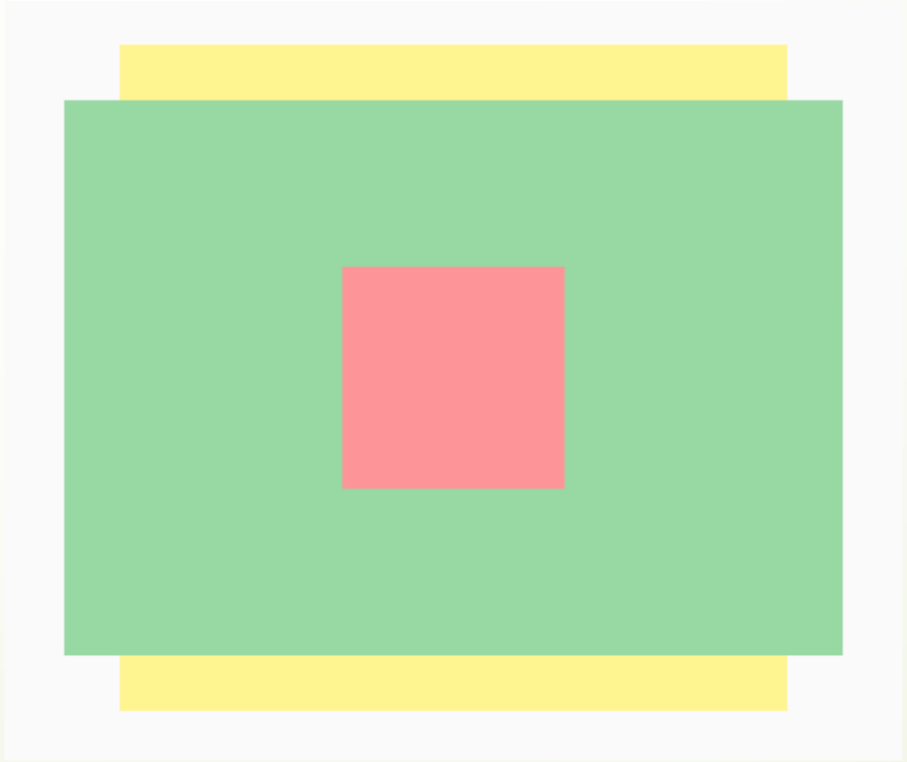
Stack widget

```
Stack(children: [
    Container(width: 300, height: 300,color: Colors.yellow[200]),
    Container(width: 350, height: 250, color: Colors.green[200]),
    Container(width: 100, height: 100, color: Colors.red[200],)
])
```



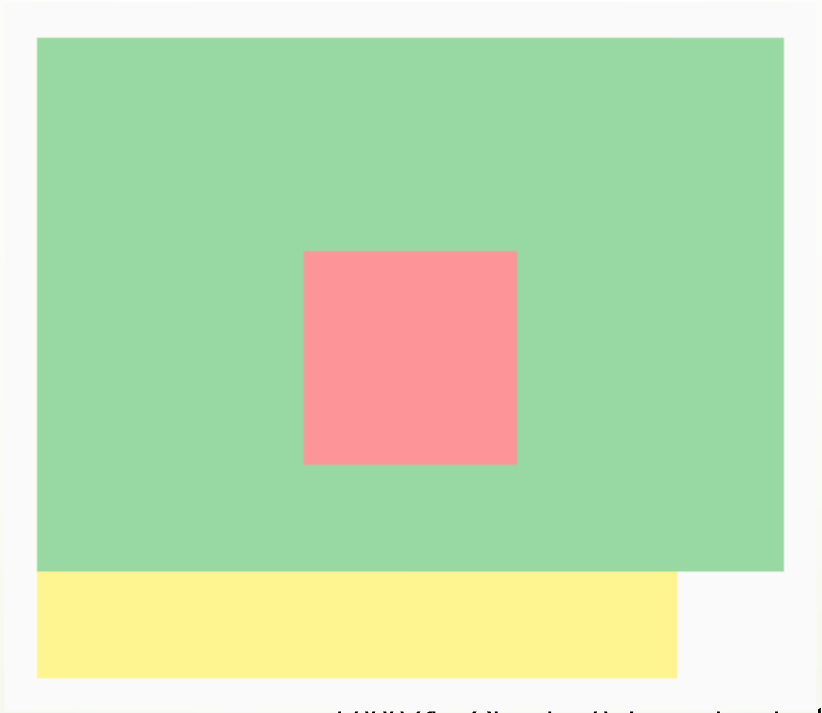
Stack widget

```
Stack(
    alignment: AlignmentDirectional.center,
    children: [
        Container(width: 300, height: 300,color: Colors.yellow[200]),
        Container(width: 350, height: 250, color: Colors.green[200]),
        Container(width: 100, height: 100, color: Colors.red[200],)]
)
```



Stack widget

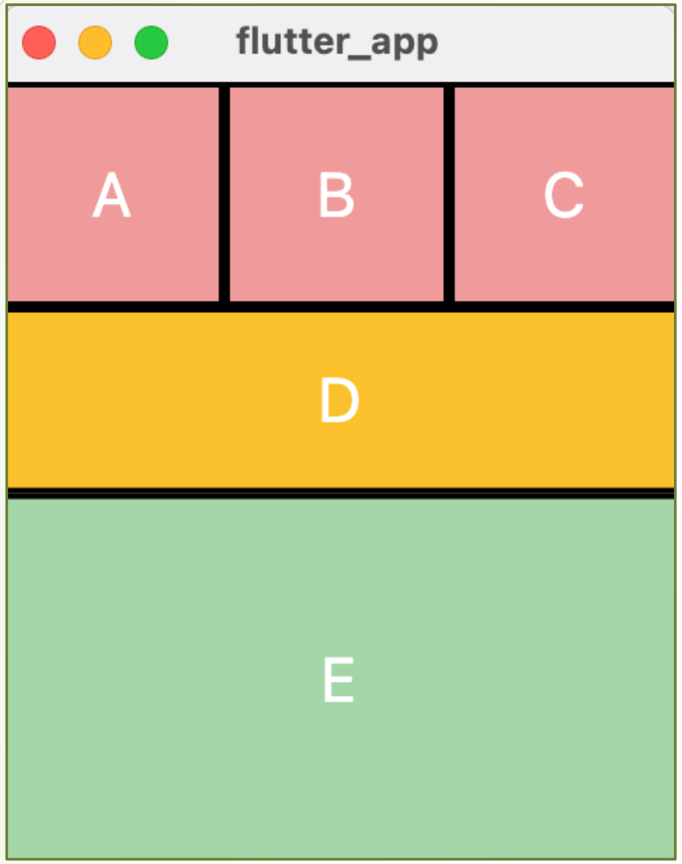
```
Stack(children: [
    Container(width: 300, height: 300, color: Colors.yellow[200],),
    Container(width: 350, height: 250, color: Colors.green[200],),
    Positioned(top: 100, left: 125,
        child: Container(
            width: 100, height: 100, color: Colors.red[200],),
    )
])
```

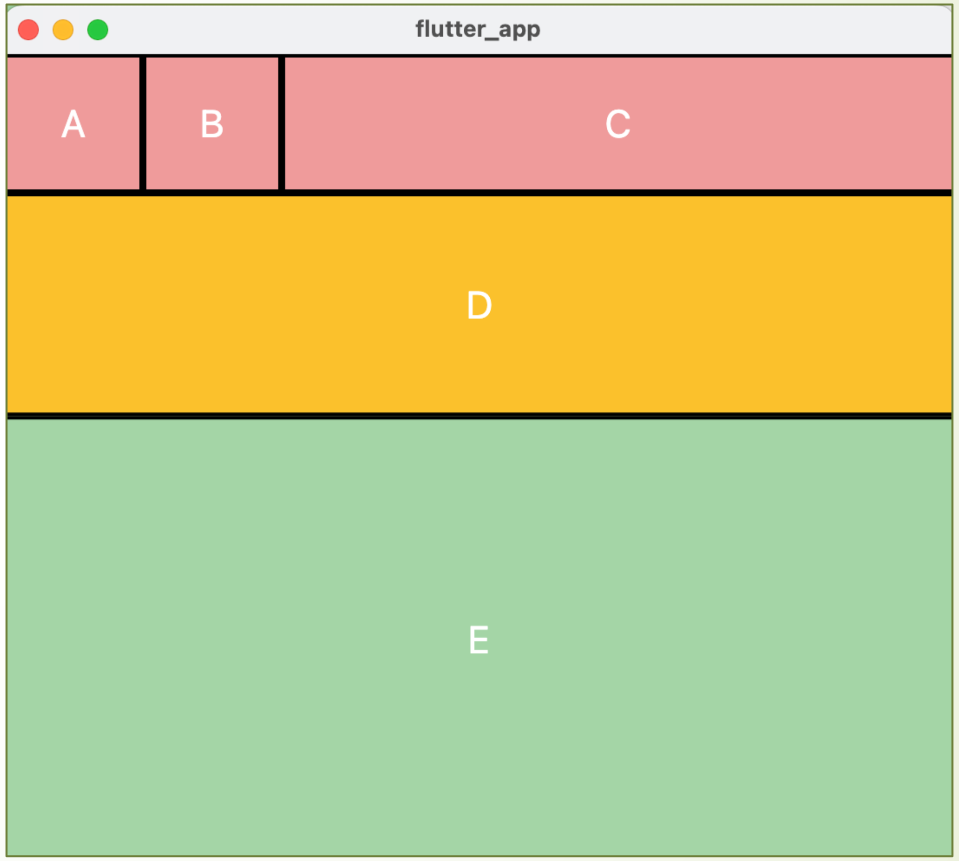


Expaned widget

- The Flutter Expanded widget is a layout widget in the Flutter framework that allows developers to create flexible and non-fixed-size widgets. It's used in combination with other layout widgets, such as the Row and Column widgets, to create a flexible layout that adjusts to different screen sizes.
- The Expanded widget is used to specify how much space a widget should take up relative to the other widgets in the same row or column. For example, you can use the Expanded widget to make a widget take up more space than other widgets in the same row or column, or you can use it to make a widget take up less space.

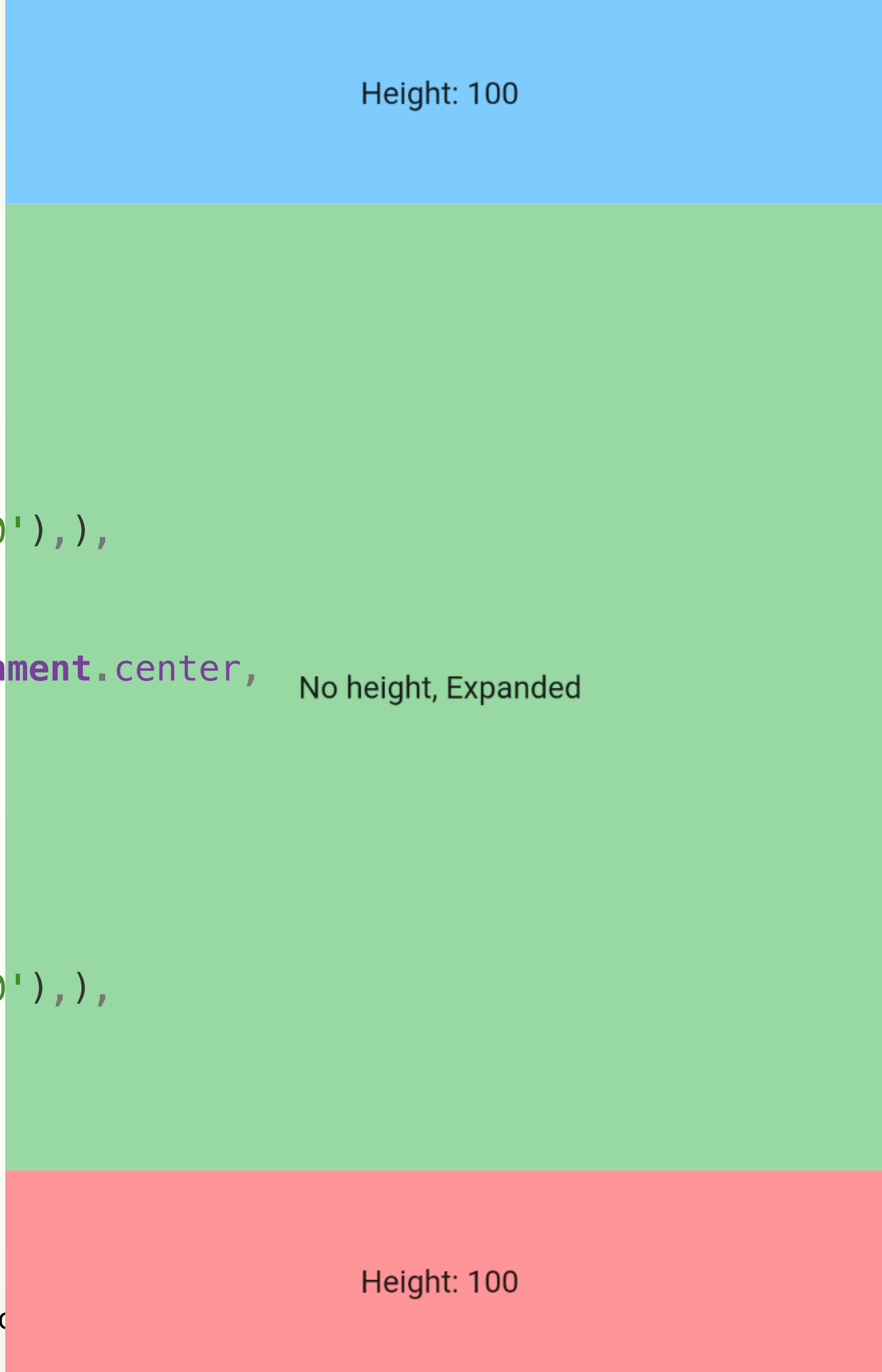
Expaned widget





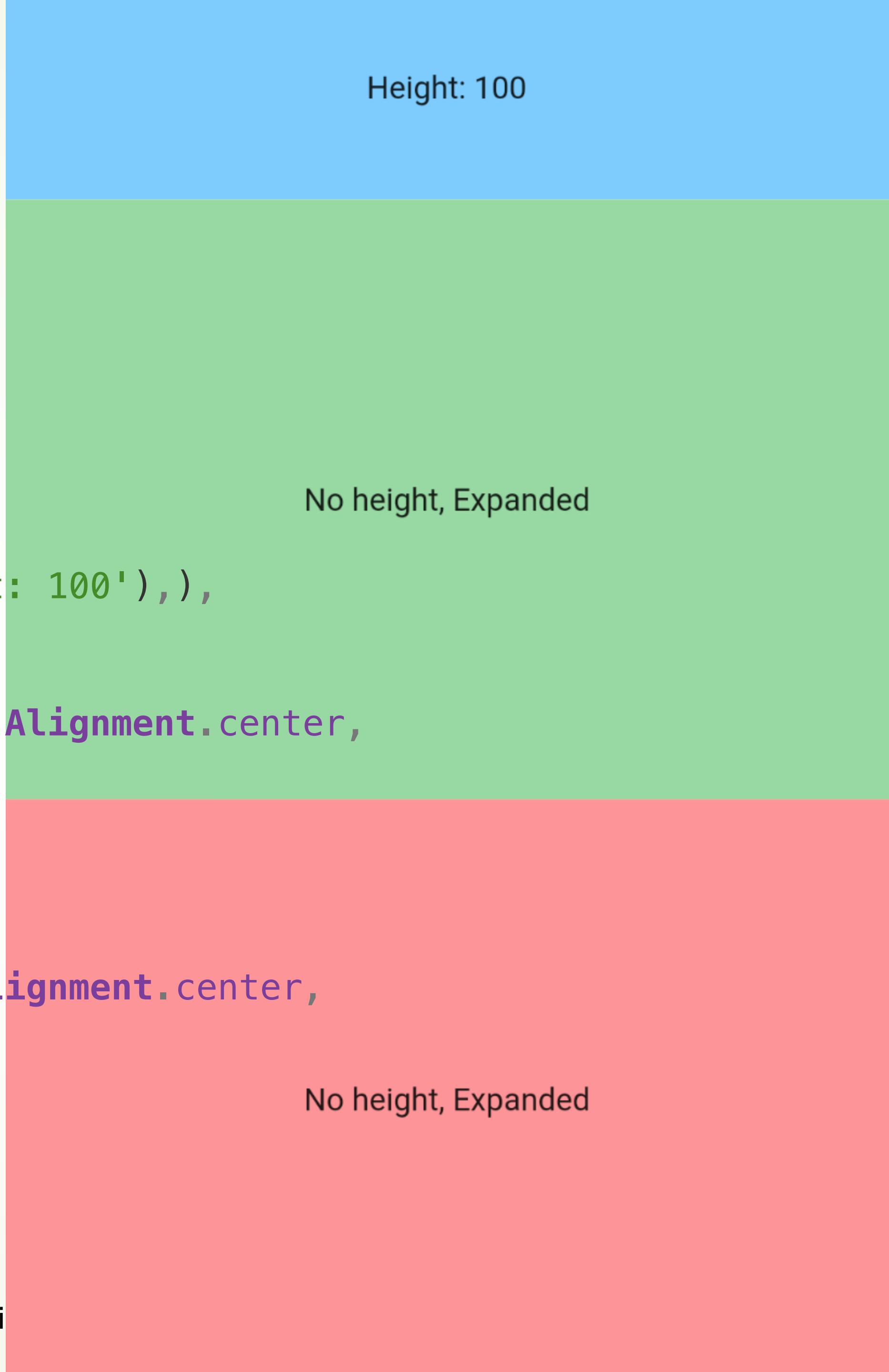
Expaned widget

```
Column(children: [
    Container(
        color: Colors.blue[200], height: 100,
        alignment: Alignment.center, child: const Text('Height: 100'),),
    Expanded(
        child: Container(color: Colors.green[200], alignment: Alignment.center,
        child: const Text('No height, Expanded'),),
    ),
    Container(
        color: Colors.red[200], height: 100,
        alignment: Alignment.center, child: const Text('Height: 100'),),
])
```



Expaned widget

```
Column(children: [
    Container(
        color: Colors.blue[200], height: 100,
        alignment: Alignment.center, child: const Text('Height: 100'),),
    Expanded(
        child: Container(color: Colors.green[200], alignment: Alignment.center,
        child: const Text('No height, Expanded'),),
    ),
    Expanded(
        child: Container(color: Colors.red[200], alignment: Alignment.center,
        child: const Text('No height, Expanded'),),
    ),
])
```

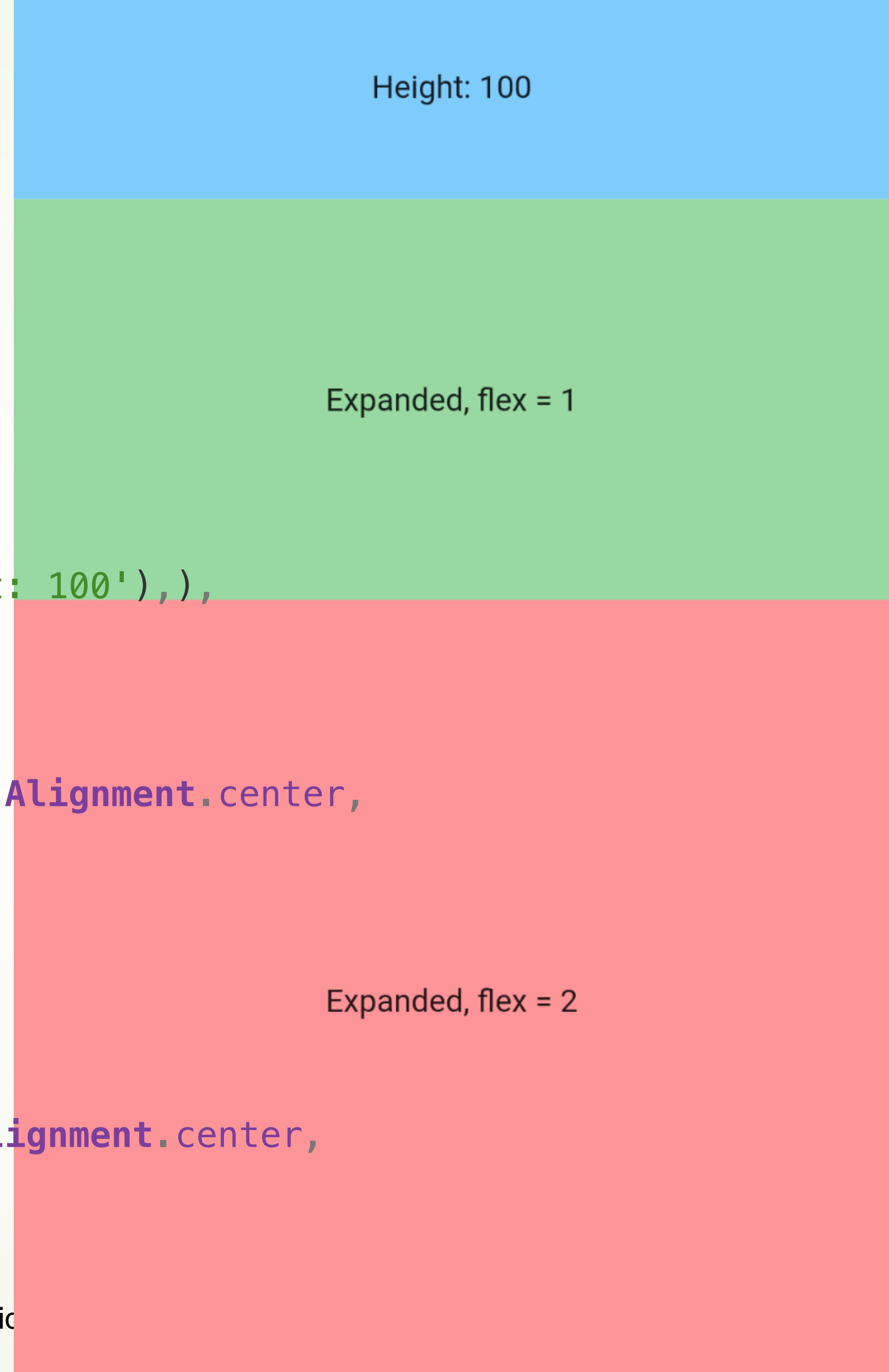


Expaned widget

- The Flutter Expanded widget provides the flex property, which allows developers to specify the proportion of space that the widget should take up relative to the other widgets in the same row or column.
- For example, you can set the flex property to 2 to make a widget take up twice as much space as other widgets in the same row or column.

Expaned widget

```
Column(children: [
    Container(
        color: Colors.blue[200], height: 100,
        alignment: Alignment.center, child: const Text('Height: 100'),),
    Expanded(
        flex: 1,
        child: Container(color: Colors.green[200], alignment: Alignment.center,
        child: const Text('Expanded, flex = 1'),),
    ),
    Expanded(
        flex: 2,
        child: Container(color: Colors.red[200], alignment: Alignment.center,
        child: const Text('Expanded, flex = 2'),),
    ),
])
```



Flutter Spacing

Padding vs Margin

- The padding property is a common property in Flutter widgets that allows you to add space around the widget.
- The value of the padding property is specified using an EdgeInsets object, which represents the amount of space to add on each side of the widget.

```
Container(
    padding: EdgeInsets.all(16.0),
    child: Text("Hello, World!"),
)
```

- In this example, the Container widget has a padding value of EdgeInsets.all(16.0), which means that 16.0 units of space will be added to all sides of the widget

Padding vs Margin

- You can also specify different values for the padding property for each side, for example:

```
Container(
    padding: EdgeInsets.only(left: 16.0, right: 16.0),
    child: Text("Hello, World!"),
)
```

- In this case, the Container widget has a padding value of EdgeInsets.only(left: 16.0, right: 16.0), which means that 16.0 units of space will be added to the left and right sides of the widget, but no space will be added to the top or bottom.

Padding vs Margin

- You can also specify all values at once in the following order: left, top, right, bottom:

```
Container(
    padding: EdgeInsets.fromLTRB(10, 20, 30, 40),
    child: Text("Hello, World!"),
)
```

- In this case, the Container widget has a padding value of 10px for the left, 20px for the top, 30px for the right and 40px for the bottom.

Padding vs Margin

- In addition, you can also provide the same padding value for vertical and horizontal by doing the following:

```
Container(
    padding: EdgeInsets.symmetric(10, 20),
    child: Text("Hello, World!"),
)
```

- In this case, the Container widget has a padding value of 10px on the top and bottom, 20px on the left and right.

Padding vs Margin

- In Flutter, **Padding** is a widget that can be used to add space around another widget. It's essentially a container that provides a specified amount of space on each side of its child widget.
- The Padding widget can be used to add space around a widget to separate it from other widgets or from the edge of the screen.

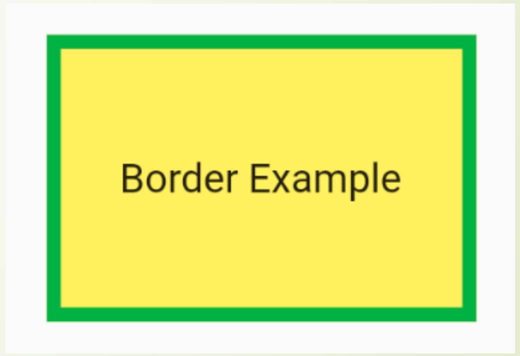
```
Padding(
    padding: EdgeInsets.all(16.0),
    child: Container(
        color: Colors.blue,
        child: Text("Hello, World!"),
    ),
)
```

Border

- The Border.all() method is used to create a border that has a uniform appearance on all four sides of a widget. It is part of the Border class, which is used to define the decorative border of a BoxDecoration widget.
- The Border.all() method takes several arguments to customize the appearance of the border. Some of these arguments include:
  - width: The width of the border in logical pixels.
  - color: The color of the border.
  - style: The style of the border. Possible values include BorderStyle.solid, BorderStyle.none, BorderStyle.dotted, etc.

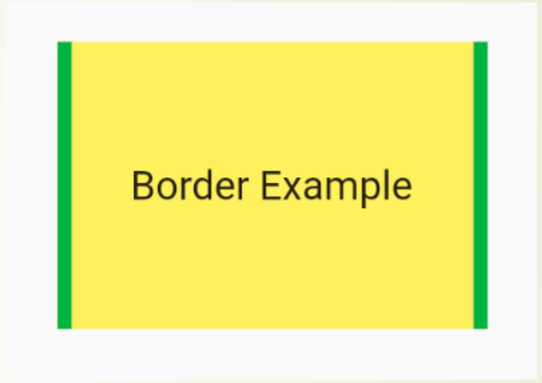
Border

```
Container(
    width: 150,
    height: 100,
    decoration: BoxDecoration(
        color: Colors.yellow[300],
        border: Border.all(width: 5, color: Colors.green)
    ),
    alignment: Alignment.center,
    child: const Text('Border Example'),
)
```



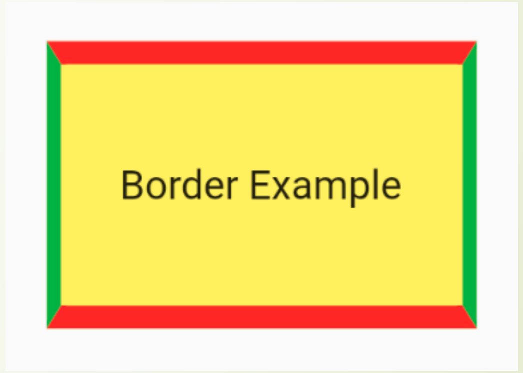
Border

```
Container(
    width: 150,
    height: 100,
    decoration: BoxDecoration(
        color: Colors.yellow[300],
        border: const Border(
            left: BorderSide(width: 5, color: Colors.green),
            right: BorderSide(width: 5, color: Colors.green),
        )
    ),
    alignment: Alignment.center,
    child: const Text('Border Example'),
)
```



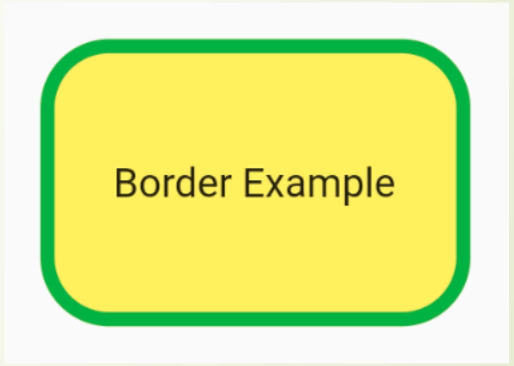
Border

```
Container(
    width: 150,
    height: 100,
    decoration: BoxDecoration(
        color: Colors.yellow[300],
        border: const Border.symmetric(
            vertical: BorderSide(width: 5, color: Colors.green),
            horizontal: BorderSide(width: 8, color: Colors.red),)
    ),
    alignment: Alignment.center,
    child: const Text('Border Example'),
)
```



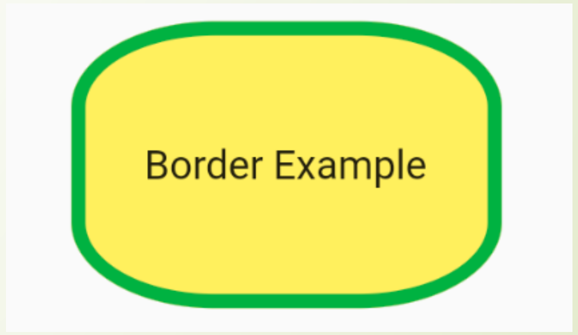
Border Radius

```
Container(
    width: 150,
    height: 100,
    decoration: BoxDecoration(
        color: Colors.yellow[300],
        border: Border.all(width: 5, color: Colors.green),
        borderRadius: BorderRadius.circular(24)
    ),
    alignment: Alignment.center,
    child: const Text('Border Example'),
)
```



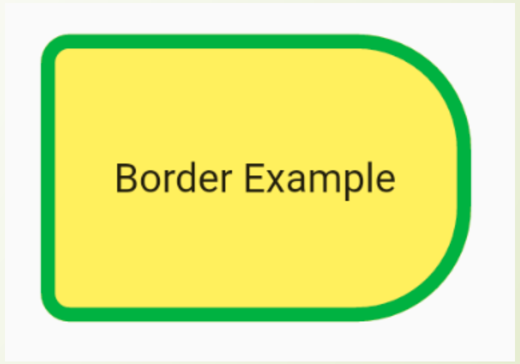
Border Radius

```
Container(
    width: 150,
    height: 100,
    decoration: BoxDecoration(
        color: Colors.yellow[300],
        border: Border.all(width: 5, color: Colors.green),
        borderRadius: const BorderRadius.all(Radius.elliptical(50, 30),)
    ),
    alignment: Alignment.center,
    child: const Text('Border Example'),
)
```


Border Example

Border Radius

```
Container(
     width: 150,
     height: 100,
     decoration: BoxDecoration(
          color: Colors.yellow[300],
          border: Border.all(width: 5, color: Colors.green),
          borderRadius: const BorderRadius.horizontal(
               left: Radius.circular(10),
               right: Radius.circular(40)),)
     ),
     alignment: Alignment.center,
     child: const Text('Border Example'),
)
```


Border Example