Flutter Widgets and Layouts

Introduction to Flutter

Introduction to Widgets

Widgets is an immutable object that describes a specific part of a UI **given** their **current configuration** and **state**.

Widgets are classes used to build UIs

Widgets are used for both **layout** and UI **elements**

You can compose simple widgets to build complex widgets.

When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description in order to **determine the minimal changes** needed in the underlying render tree to transition from one state to the next.

Minimal Flutter App

```
import 'package:flutter/material.dart';

void main() {
  runApp(
    Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  );
}
```

The `runApp()` function takes the given Widget and makes it the root of the widget tree.

This example consists of two widgets, the **`Center`** widget and its child, the **`Text`** widget

Commonly Used Basic widgets

`Text`

The **`Text`** widget lets you create a run of styled text within your application

`Row`, `Column`

These **flex** widgets let you create flexible layouts in both the horizontal (**`Row`**) and vertical (**`Column`**) directions

The design of these objects is based on the web's flexbox layout model.

Commonly Used Basic widgets

`Stack`

A Stack widget lets you place widgets on top of each other in paint order.

You can then use the **Positioned** widget on children of a Stack to position them relative to the **top**, **right**, **bottom**, or **left** edge of the stack.

Stacks are based on the web's absolute positioning layout model

Commonly Used Basic widgets

`Container`

The `Container` widget lets you create a rectangular visual element

A container can be decorated with a `BoxDecoration`, such as a **background**, a **border**, or a **shadow**.

A `Container` can also have **margins**, **padding**, and **constraints** applied to its size.

A `Container` can be transformed in three dimensional space using a matrix.

Commonly Used Basic widgets

Text Widget

```
    Text(
           ‘Example title',
           style: Theme.of(context).primaryTextTheme.headline6,
       )
```

Commonly Used Basic widgets

Row Widget

```
Row(
    children: <Widget>[
      IconButton(
        icon: Icon(Icons.menu),
        tooltip: 'Navigation menu',
        onPressed: null, // null disables the button
      ),

      IconButton(
        icon: Icon(Icons.search),
        tooltip: 'Search',
        onPressed: null,
      ),
    ],
)
```

Commonly Used Basic widgets

Column Widget

```
Column(
        children: <Widget>[
          MyAppBar(
            title: Text(
              'Example title',
              style: Theme.of(context).primaryTextTheme.headline6,
            ),
          ),
          Expanded(
            child: Center(
              child: Text('Hello, world!'),
            ),
          ),
        ],
    )
```

Using Material Components

Flutter provides a number of widgets that help you build apps that follow Material Design

A Material app starts with the `MaterialApp` widget, which builds a number of useful widgets at the root of your app

```
void main() {
  runApp(MaterialApp(
    home: TutorialHome(),
  ));
}
```

Handling User Interaction

```
class MyButton extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTap: () {
        print('MyButton was tapped!');
      },
      child: Container(),
      ),
    );
  }
}
```

The **GestureDetector** widget doesn’t have a visual representation but instead detects gestures made by the user

You can use **GestureDetector** to detect a variety of input gestures, including taps, drags, and scales.

When the user taps the Container, the **GestureDetector** calls its on**Tap()** callback

Stateless and Stateful Widgets

When writing an app, you'll commonly create new widgets that are subclasses of either `StatelessWidget` or `StatefulWidget`

`StatefulWidget` are special widgets that know how to generate `State` objects, which are then used to **hold state**.

A widget's main job is to implement a `build()` function, which describes the widget in terms of other lower-level widgets

The flutter framework builds those widgets in turn until the process bottoms out in widgets that represent the underlying `RenderObject`, which computes and describes the geometry of the widget

Stateless Widgets

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Material(
      child: Container (),
   );
  }
}
```

```
void main() {
  runApp(MaterialApp(
    title: 'My app',
    home: SafeArea(
      child: MyWidget(),
    ),
  ));
}
```

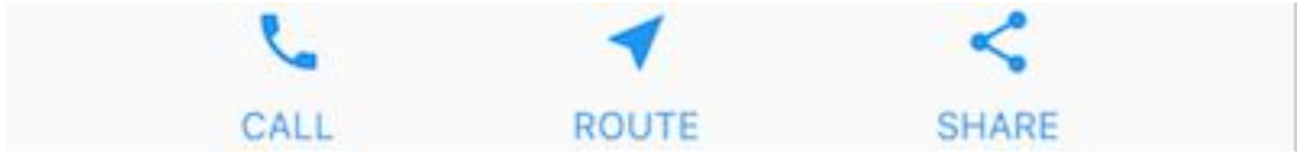
Statefull Widgets

```
class Counter extends StatefulWidget {
  @override
  _CounterState createState() =>
_CounterState();
}
```

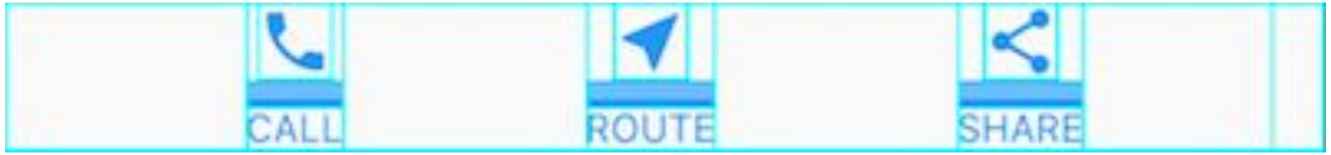
```
class _CounterState extends State<Counter>
{
  int _counter = 0;

  void _increment() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
      return Row(
      children: <Widget>[
        ElevatedButton(
          onPressed: _increment,
          child: Text('Increment'),
        ),
        Text('Count: $_counter'),
      ],
    );
  }
}
```

Composing Widgets

The following diagram is composed of 14 widgets

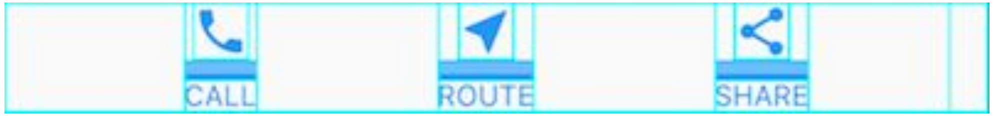


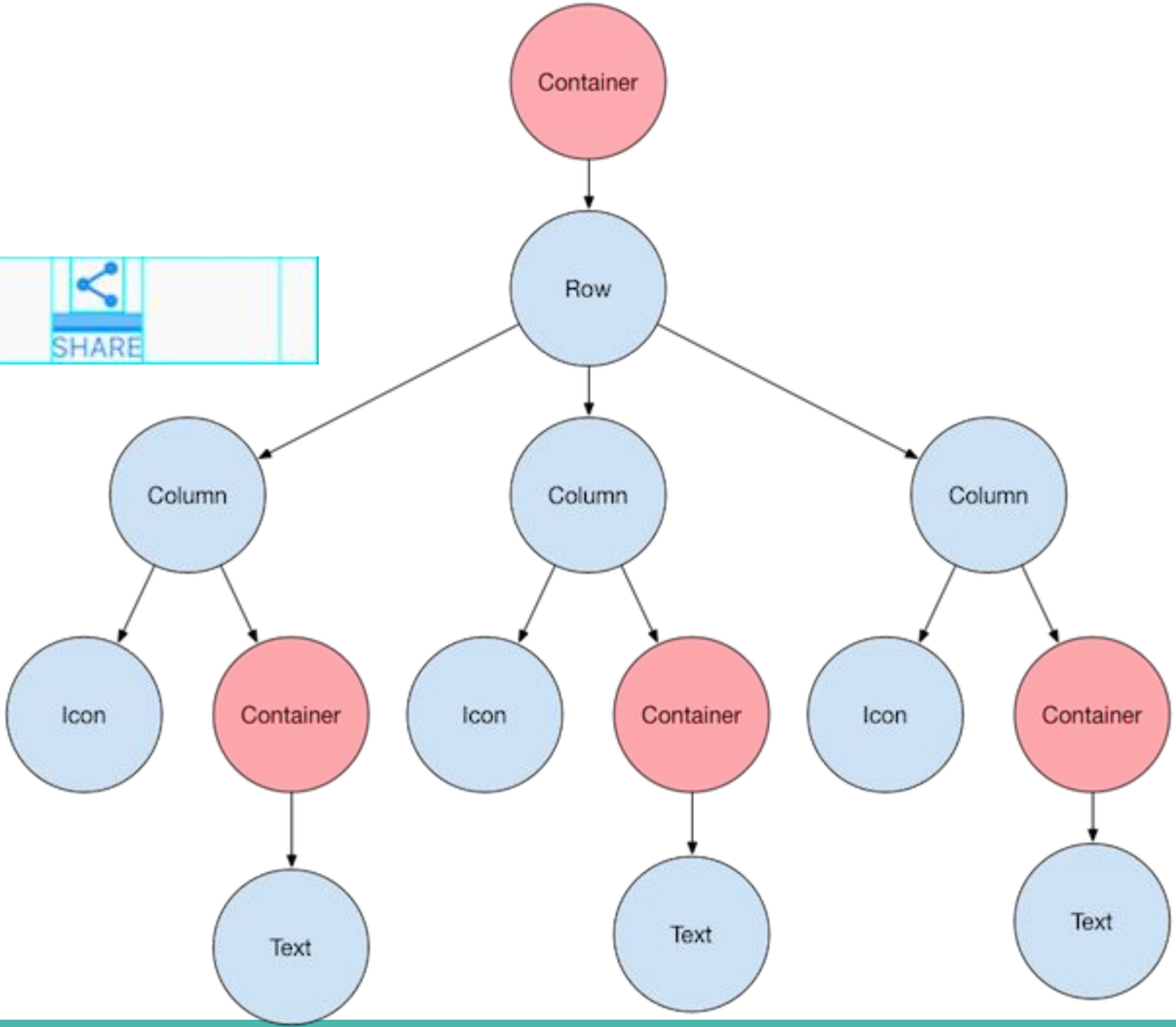
The following diagram highlights the invisible widgets



The widget tree for this is shown in the next slide

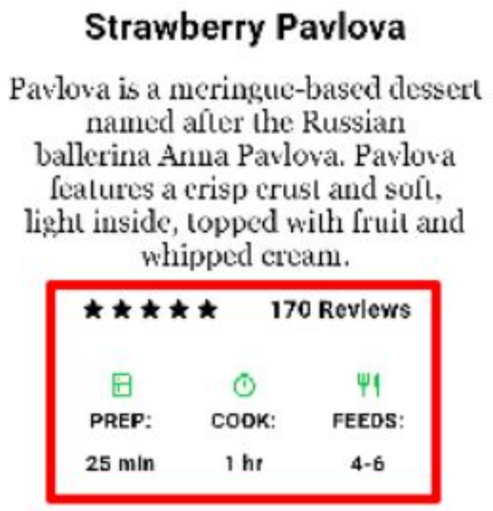
Widget Tree


CALL
ROUTE
SHARE


Container
Row
Column
Column
Column
Icon
Container
Icon
Container
Icon
Container
Text
Text
Text

Widget Tree

The widget tree for the ratings row

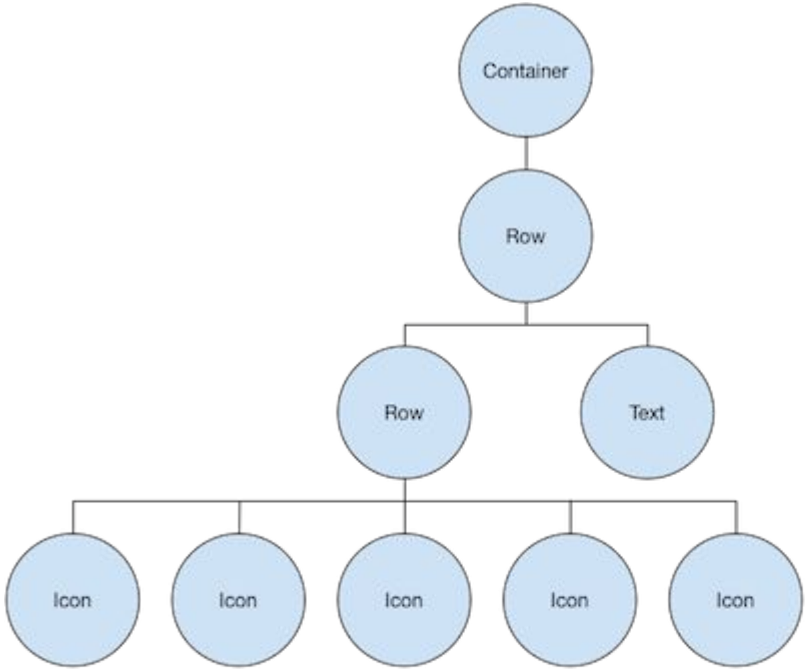


Strawberry Pavlova

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

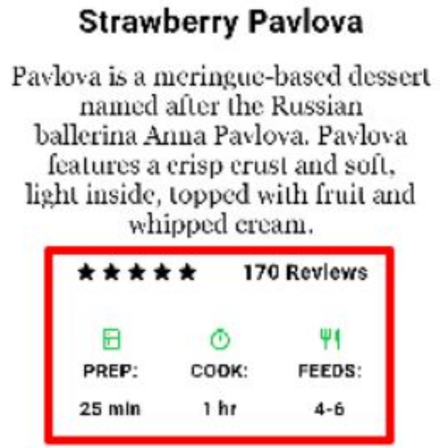
PREP:	COOK:	FEEDS:
25 min	1 hr	4-6



$$\begin{tikzcd}
 & & \text{Container} \arrow[d, no head] & & \\
 & & \text{Row} \arrow[dl, no head] \arrow[dr, no head] & & \\
 & \text{Row} \arrow[dl, no head] \arrow[d, no head] \arrow[dr, no head] \arrow[drr, no head] \arrow[dll, no head] & & \text{Text} & \\
\text{Icon} & \text{Icon} & \text{Icon} & \text{Icon} & \text{Icon}
\end{tikzcd}$$

Widget Tree

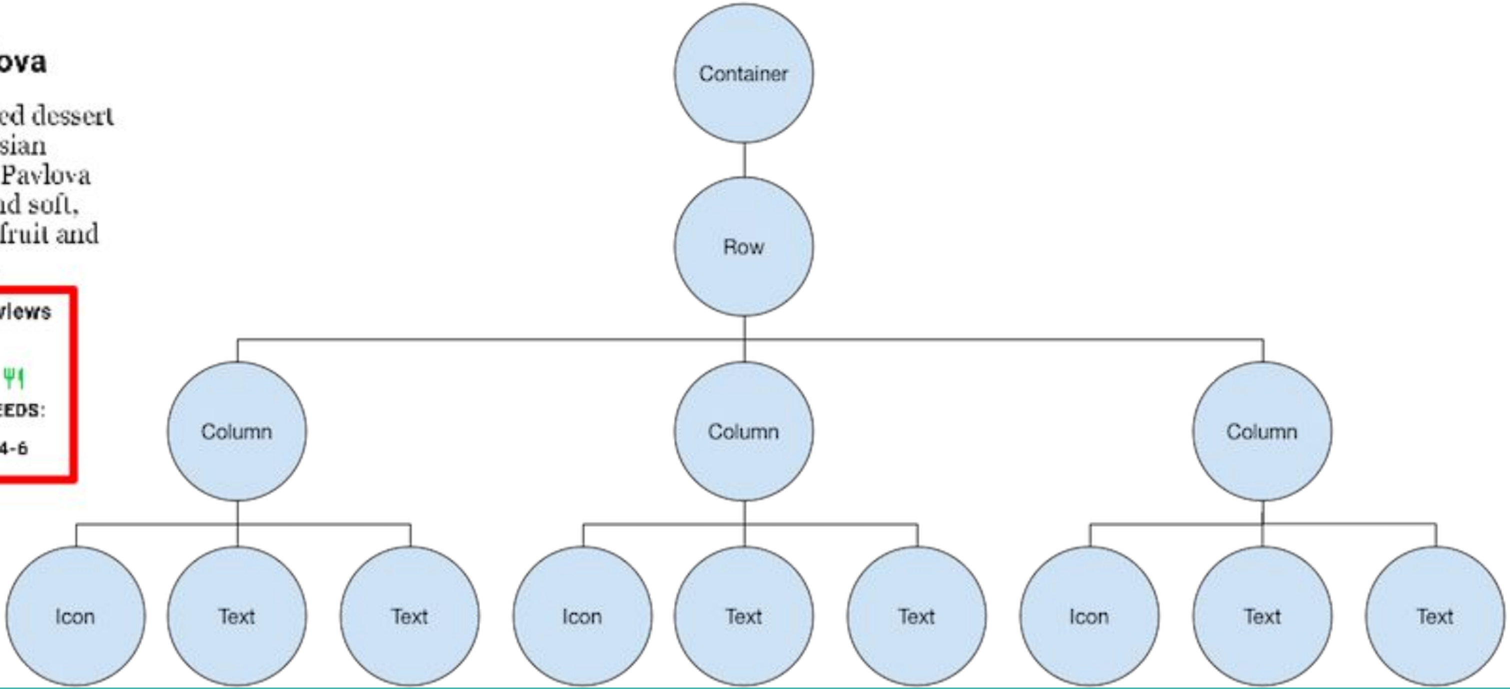
The widget tree for the icons row


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:	COOK:	FEEDS:
25 min	1 hr	4-6



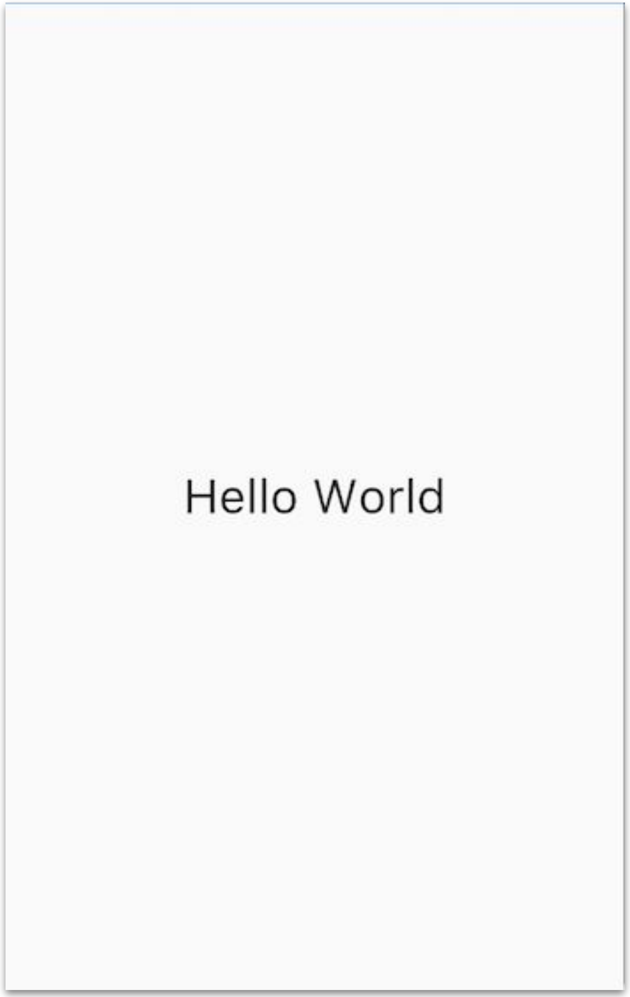
Layouts in Flutter

The core of **Flutter's layout** mechanism is widgets

Rows, columns, and grids that **arrange**, **constrain**, and **align** the visible widgets are also widgets

How to Lay out a single Widget

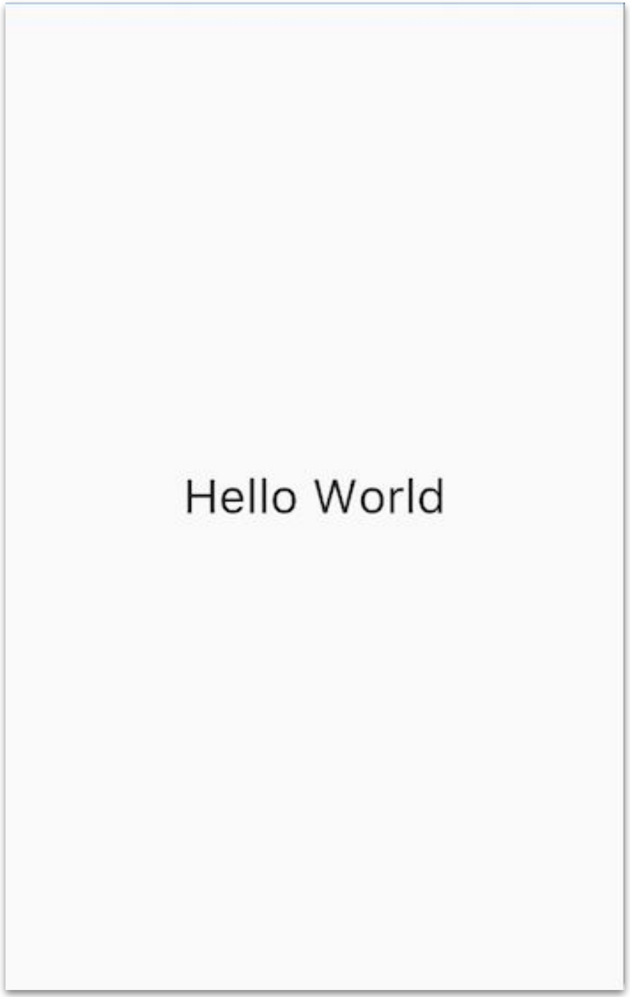
Let us see as an example how to lay out the text widget shown in the right



How to Lay out a single Widget

Step 1: Select a layout widget

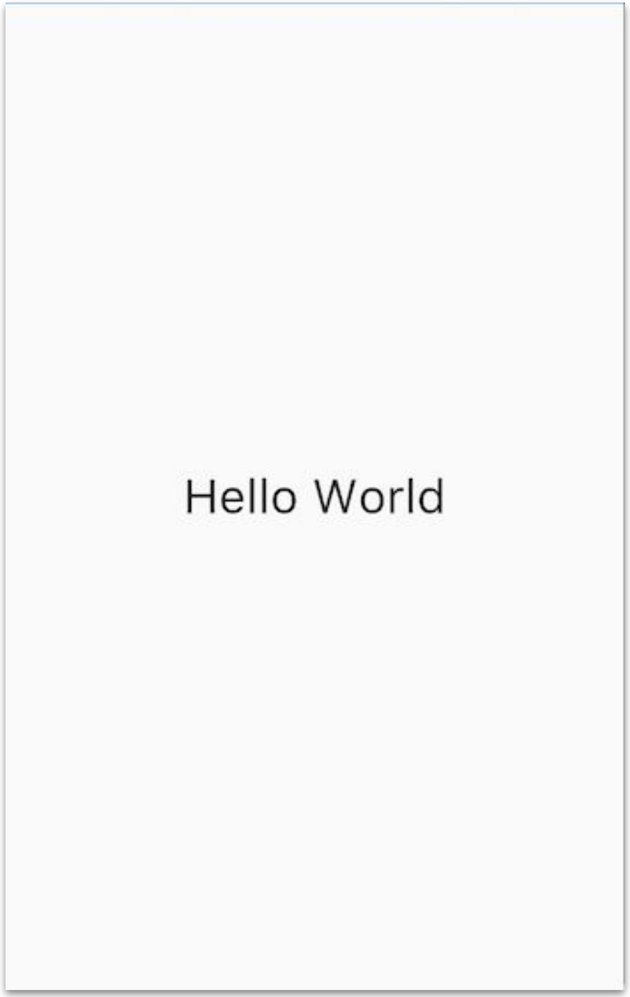
You can select `Center` layout widget which centers its content horizontally and vertically



How to Lay out a single Widget

Step 2: Create the visible widget you want to lay out

```
Text('Hello World')
```



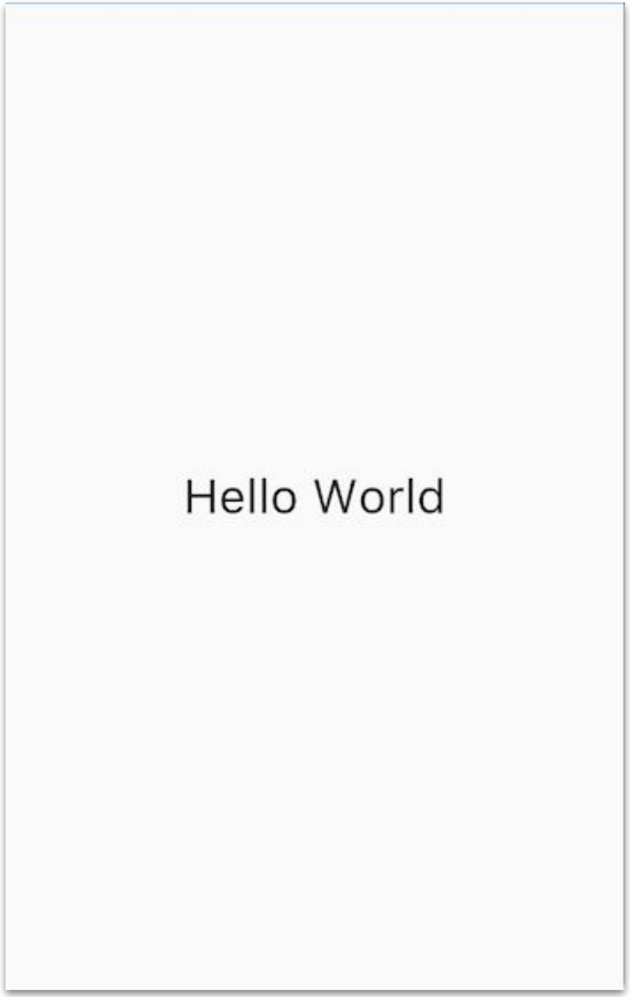
Hello World

How to Lay out a single Widget

Step 3: Add the visible widget to the layout widget

All layout widgets have either of the following:

> A **`child`** property if they take a single child—for example, `Center` or `Container`
>
> A **`children`** property if they take a list of widgets—for example, `Row`, `Column`, `ListView`, or `Stack`

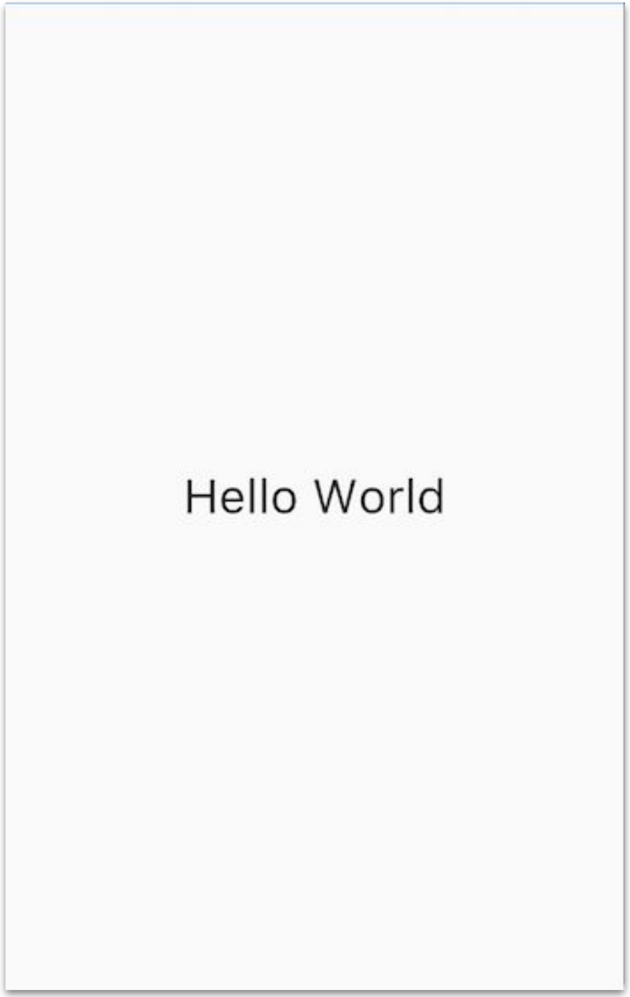


How to Lay out a single Widget

Step 3: Add the visible widget to the layout widget

Add the **`Text`** widget to the **`Center`** widget:

```
Center(
  child: Text('Hello World'),
)
```



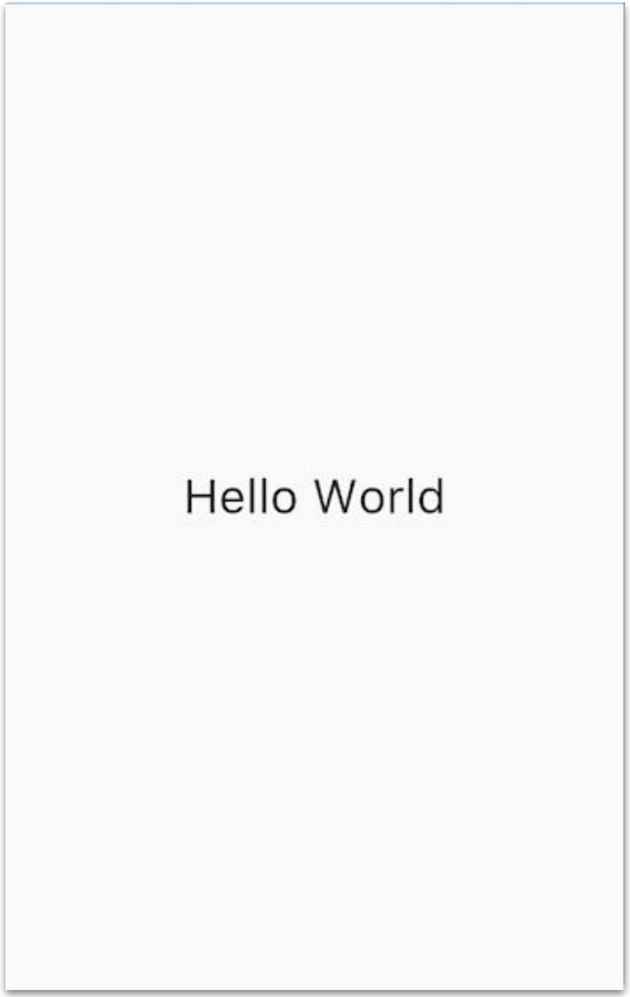
Hello World

How to Lay out a single Widget

Step 4: Add the layout widget to the page

A Flutter app is itself a widget, and most widgets have a **`build()`** method

Instantiating and returning a widget in the app's **`build()`**method displays the widget


Hello World

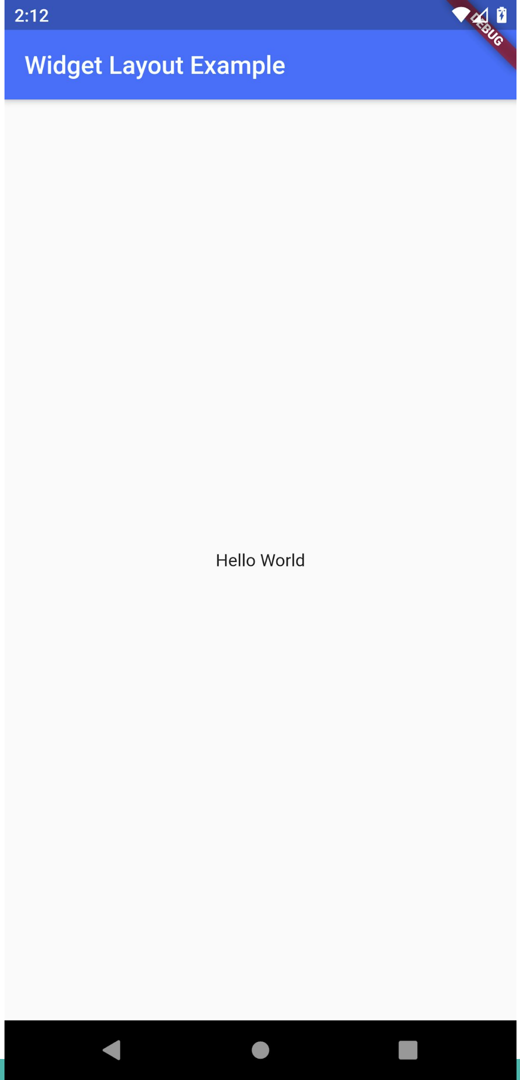
How to Lay out a single Widget

Step 4: Add the layout widget to the page

For a Material app, you can use a `Scaffold` widget

`Scaffold` widget provides a default **banner**, **background color**, and has API for adding drawers, snack bars, and bottom sheets

You can then add the `Center` widget directly to the **body** property for the home page



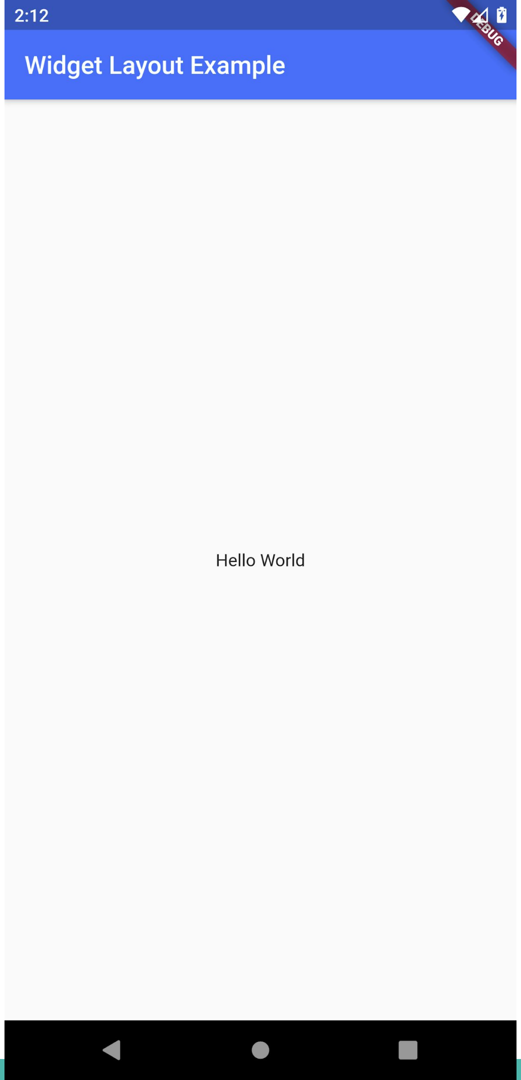
Widget Layout Example

Hello World

How to Lay out a single Widget

Step 4: Add the layout widget to the Material App

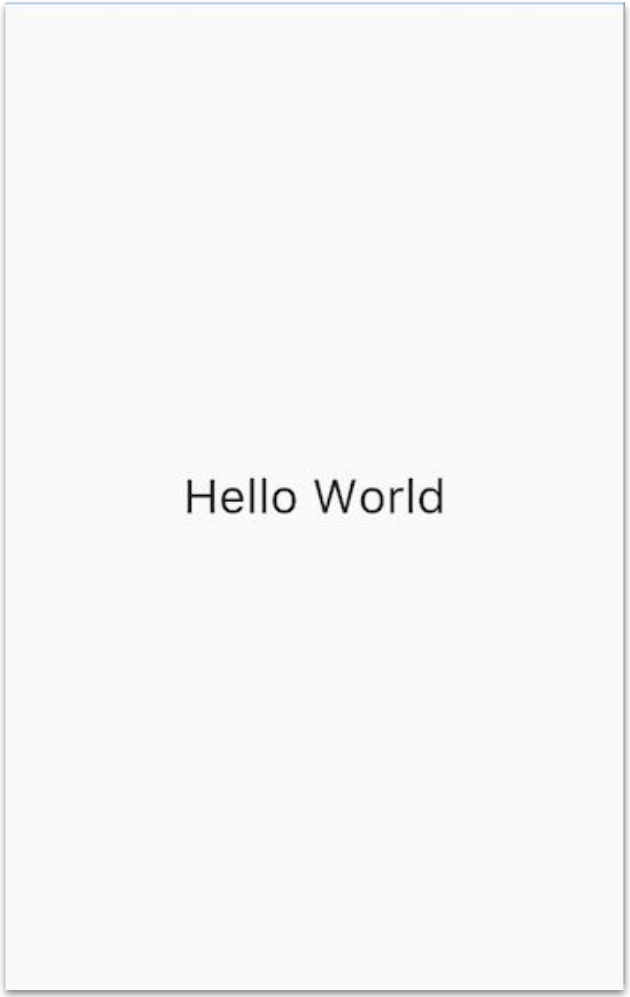
```
class MyApp extends StatelessWidget {
      @override
      Widget build(BuildContext context) {
        return MaterialApp(
          home: Scaffold(
            appBar: AppBar(
              title: Text('Layout demo'),
            ),
            body: Center(
              child: Text('Hello World'),
            ),
          ),
        );
      }
    }
```



How to Lay out a single Widget

Step 4: Add the layout widget to the page

For a non-Material app, you can add the `Center` widget to the app's `build()` method

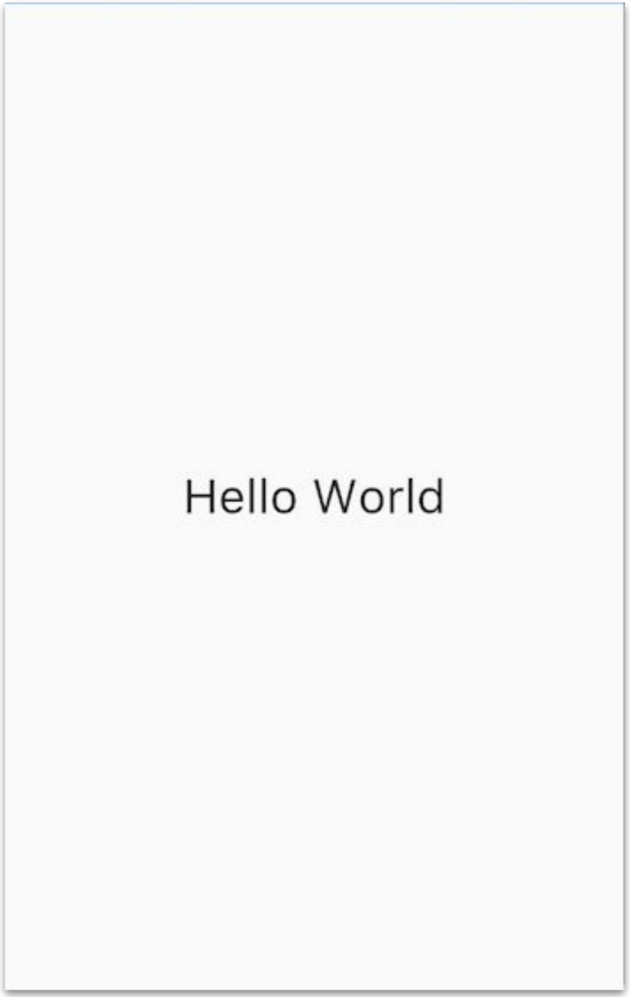


Hello World

How to Lay out a single Widget

Step 4: Add the layout widget to non-Material app

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: BoxDecoration(color: Colors.white),
      child: Center(
        child: Text(
          'Hello World',
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 32,
            color: Colors.black87,
          ),
```



Lay out multiple widgets

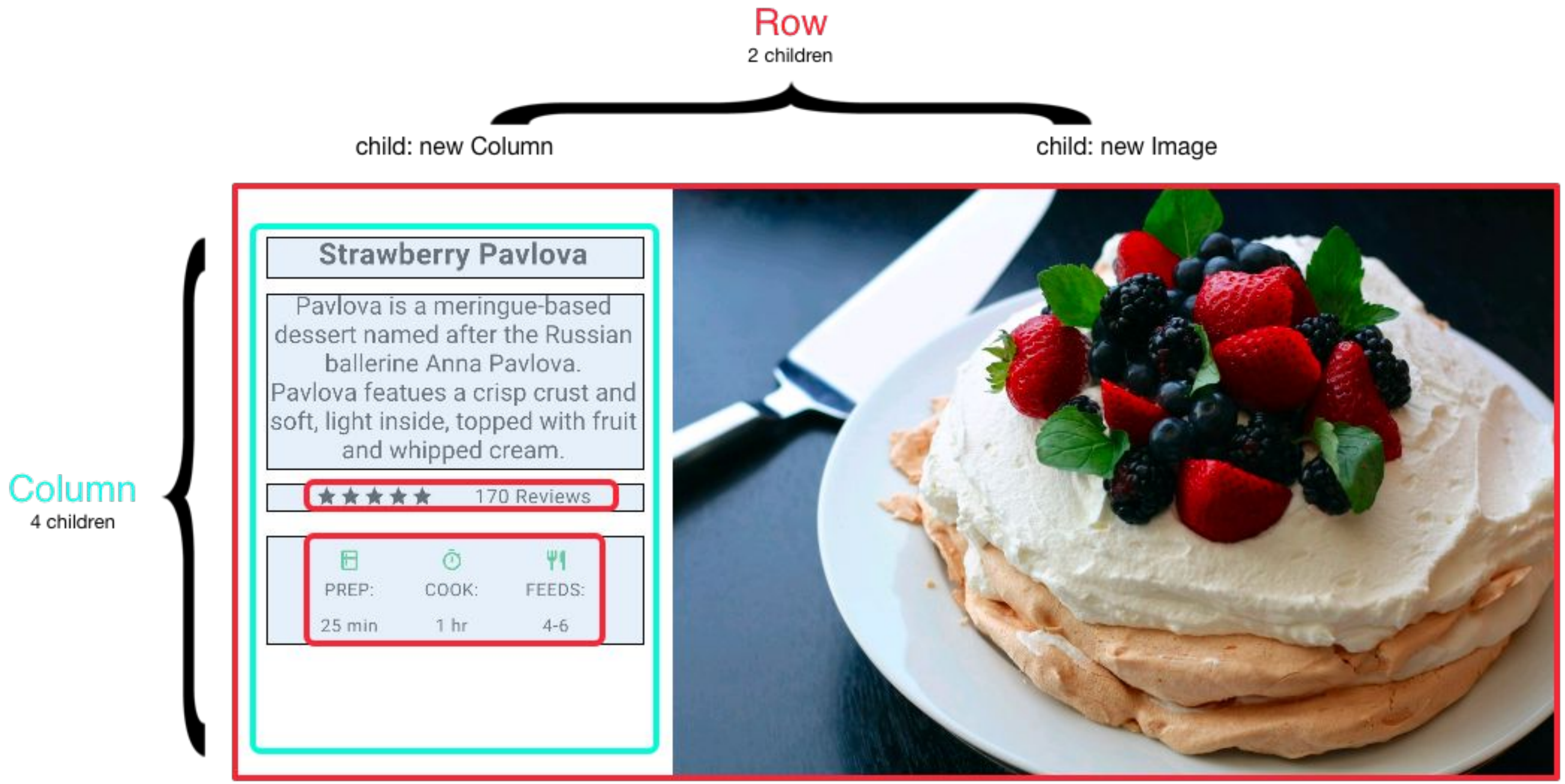
One of the most common layout patterns is to **arrange widgets vertically or horizontally**

You can use a `Row` widget to **arrange widgets horizontally**, and a `Column` widget to **arrange widgets vertically**

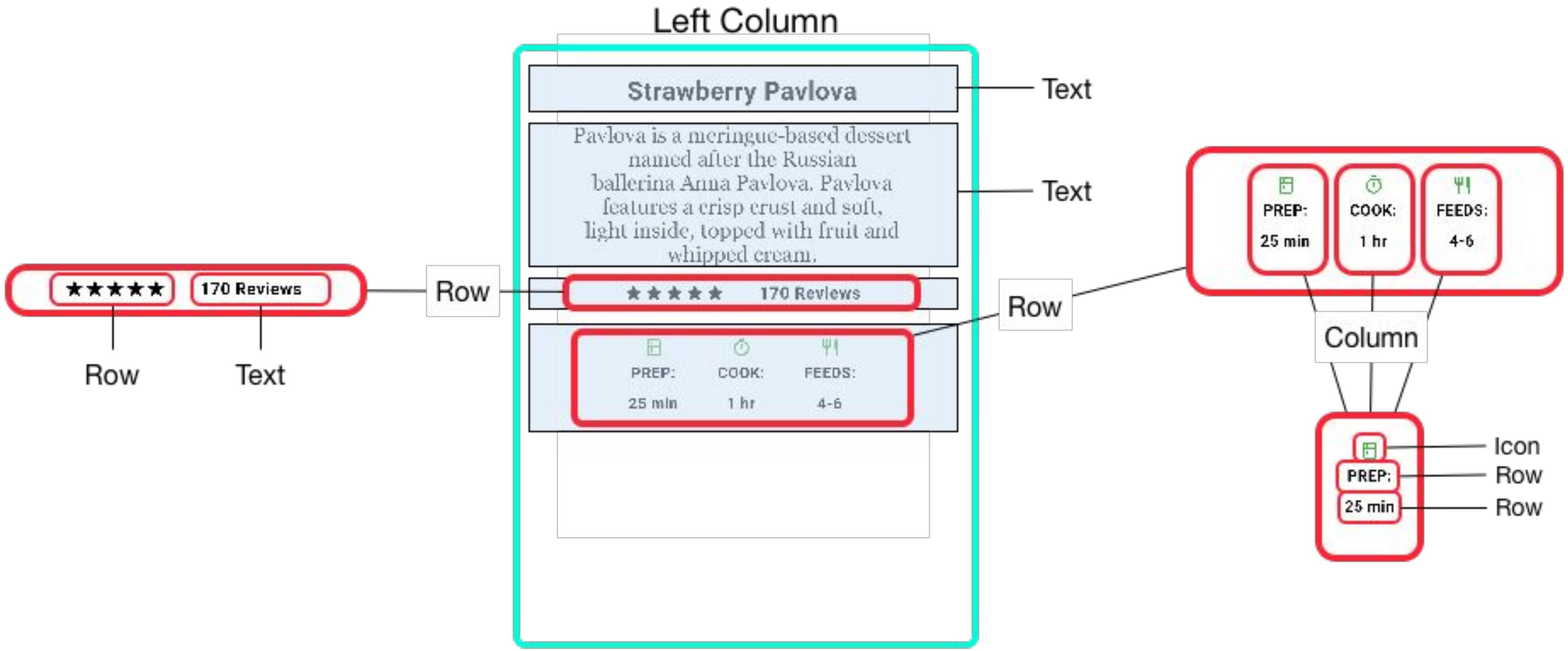
`Row` and `Column` each take a list of child widgets

A child widget can itself be a `Row`, `Column`, or other complex widget

Lay out multiple widgets


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

Lay out multiple widgets

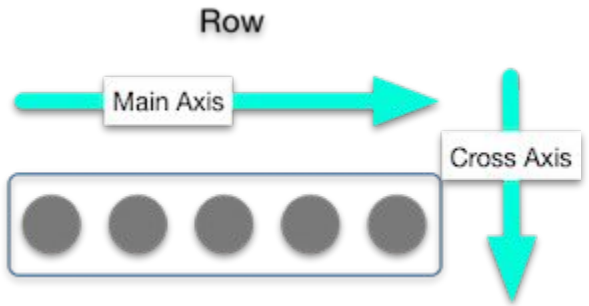

Left Column
Strawberry Pavlova
Text
Pavlova is a meringue-based dessert named after the Russian ballerina Anna Pavlova. Pavlova features a crisp crust and soft, light inside, topped with fruit and whipped cream.
Text
★★★★★ 170 Reviews
Row
Row
Text
Row
PREP: 25 min
COOK: 1 hr
FEEDS: 4-6
Row
Column
Icon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

Aligning widgets

You control how a row or column aligns its children using the **`mainAxisAlignment`** and **`crossAxisAlignment`** properties

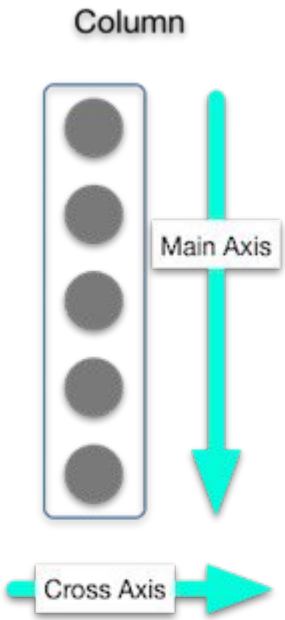
Aligning widgets

For a **row**, the **main axis runs horizontally** and the **cross axis runs vertically**



Aligning widgets

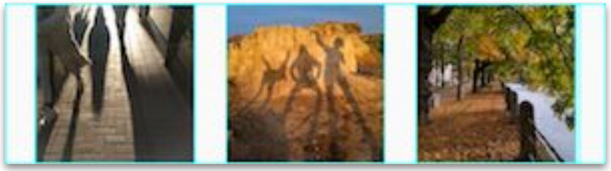
For a **column**, the **main axis runs vertically** and the **cross axis runs horizontally**



Aligning widgets

The **MainAxisAlignment** and **CrossAxisAlignment** classes offer a variety of constants for controlling alignment

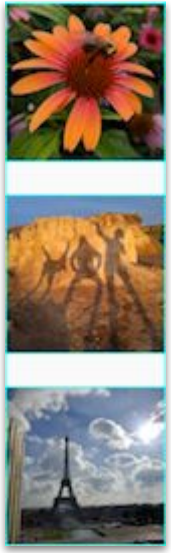
```
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Image.asset('images/pic1.jpg'),
    Image.asset('images/pic2.jpg'),
    Image.asset('images/pic3.jpg'),
  ],
);
```



Aligning widgets

The **`MainAxisAlignment`** and **`CrossAxisAlignment`** classes offer a variety of constants for controlling alignment

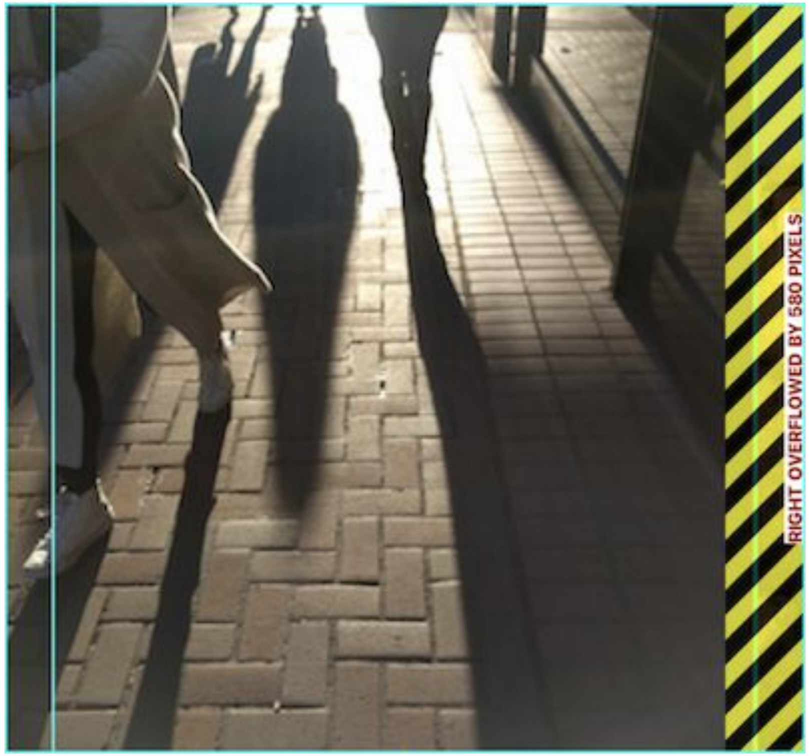
```
Column(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Image.asset('images/pic1.jpg'),
    Image.asset('images/pic2.jpg'),
    Image.asset('images/pic3.jpg'),
  ],
);
```



Sizing widgets

When a layout is too large to fit a device, a **yellow and black striped pattern** appears along the affected edge

Here is an example of a row that is too wide



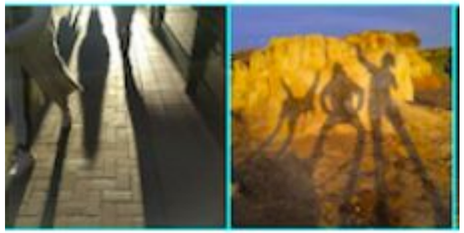
Sizing widgets

Widgets can be sized to fit within a **row/column** by using the **Expanded** widget

To fix the previous example wrap each image with an **Expanded** widget

Sizing widgets

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic2.jpg'),
    ),
  ],
);
```



Sizing widgets

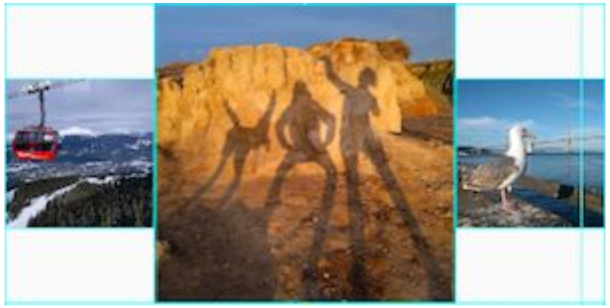
If you want a widget to occupy twice as much space as its siblings, for example, use the **Expanded** widget `flex` property, an integer that determines the flex factor for a widget

The default `flex` factor is 1

Sizing widgets

The following code sets the `flex` factor of the middle image to 2:

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ),
    Expanded(
      flex: 2,
      child: Image.asset('images/pic2.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic3.jpg'),
    ),
  ],
);
```

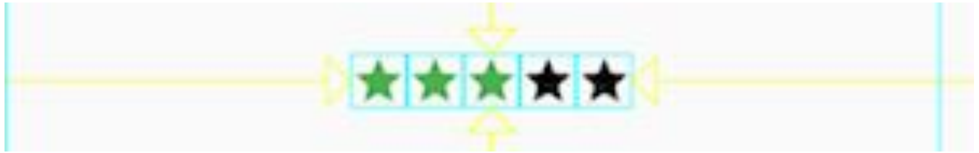


Packing widgets

By default, **a row or column occupies as much space along its main axis** as possible, but if you want to pack the children closely together, set its **`mainAxisSize`** to **`MainAxisSize.min`**

Packing widgets

```
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.black),
    Icon(Icons.star, color: Colors.black),
  ],
)
```



Common layout widgets

Standard widgets

`Container`: Adds padding, margins, borders, background color, or other decorations to a widget.

`GridView`: Lays widgets out as a scrollable grid.

`ListView`: Lays widgets out as a scrollable list.

`Stack`: Overlaps a widget on top of another.

Common layout widgets

Material widgets

`Card`: Organizes related info into a box with rounded corners and a drop shadow

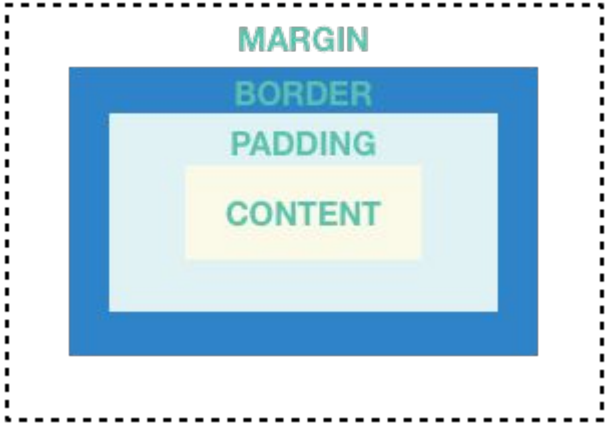
`ListTile`: Organizes up to 3 lines of text, and optional leading and trailing icons, into a row

Common layout widgets: `Container`

Add **padding**, **margins**, **borders**

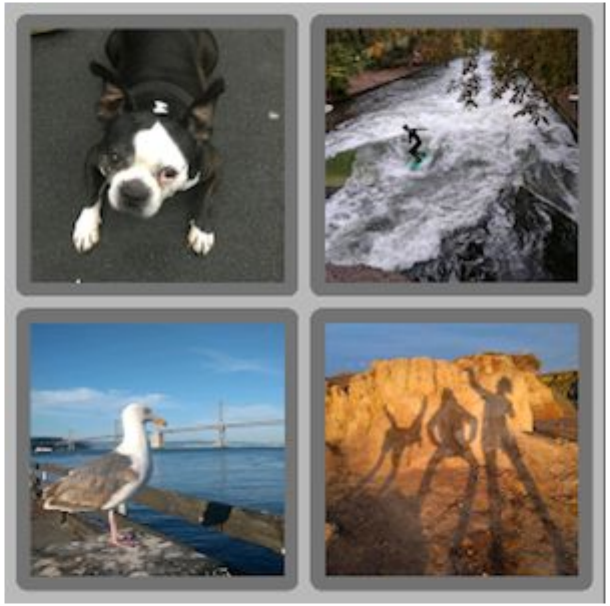
Change **background color** or **image**

Contains a **single child** widget, but that child can be a Row, Column, or even the root of a widget tree



Common layout widgets: Container

```
Widget _buildImageColumn() => Container(
      decoration: BoxDecoration(
        color: Colors.black26,
      ),
      child: Column(
        children: [
          _buildImageRow(1),
          _buildImageRow(3),
        ],
      ),
    );
```



Common layout widgets: `GridView`

Use `GridView` to lay widgets out as a **two-dimensional list**

`GridView` provides two pre-fabricated lists, or you can build your own custom grid.

When a `GridView` detects that its contents are too long to fit the render box, it automatically scrolls

Common layout widgets: `GridView`

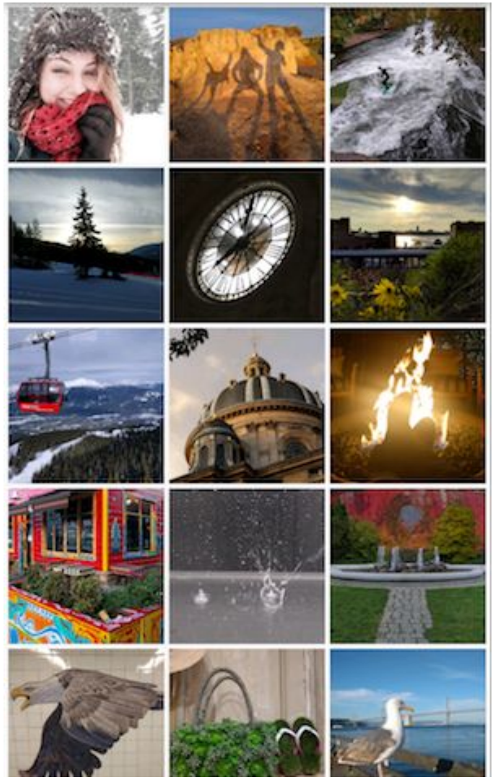
`GridView.count`

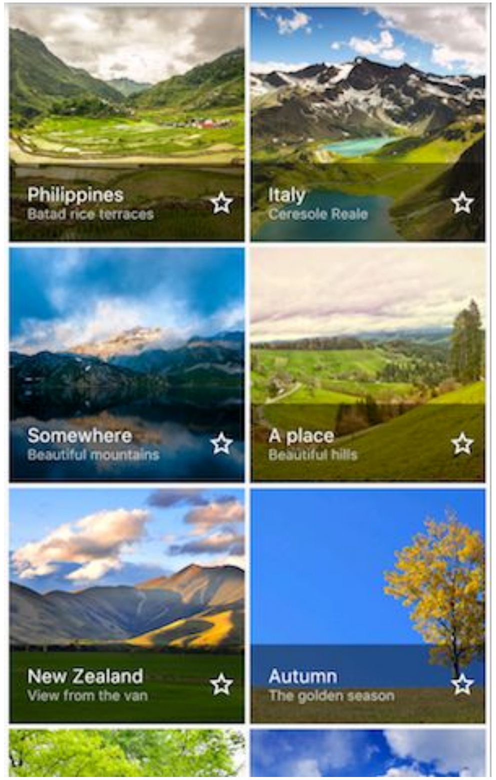
allows you to specify the number of columns

`GridView.extent`

allows you to specify the maximum pixel width of a tile

Common layout widgets: `GridView`





Common layout widgets: `GridView`

```
Widget _buildGrid() => GridView.extent(
    maxCrossAxisExtent: 150,
    padding: const EdgeInsets.all(4),
    mainAxisSpacing: 4,
    crossAxisSpacing: 4,
    children: _buildGridTileList(30));

List<Container> _buildGridTileList(int count) => List.generate(
    count, (i) => Container(child:
Image.asset('images/pic$i.jpg')));
```

Common layout widgets: `ListView`

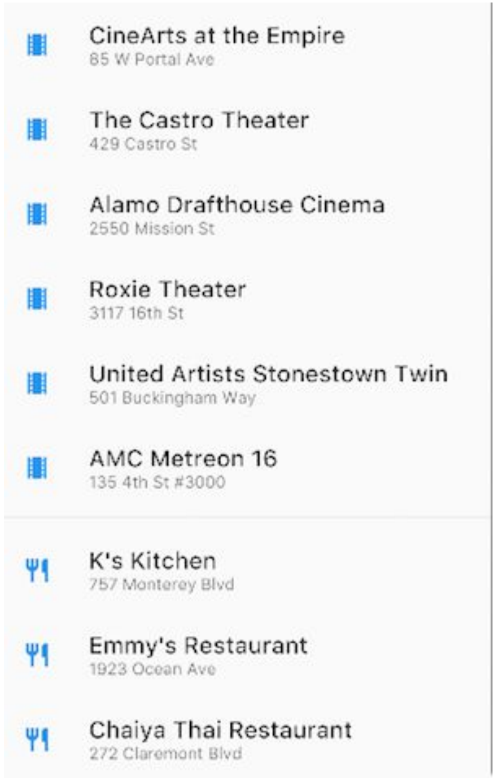
A specialized `Column` for organizing a list of boxes

Can be laid out **horizontally** or **vertically**

Detects when its content won’t fit and provides **scrolling**

Less configurable than `Column`, but easier to use and supports scrolling

Common layout widgets: `ListView`



CineArts at the Empire
85 W Portal Ave

The Castro Theater
429 Castro St

Alamo Drafthouse Cinema
2550 Mission St

Roxie Theater
3117 16th St

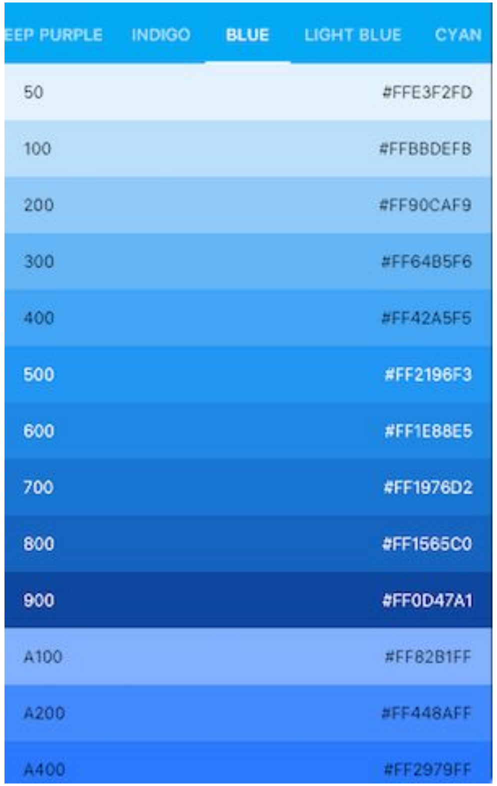
United Artists Stonestown Twin
501 Buckingham Way

AMC Metreon 16
135 4th St #3000

K's Kitchen
757 Monterey Blvd

Emmy's Restaurant
1923 Ocean Ave

Chaiya Thai Restaurant
272 Claremont Blvd



EEP PURPLE INDIGO BLUE LIGHT BLUE CYAN

50	#FFE3F2FD
100	#FFBBDEFB
200	#FF90CAF9
300	#FF64B5F6
400	#FF42A5F5
500	#FF2196F3
600	#FF1E88E5
700	#FF1976D2
800	#FF1565C0
900	#FF0D47A1
A100	#FF82B1FF
A200	#FF448AFF
A400	#FF2979FF

Common layout widgets: `ListView`

A specialized `Column` for organizing a list of boxes

Can be laid out **horizontally** or **vertically**

Detects when its content won’t fit and provides **scrolling**

Less configurable than `Column`, but easier to use and supports scrolling

Common layout widgets: `ListView`

```
Widget _buildList() => ListView(
      children: [
        _tile('CineArts at the Empire', '85 W Portal Ave',
Icons.theaters),
        _tile('The Castro Theater', '429 Castro St', Icons.theaters),
       ...
        Divider(),
        _tile('K\'s Kitchen', '757 Monterey Blvd', Icons.restaurant),
        _tile('Emmy\'s Restaurant', '1923 Ocean Ave', Icons.restaurant),

            ],
    );
```

Common layout widgets: ListView

```
ListTile _tile(String title, String subtitle, IconData icon) => ListTile(
      title: Text(title,
          style: TextStyle(
            fontWeight: FontWeight.w500,
            fontSize: 20,
          )),
      subtitle: Text(subtitle),
      leading: Icon(
        icon,
        color: Colors.blue[500],
      ),
    );
```

Common layout widgets: `Stack`

Use Stack to **arrange widgets on top of a base widget**—often an image

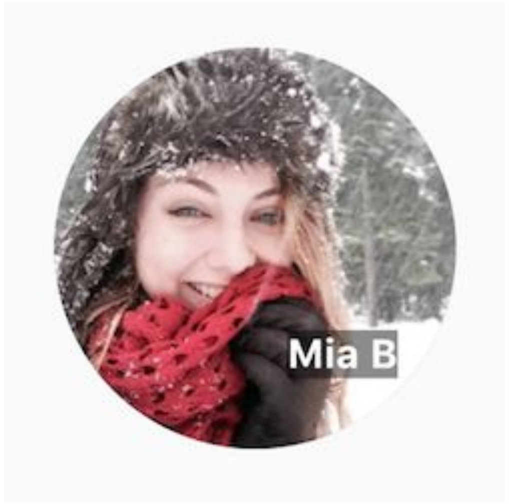
The widgets can **completely or partially overlap** the base widget

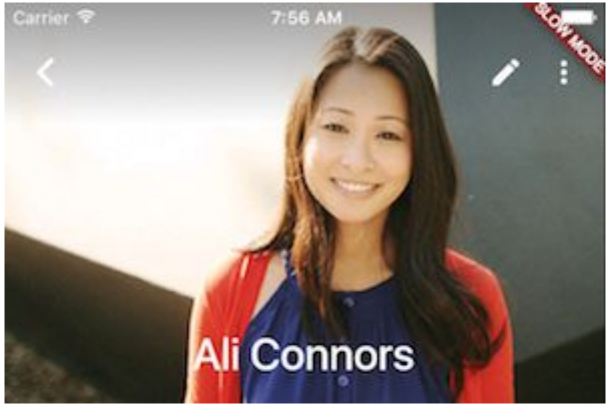
Use for widgets that overlap another widget

The first widget in the list of children is the base widget; subsequent children are overlaid on top of that base widget

A Stack’s content can’t scroll

Common layout widgets: `Stack`





Common layout widgets: `Stack`

```
Widget _buildStack() => Stack(
    alignment: const Alignment(0.6, 0.6),
    children: [
      CircleAvatar(
        backgroundImage: AssetImage('images/pic.jpg'),
        radius: 100,
      ),
      Container(
        decoration: BoxDecoration(
          color: Colors.black45,
        ),
        child: Text(
          'Mia B',
          style: TextStyle(
            fontSize: 20,
            fontWeight: FontWeight.bold,
            color: Colors.white,
          ),
        ),
      ),
    ],
  );
```

Common layout widgets: `Card`

Implements a **Material card**

Used for presenting related nuggets of information

Accepts a single child, but that child can be a Row, Column, or other widget that holds a list of children

Displayed with rounded corners and a drop shadow

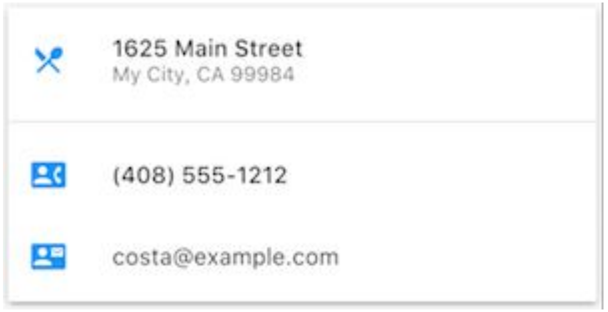
A Card’s content can’t scroll

Common layout widgets: `Card`

Often used with `ListTile`

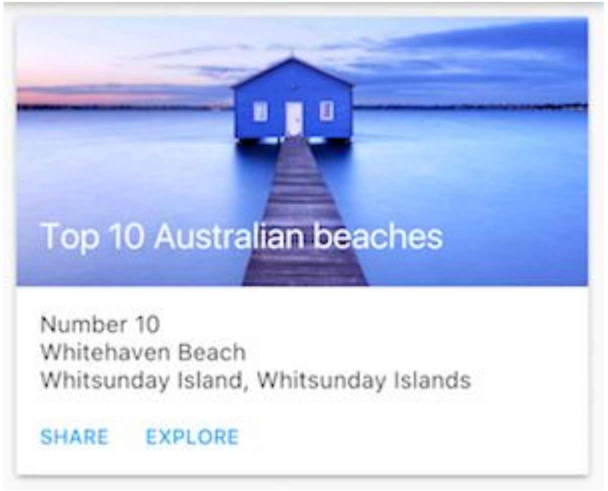
By default, a `Card` shrinks its size to 0 by 0 pixels

You can use `SizedBox` to constrain the size of a card


1625 Main Street
My City, CA 99984

(408) 555-1212

costa@example.com



Number 10
Whitehaven Beach
Whitsunday Island, Whitsunday Islands

Common layout widgets: Card

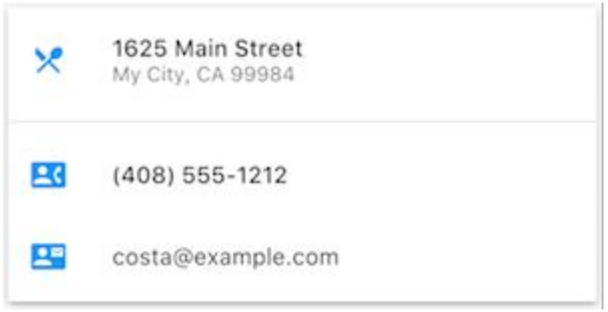
```
Widget _buildCard() => SizedBox(
    height: 210,
    child: Card(
      child: Column(
        children: [
          ListTile(
            title: Text('1625 Main Street',
                style: TextStyle(fontWeight: FontWeight.w500)),
            subtitle: Text('My City, CA 99984'),
            leading: Icon(
              Icons.restaurant_menu,
              color: Colors.blue[500],
            ),
          ),
        ],),
```

Common layout widgets: `ListTile`

A specialized row that contains up to 3 lines of text and optional icons

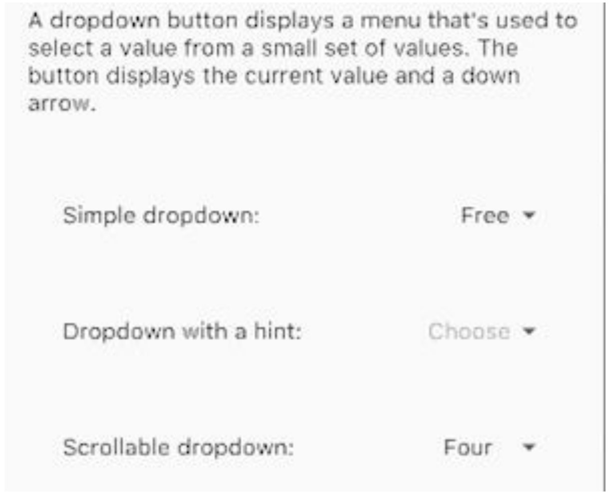
Less configurable than Row, but easier to use

From the Material library


1625 Main Street
My City, CA 99984

(408) 555-1212

costa@example.com


A dropdown button displays a menu that's used to select a value from a small set of values. The button displays the current value and a down arrow.

Simple dropdown: Free

Dropdown with a hint: Choose

Scrollable dropdown: Four