Flutter Fundamentals
App Layout

Peter Kassenaar –
info@kassenaar.com

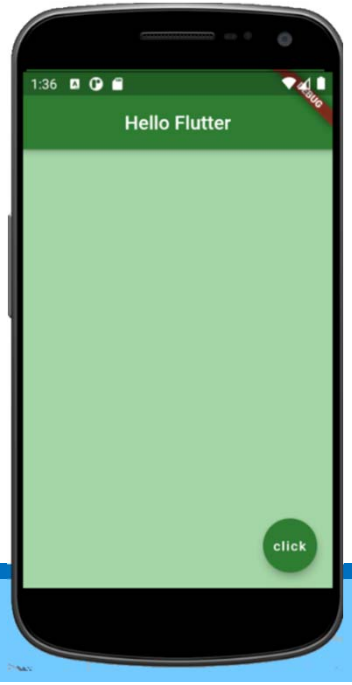
Layout of the app

Using Container, Margin and Padding to create more complex lay-outs

Container() Widget

- Is like a in HTML
- Container takes up all available space if it has no children
- If it *has* a child, the container is as big as the child

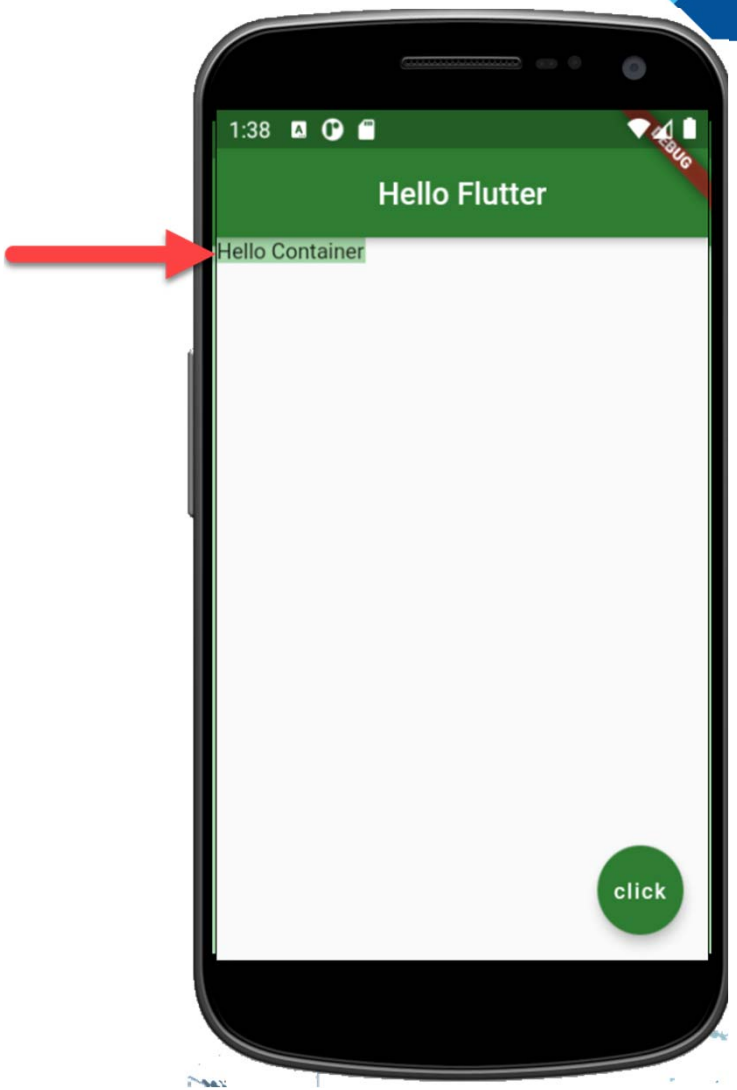
```
body: Container(
  // No child, so the container will fill
  // the complete background
  color: Colors.green[200],
),
```



child in the Container()

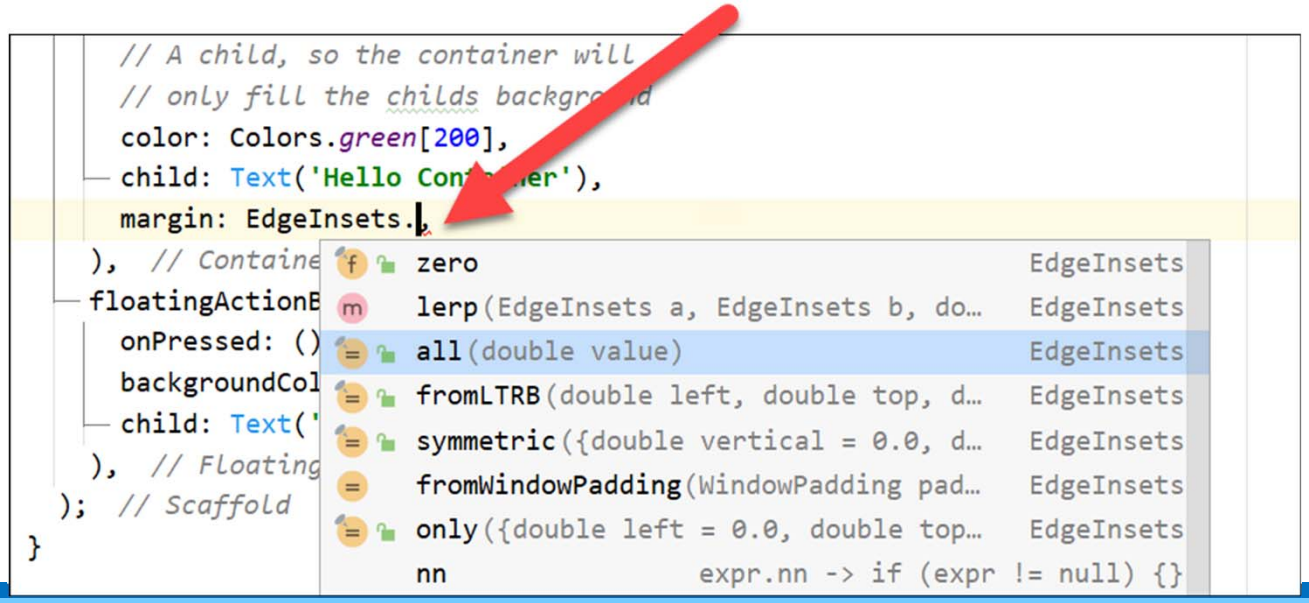
If there is a child in the container, the container takes up as much space as the child,

```
body: Container(
  // A child, so the container will
  // only fill the childs background
  color: Colors.green[200],
  child: Text('Hello Container'),
),
```



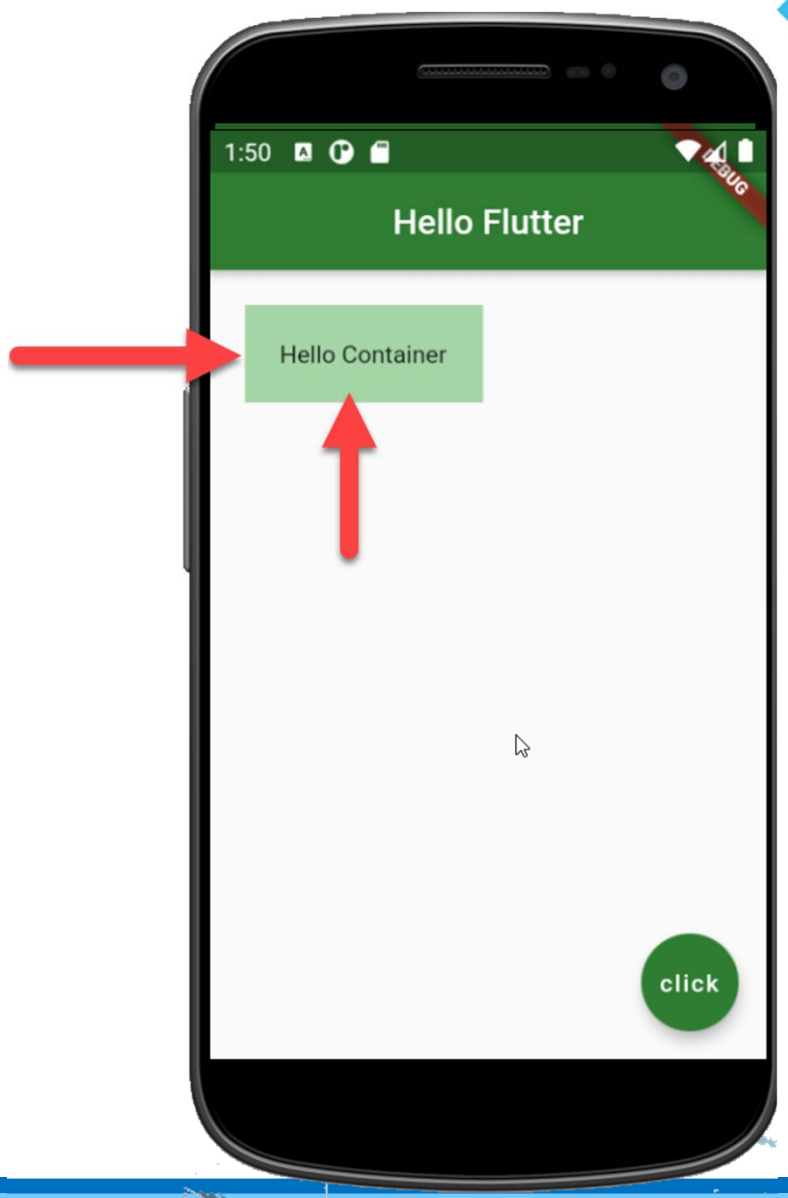
Controlling the padding and margin

- `padding` – whitespace within an element
- `margin` – whitespace around an element
- The amount of whitespace is controlled by an `EdgeInsets` widgets – with different values



Margin & padding in action

```
body: Container(
  // A child, so the container will
  // only fill the childs background
  color: Colors.green[200],
  child: Text('Hello Container'),
  margin: EdgeInsets.all(20.0),
  padding: EdgeInsets.all(20.0),
),
```



EdgeInsets

Flutter › painting.dart › EdgeInsets class

Search API Docs

painting library

CLASSES
Accumulator
Alignment
AlignmentDirectional
AlignmentGeometry
AssetBundleImageKey
AssetBundleImageProvi...
AssetImage
AutomaticNotchedShape
BeveledRectangleBorder
Border
BorderDirectional
BorderRadius
BorderRadiusDirectional
BorderRadiusGeometry
BorderSide
BoxBorder

EdgeInsets class

An immutable set of offsets in each of the four cardinal directions.

Typically used for an offset from each of the four sides of a box. For example, the padding inside a box can be represented using this class.

The EdgeInsets class specifies offsets in terms of visual edges, left, top, right, and bottom. These values are not affected by the TextDirection. To support both left-to-right and right-to-left layouts, consider using EdgeInsetsDirectional, which is expressed in terms of *start*, top, *end*, and bottom, where start and end are resolved in terms of a TextDirection (typically obtained from the ambient Directionality).

Here are some examples of how to create EdgeInsets instances:

Typical eight-pixel margin on all sides:

```
const EdgeInsets.all(8.0)
```

Eight pixel margin above and below, no horizontal margins:

CONSTRUCTORS
all
fromLTRB
fromViewPadding
~~fromWindowPaddi~~...
only
symmetric

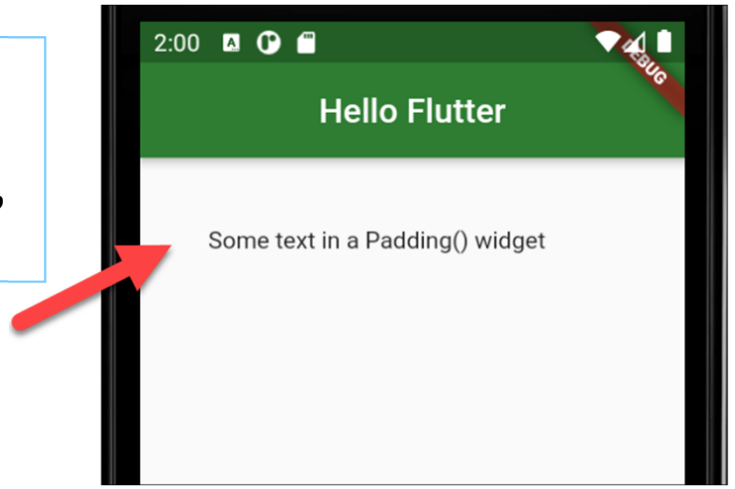
PROPERTIES
bottom
bottomLeft
bottomRight
collapsedSize
flipped
hashCode
horizontal
isNonNegative
left
right

api.flutter.dev/flutter/painting/EdgeInsets-class.html

Padding() Widget

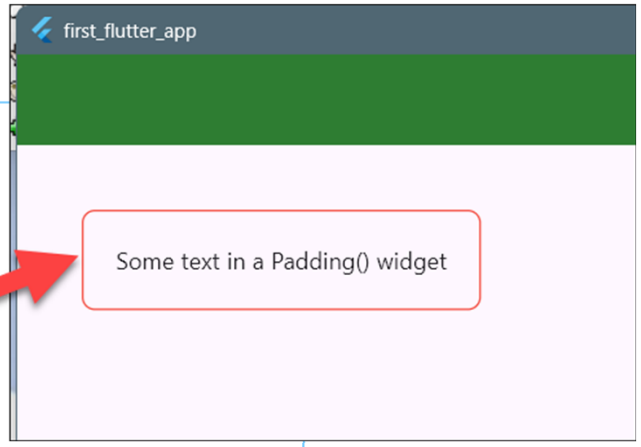
- We can also use a Padding() widget, if we just want some padding on an element.
- Padding() can NOT have margins, or (background) colors. Just padding. Otherwise, use Container()

```
body: Padding(
  padding: EdgeInsets.all(40.0),
  child: Text('Some text in a Padding() widget'),
),
```



Container() Widget cont'd

- If you need *just padding*, the Padding() widget is sufficient.
- If you need *padding and additional layout/styling features* (like a border, background color, or size constraints), the Container() widget is a better choice.




```
body: Container(
  decoration: BoxDecoration(
    border: Border.all(
      color: Colors.red, // Border color
      width: 1.0, // Border width
    ),
  ),
  margin: EdgeInsets.all(40.0),
  padding: EdgeInsets.all(20.0), // Add some padding inside the border
  child: Text('Some text in a Padding() widget'),
),
```

Rows

Dividing your layout in logical rows. Rows can have multiple childs

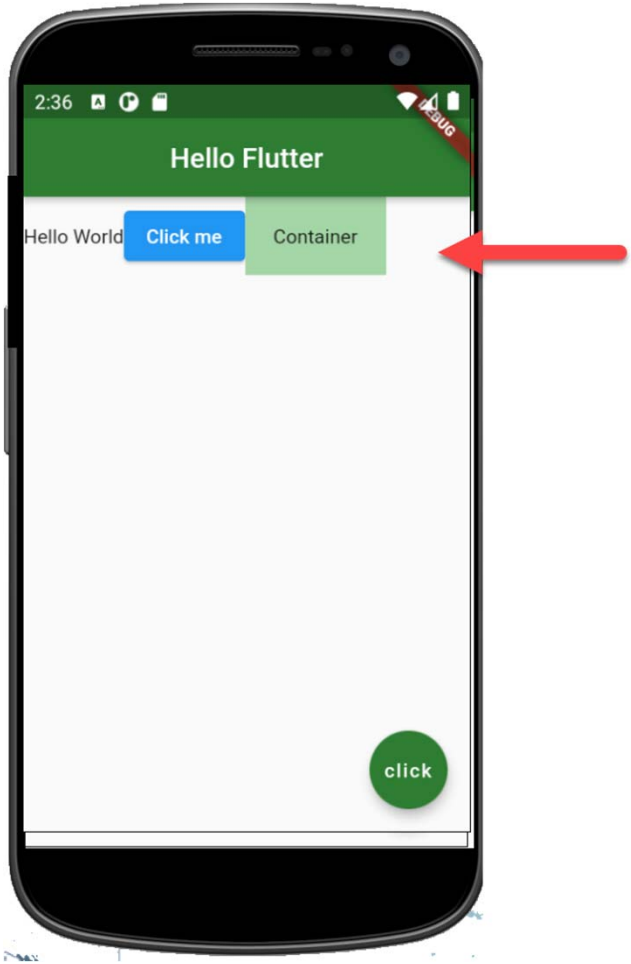
Multiple Widgets in a lay-out

- Until now we only have one `Container()` as a child. This is not very flexible.
- We can divide our lay-out in Rows and Columns.
- Both do NOT have a widget as `child`, but an array of children.
  - `children: <Widget>[…]`
  - The type annotation `<Widget>` is now optional. You may omit it
- Lets look at `Row()` first
- Children are positioned from Left to Right

Row with multiple children

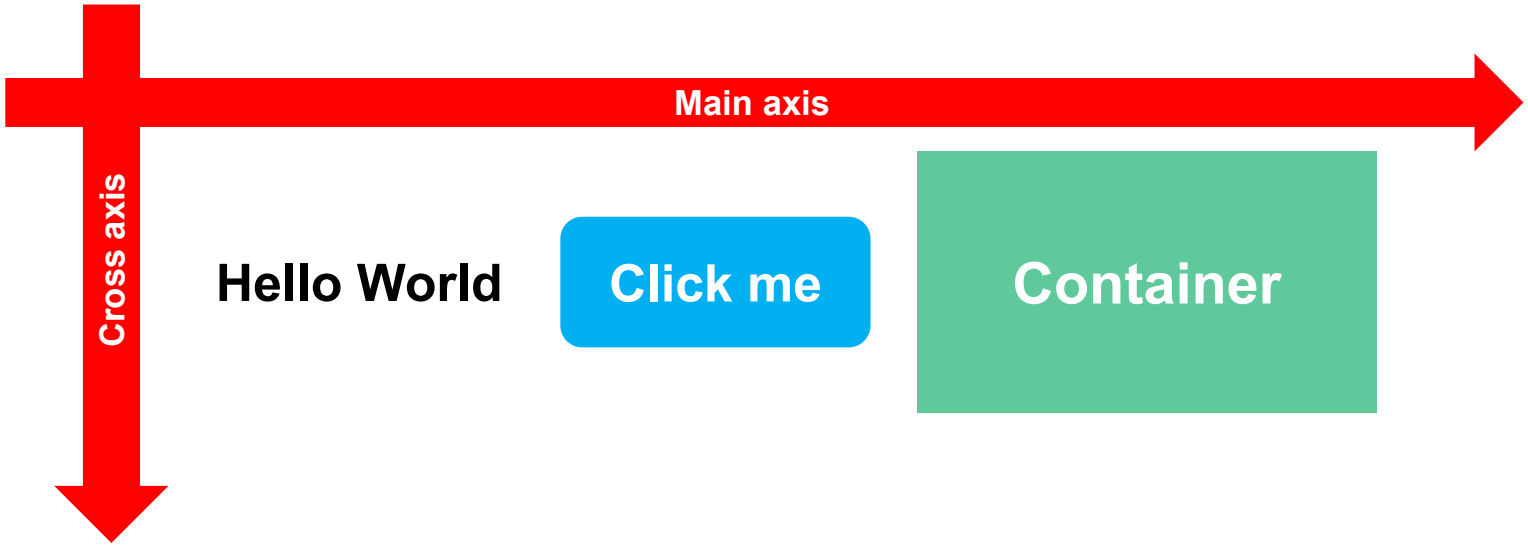
- Let's create a row with three(3) child widgets

```
body: Row(
  children: [
    Text('Hello World'),
    ElevatedButton(
        onPressed: (){},
        child: Text('Click me')
    ),
    Container(
      padding: EdgeInsets.all(20.0),
      child: Text('Container'),
      color: Colors.green[200],
    )
  ],
),
```



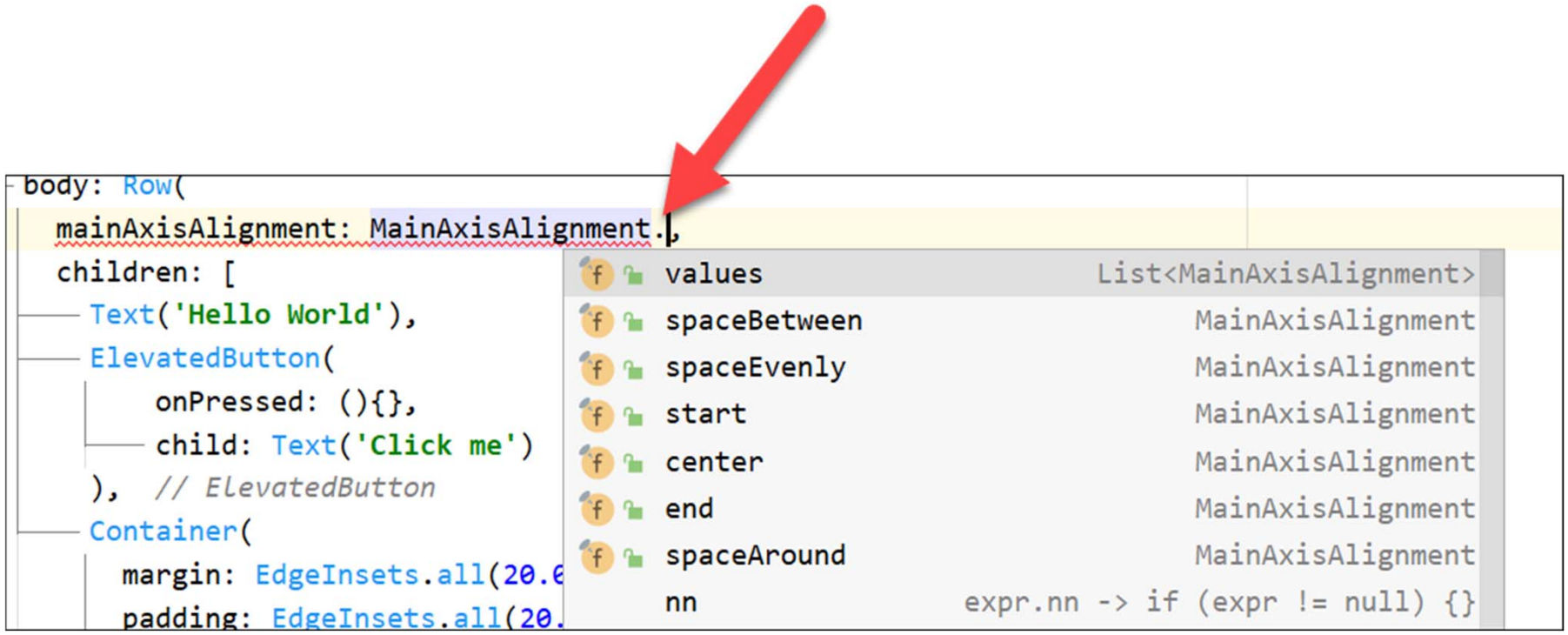
Controlling the alignment

- In Rows:
  - Use `MainAxisAlignment` for horizontal layout
  - Use `CrossAxisAlignment` for vertical layout



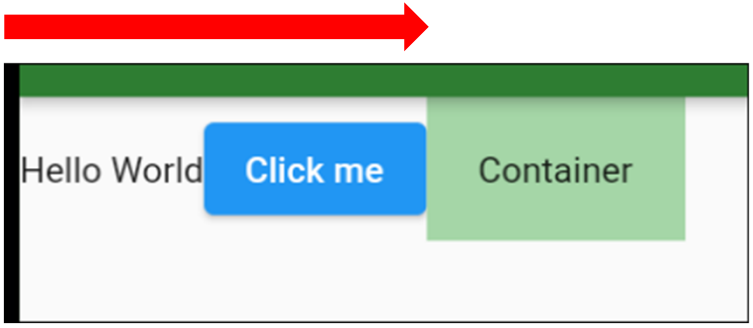
MainAxisAlignment

- Make sure to add `mainAxisAlignment` on the `Row()` itself, not on one of its children
- Options: `start`, `end`, `spaceBetween`, etc.

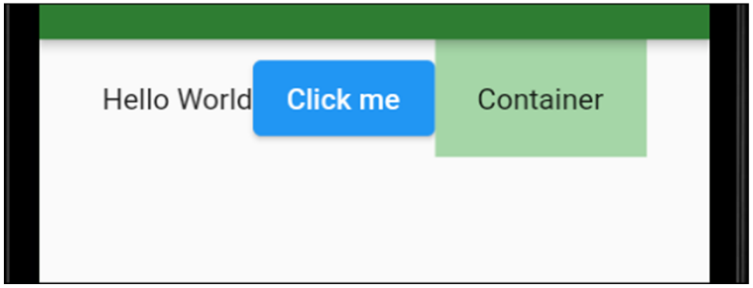


Options

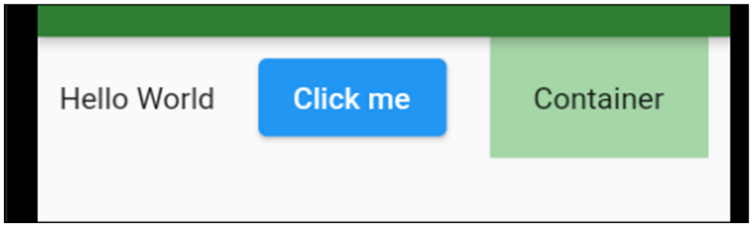
`MainAxisAlignment.start`



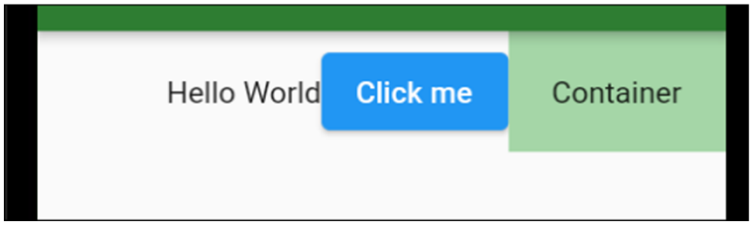
`MainAxisAlignment.center`



`MainAxisAlignment.spaceAround`

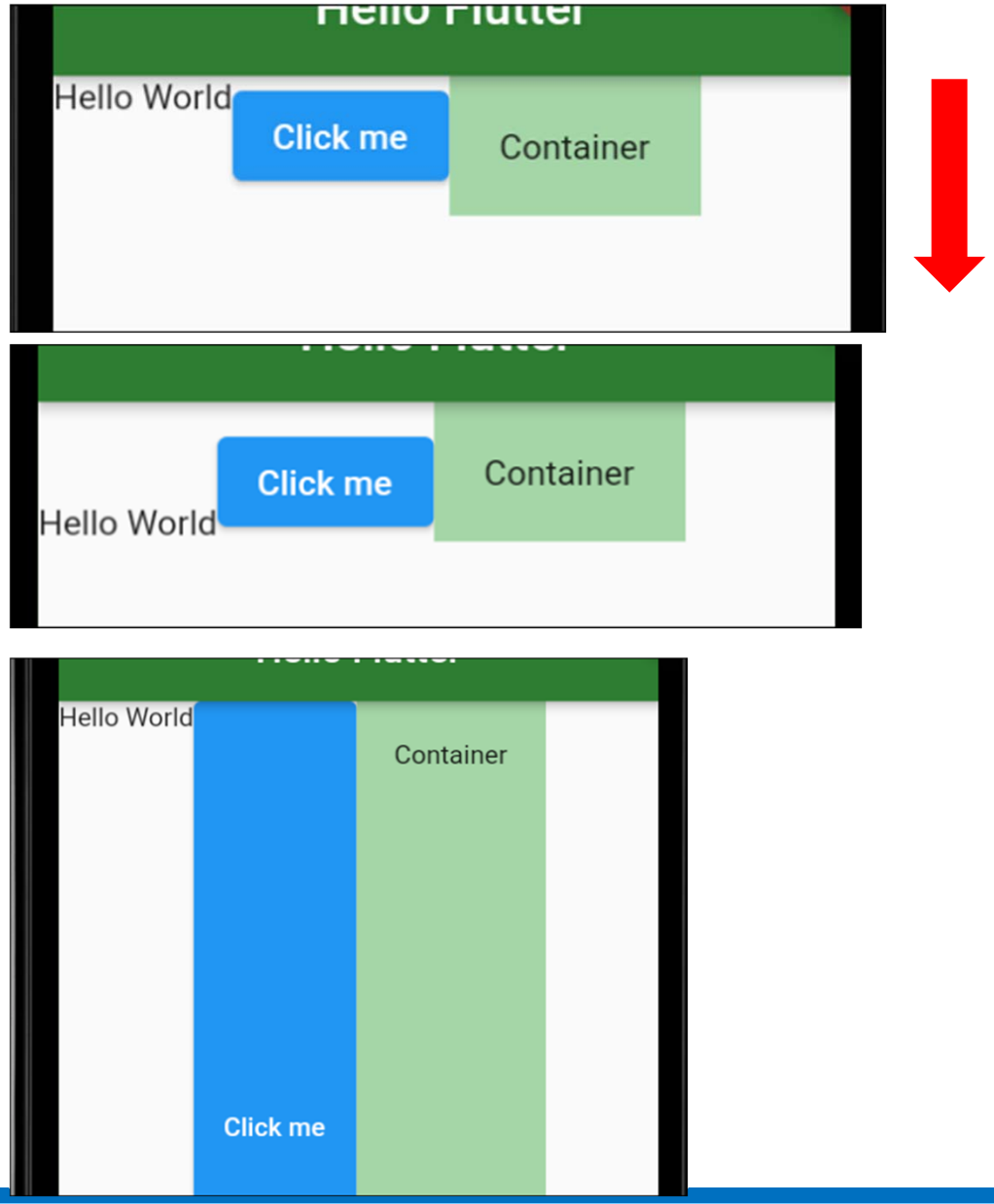


`MainAxisAlignment.end`



CrossAxisAlignment

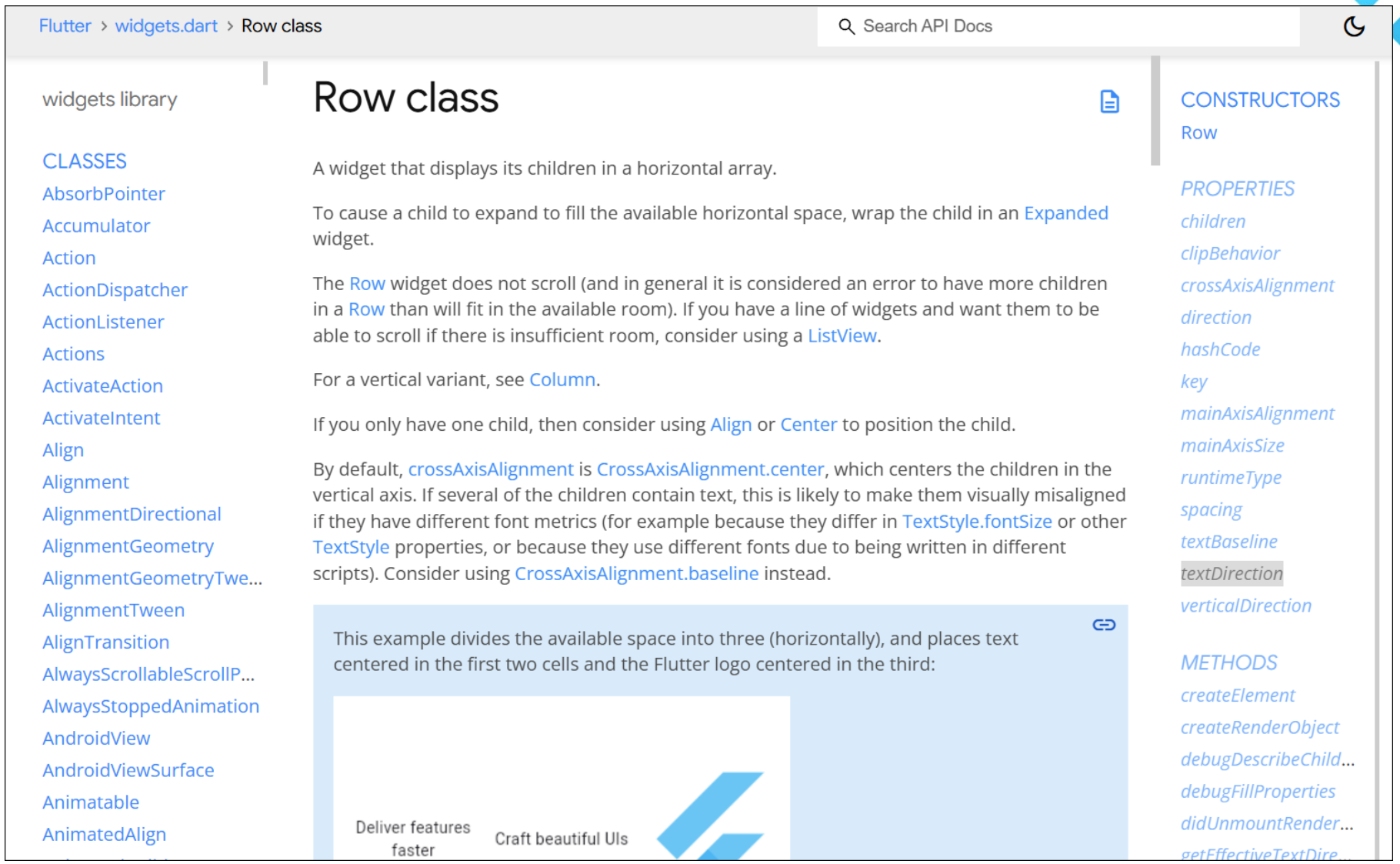
`CrossAxisAlignment.start`



`CrossAxisAlignment.end`

`CrossAxisAlignment.stretch`

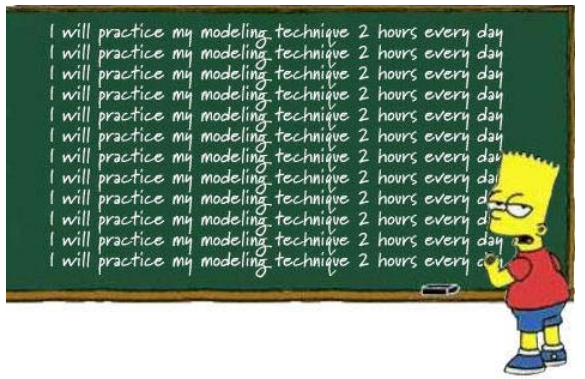
More info



https://api.flutter.dev/flutter/widgets/Row-class.html

Workshop

- Create a Row() of Buttons in your `StatelessWidget`
- Experiment with the ways to layout the buttons
  - mainAxisAlignment: `start`, `center`, `spaceEvenly`, `spaceBetween`, `end`
  - crossAxisAlignment: `start`, `end`, `stretch` (and more)
- What happens if your content is too wide to fit on the screen?
- Example: `../_170-row`
- Official docs: `api.flutter.dev/flutter/widgets/Row-class.html`



Columns

Going vertical instead of horizontal – with the `Column()` Widget

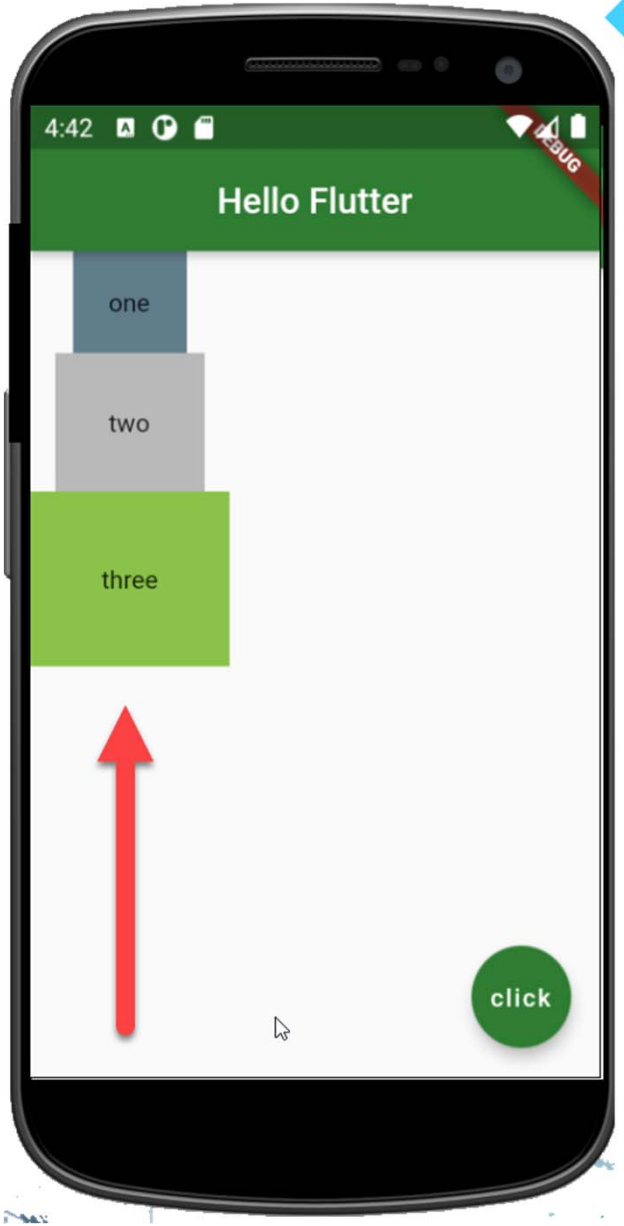
The Column() Widget

- The `Column()` Widget is used to stack child widgets on top of each other
- It also takes `children: <Widget>[]` as a property
- You can place multiple widgets in a column
- Content does not scroll automatically.
  - You'll need an additional Widget for that, for instance `Expanded()` or `ListView()`

Example

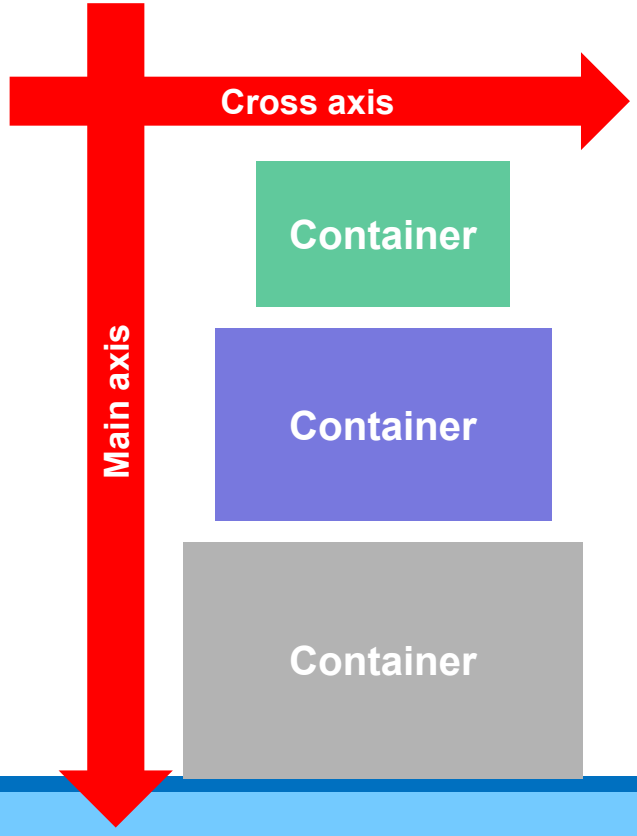
```
body: Column(
  children: <Widget>[
    Container(
      padding: EdgeInsets.all(20.0),
      color: Colors.blueGrey,
      child: Text('one'),
    ),
    Container(
      padding: EdgeInsets.all(30.0),
      color: Colors.black26,
      child: Text('two'),
    ),
    Container(
      padding: EdgeInsets.all(40.0),
      color: Colors.lightGreen,
      child: Text('three'),
    ),

  ],
),
```

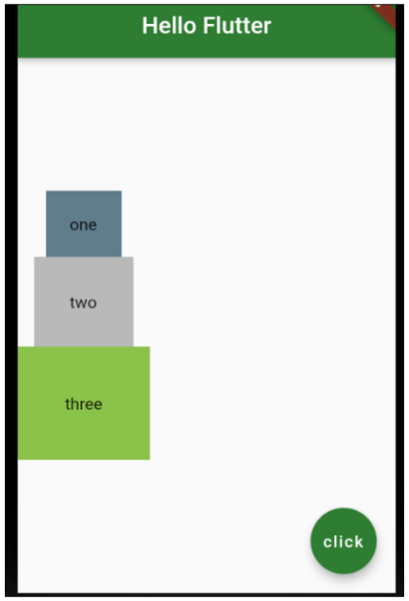


Controlling the alignment

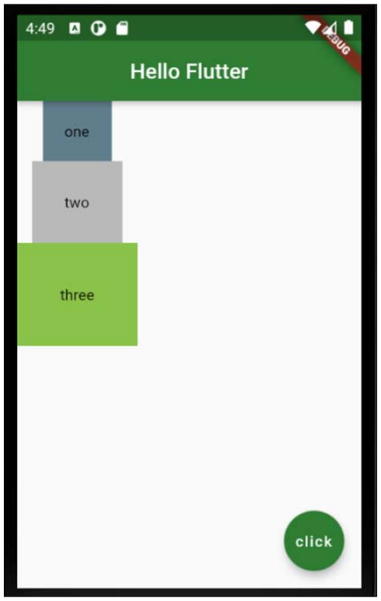
- In Columns – opposite to Rows:
  - Use `MainAxisAlignment` for vertical layout
  - Use `CrossAxisAlignment` for horizontal layout



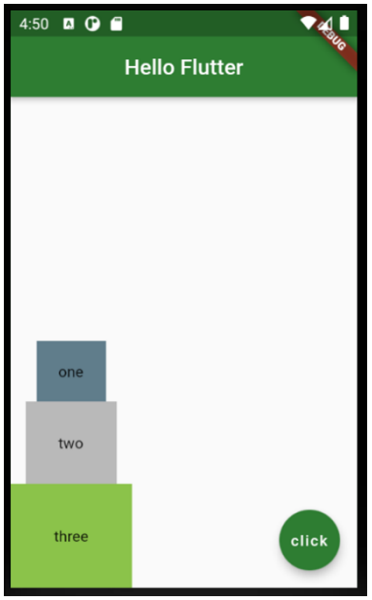
Aligning stuff - MainAxisAlignment



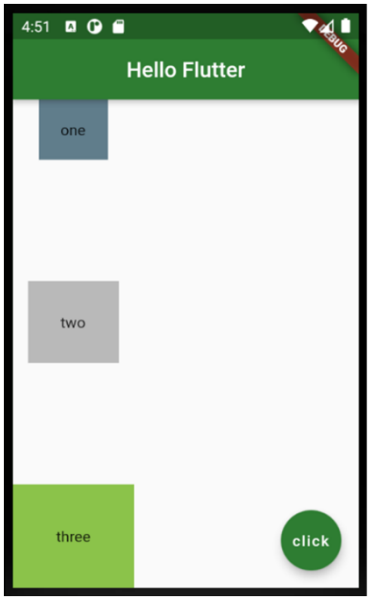
`center`



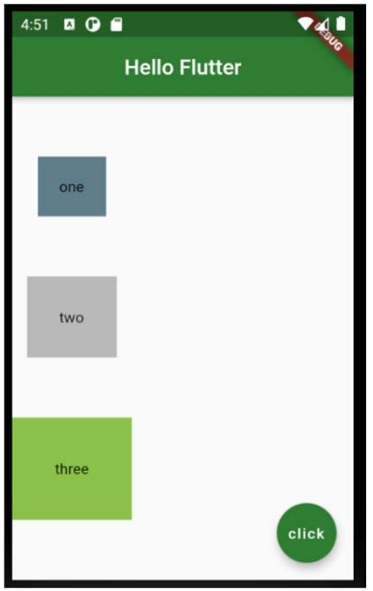
`start`



`end`



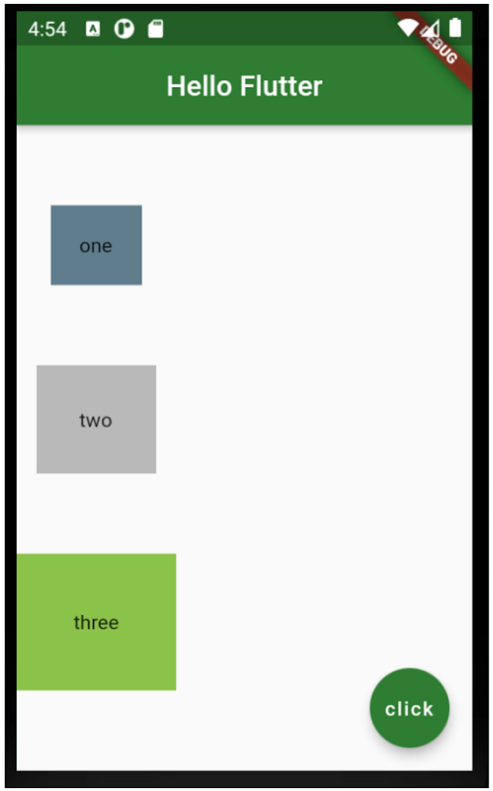
`spaceBetween`



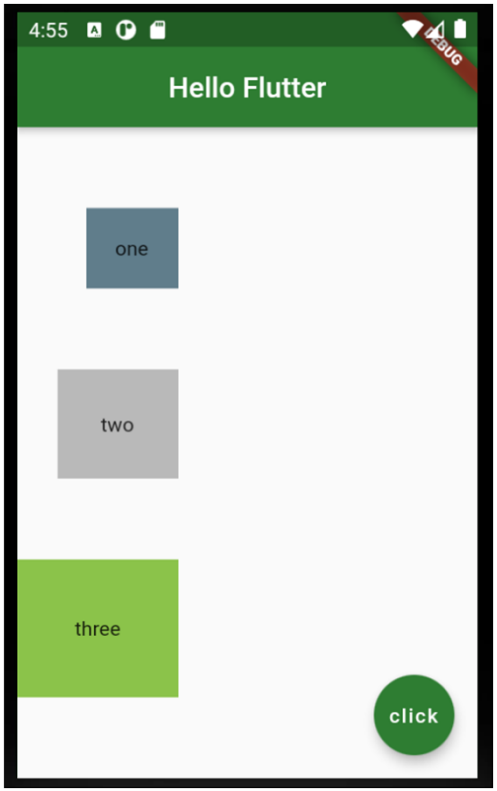
`spaceEvenly`

Look up the other options for yourself

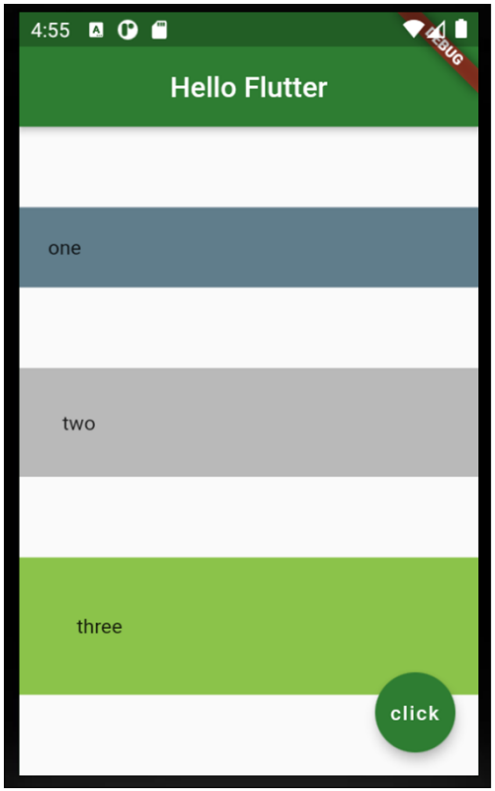
CrossAxisAlignment



`center`



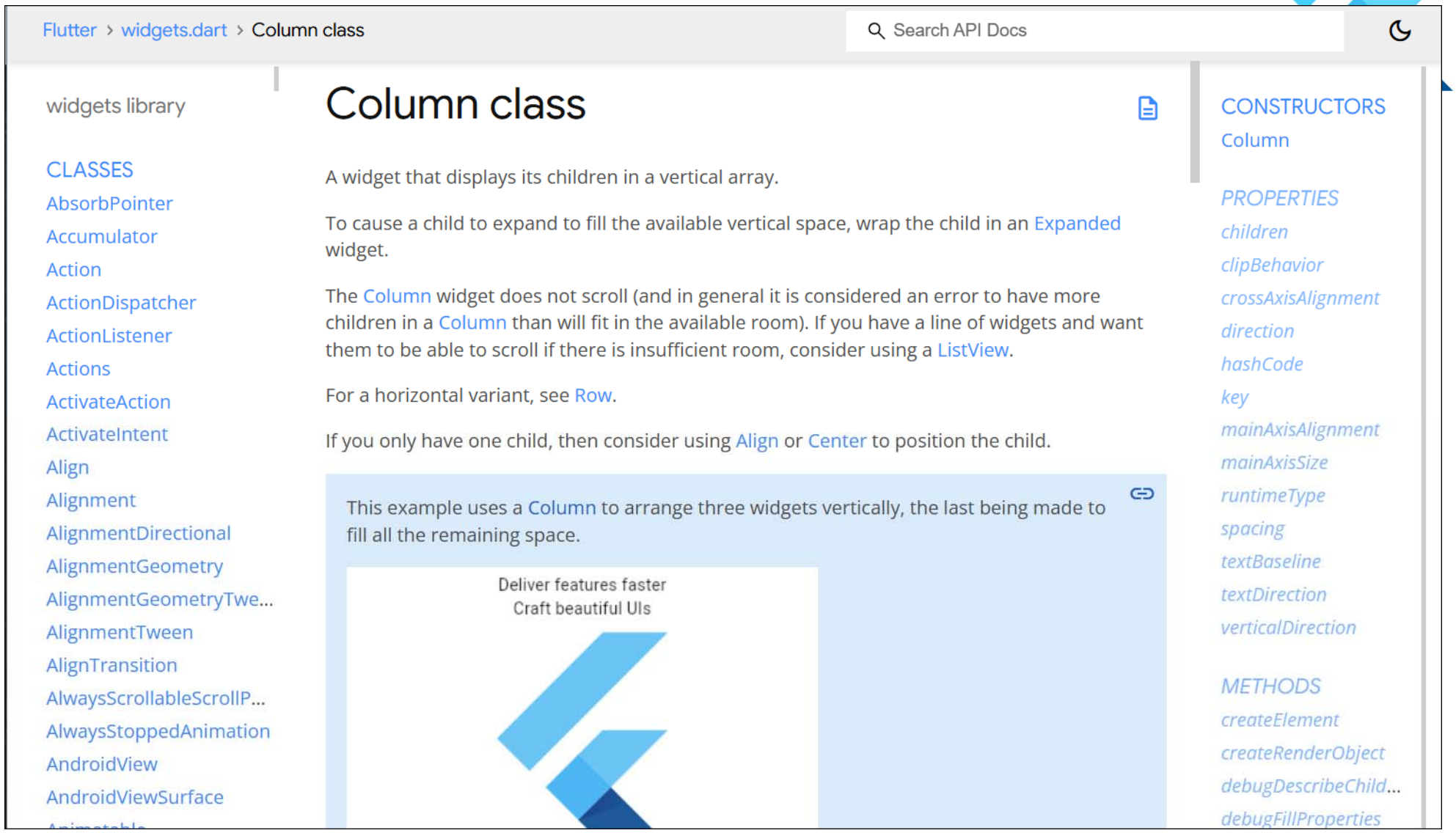
`end`



`stretch`

Look up the other options for yourself

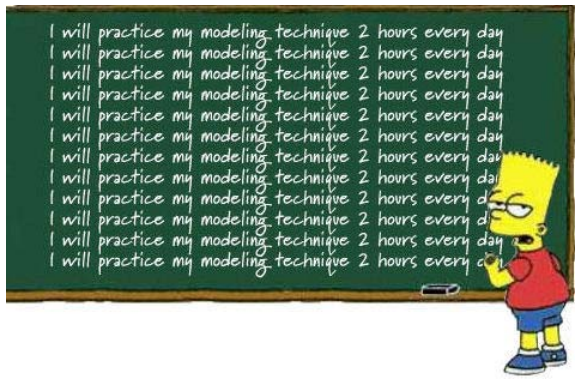
More info



api.flutter.dev/flutter/widgets/Column-class.html

Workshop

- Create a Column() of Containers in your `StatelessWidget`
- Experiment with the ways to layout the Containers
  - mainAxisAlignment: `start`, `center`, `spaceEvenly`, `spaceBetween`, `end`
  - crossAxisAlignment: `start`, `end`, `stretch` (and more)
- What happens if your content is too long to fit on the screen?
- Example: `../180-column`
- Official docs: `api.flutter.dev/flutter/widgets/Row-class.html`



Expanded() Widget

Creating flexible widgets, using available space

Expanded() Widget

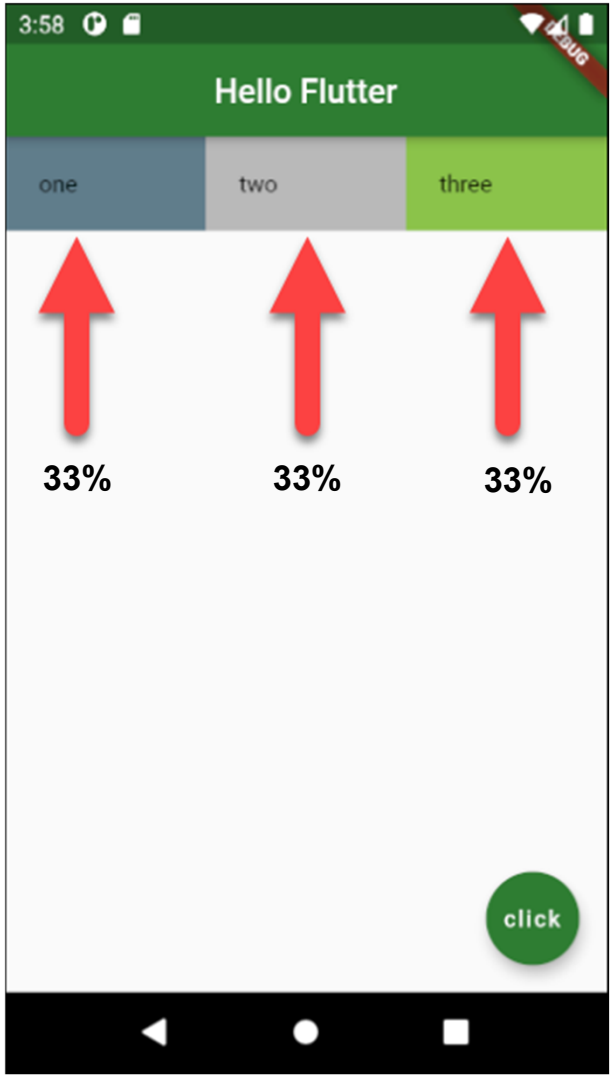
- `Expanded()` is a widget that expands the child of a `Row()`, `Column()` so that it fills the available space
- It acts like `flexbox` for CSS
- Expanding multiple children?
  - Available space is divided according to the `flex:` property

Simple row – no Expanded()

```
body: Row(
  children: <Widget>[
    Container(
      padding: EdgeInsets.all(20.0),
      color: Colors.blueGrey,
      child: Text('one'),
    ),
    Container(
      padding: EdgeInsets.all(20.0),
      color: Colors.black26,
      child: Text('two'),
    ),
    Container(
      padding: EdgeInsets.all(20.0),
      color: Colors.lightGreen,
      child: Text('three'),
    ),
  ],
),
```

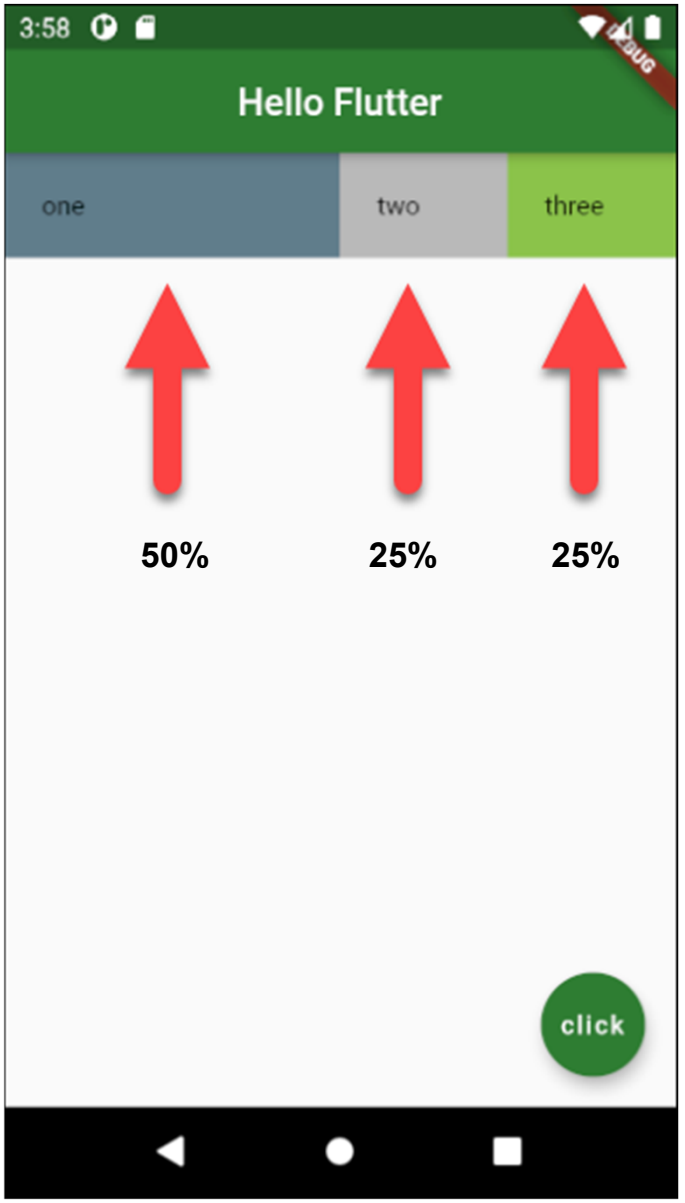
Using Expanded()

```
body: Row(
  children: <Widget>[
    Expanded(
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.blueGrey,
        child: Text('one'),
      ),
    ),
    Expanded(
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.black26,
        child: Text('two'),
      ),
    ),
    Expanded(
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.lightGreen,
        child: Text('three'),
      ),
    ),
  ],
),
```



Using Expanded() with a flex: factor

```
body: Row(
  children: <Widget>[
    Expanded(
      flex: 4,
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.blueGrey,
        child: Text('one'),
      ),
    ),
    Expanded(
      flex: 2,
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.black26,
        child: Text('two'),
      ),
    ),
    Expanded(
      flex: 2,
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.lightGreen,
        child: Text('three'),
      ),
    ),
  ],
),
```



`flex:` fractions

- The `flex:` property is a number – it represents the portion of the width that we want to associate with the child, compared to the total number
- So with (for example) flex properties of 6–3–1
  - The first child gets `6/10` of the available space
  - The second child gets `3/10` of the available space
  - The third child gets `1/10` of the available space
- Expanded( ) works with rows *and* columns.

Practical application

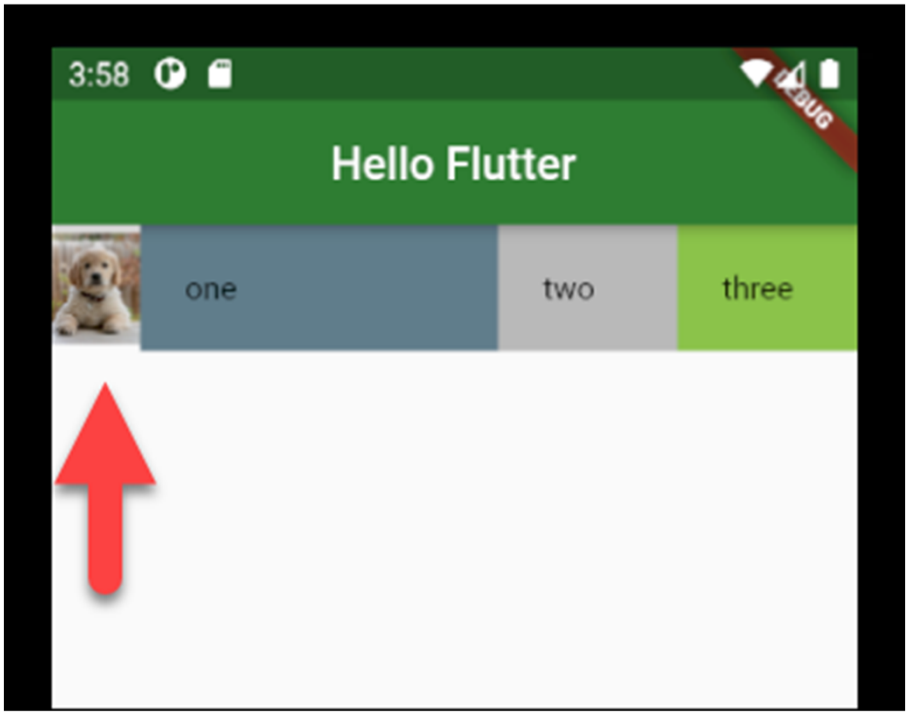
- Wrap an Image( ) with an Expanded( ) to make sure the image is not to wide/high

```
body: Row(
  children: <Widget>[
    Image.asset('assets/puppy-1.jpg'),
    Expanded(
      flex: 4,
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.blueGrey,
        child: Text('one'),
      ),
    ),
 …
)
```

Image as a child in a Row( ) - without Expanded( )

Image wrapped in Expanded()

```
body: Row(
  children: <Widget>[
    Expanded(
        child: Image.asset('assets/puppy-1.jpg')
    ),
    Expanded(
      flex: 4,
      child: Container(
        padding: EdgeInsets.all(20.0),
        color: Colors.blueGrey,
        child: Text('one'),
      ),
    ),
…
```



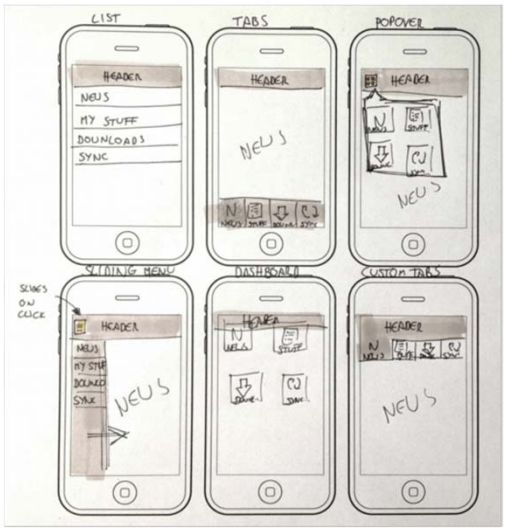
Show it above the containers?

- Wrap the `Image` in its own `Row()`, inside a `Column()` like so:

```
body: Column(
  children: [
    Row(children: <Widget>[
      Expanded(
        child: Image.asset('assets/puppy-1.jpg’)
      ),
    ]),
    Row(
      children: <Widget>[
        Expanded(
          flex: 4,
          child: Container(
            padding: EdgeInsets.all(20.0),
            color: Colors.blueGrey,
            child: Text('one'),
          ),
        ),
        …
),
```

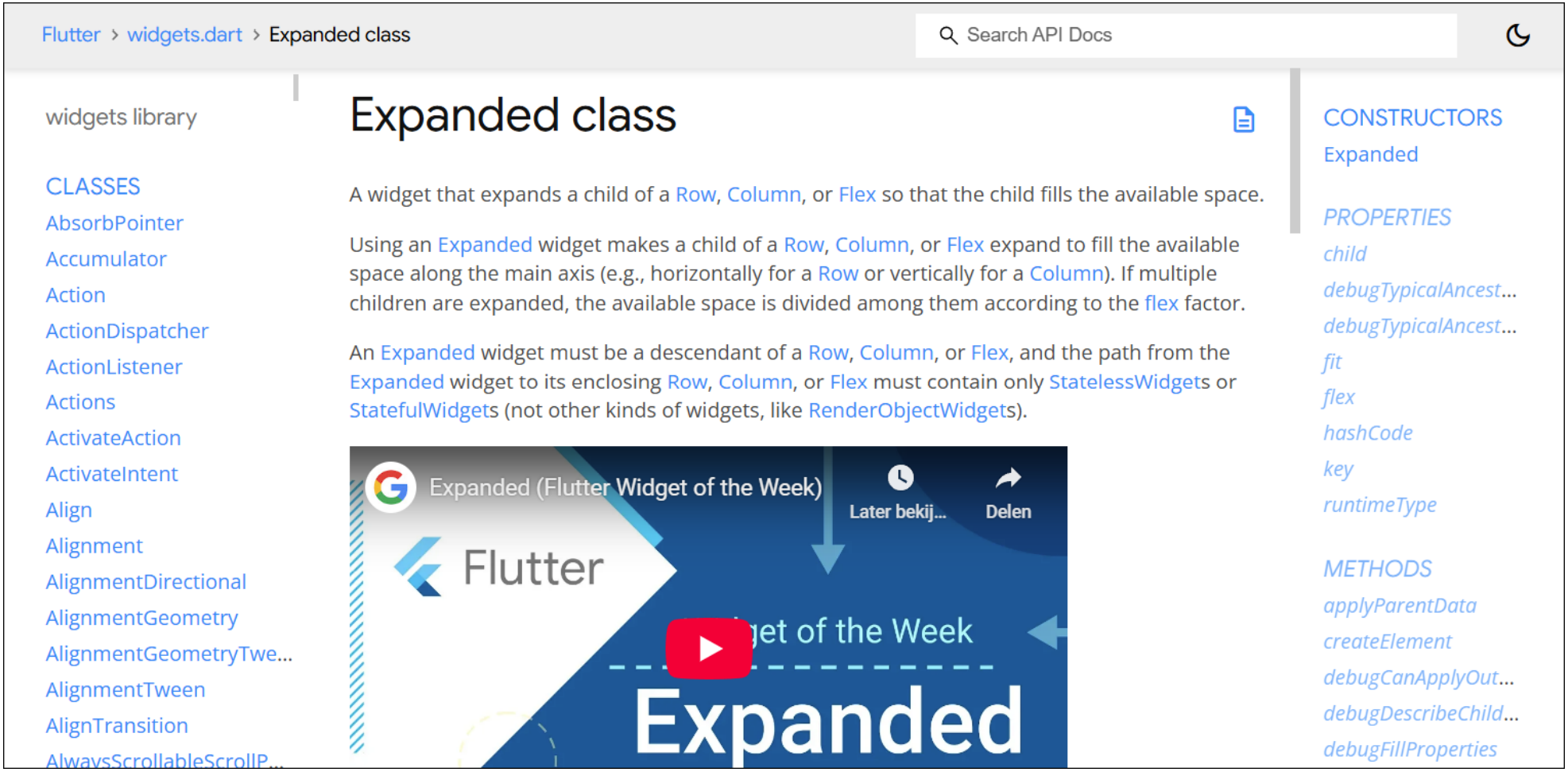
Creating Layouts

- Mobile layouts often start with a single Column
  - Desktop layouts – depends on the requirements!
- Inside that column, place rows, or widgets directly
- Sketch out a layout with pencil and paper!



https://www.smashingmagazine.com/2013/06/sketching-for-better-mobile-experiences/

More info



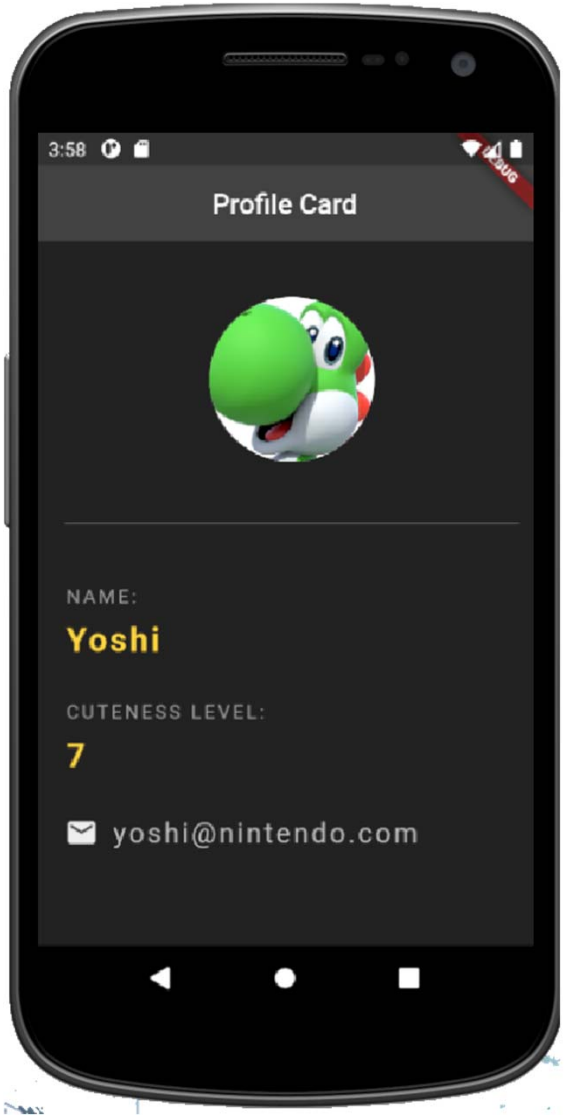
https://api.flutter.dev/flutter/widgets/Expanded-class.html

Sample Layouts

Workshop – create these layouts yourself!

3 sample layouts – each a bit more difficult than the previous one

Sample layouts - `ProfileCard.dart`



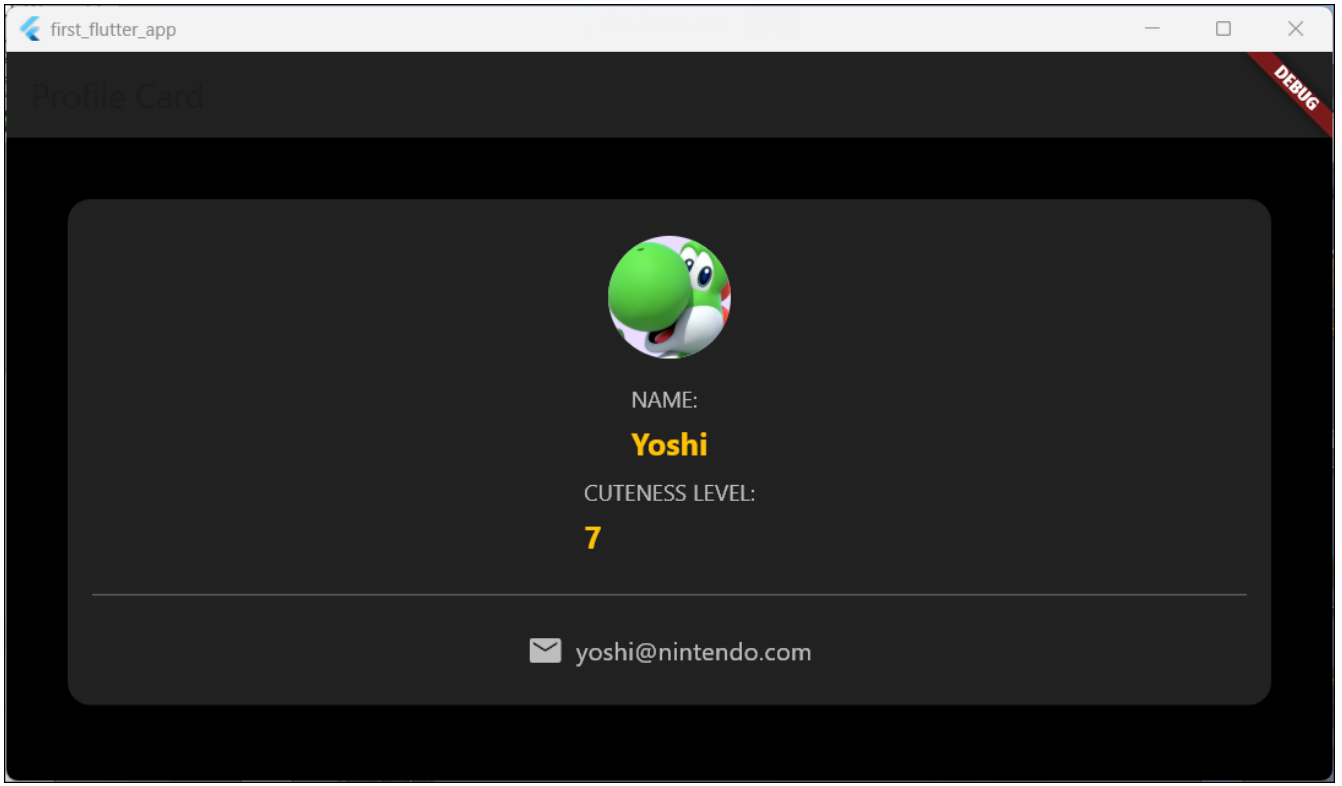
Extra widgets, used in this layout

(Look them up yourself):

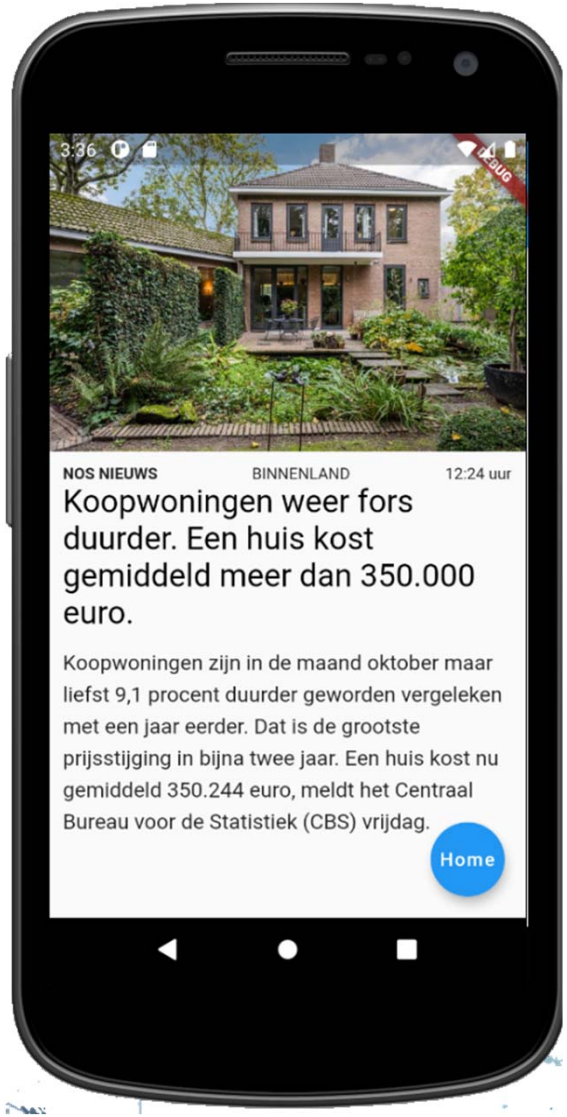
- `Card()` – To wrap content in a Card with slightly rounded corners and elevation shadow
- `SizedBox()` – to give widgets some room
- `CircleAvatar()` – to display the avatar image
- `Divider()` – to create the horizonal line and some room
- All texts are hard coded!

ProfileCard.dart

- We also created a function to return the text rows
  - `Widget buildTextRow(…){ return Column(…) …}`
- On desktop:



Sample layouts - `NewsItem.dart`



This is a single column.

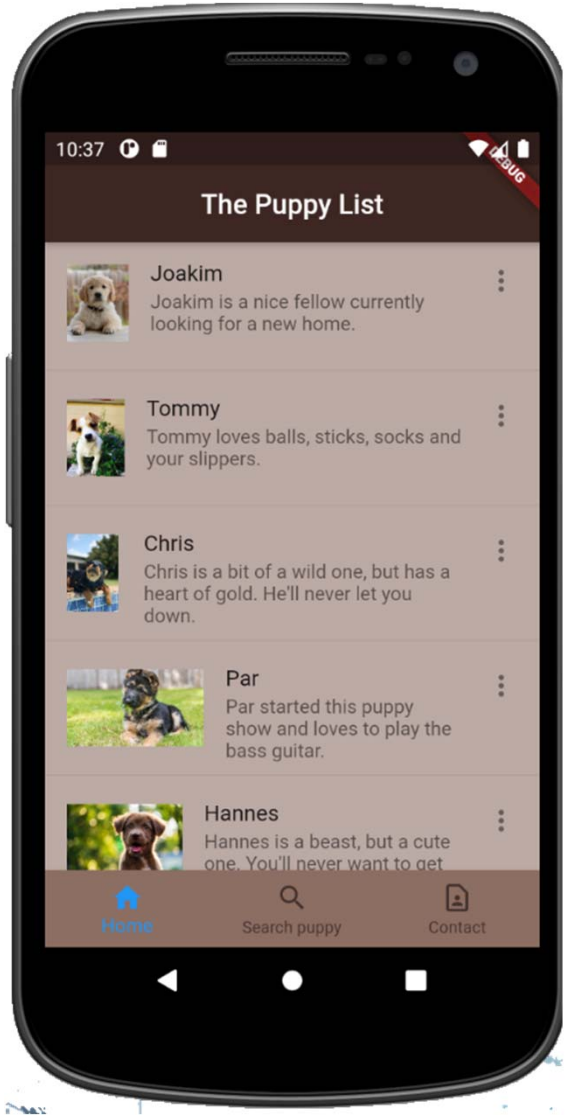
The content below the image is wrapped in a `Padding()` widget to give the content some room to the edges of the screen. Inside `Padding()` we have another `Column()` with content.

But maybe you'll solve this differently?

NewsItem.dart – desktop layout

- Never mind the two different house pictures.
- This is `assets/house_funda.jpg`
- The picture is edge-to-edge, all text below has a padding to make it stand out from the edges.

Sample layouts – PuppyList.dart



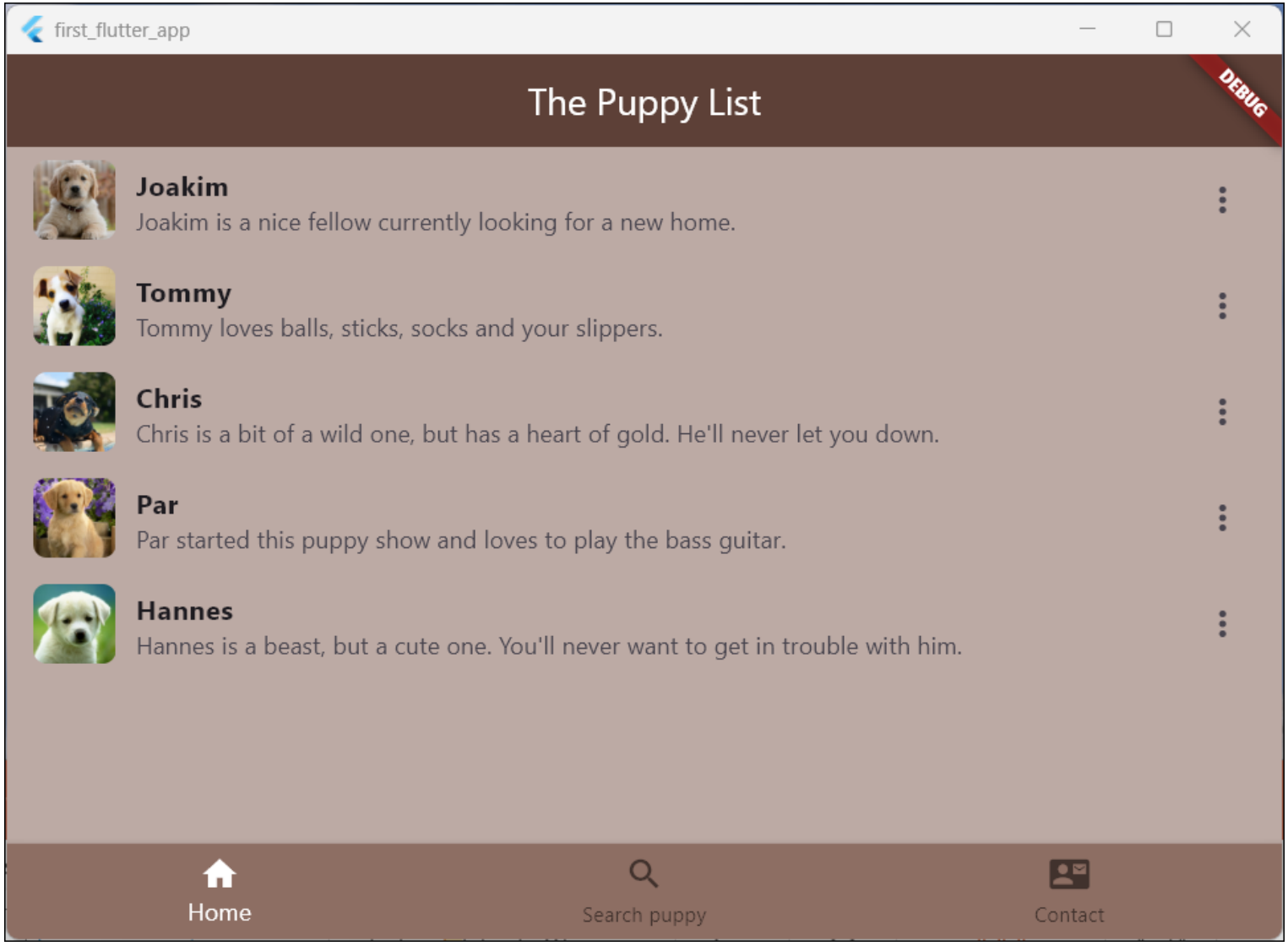
This is a scrollable list of (hardcoded!) items.

It is a little harder to recreate. It uses an array of Puppy-items that are used.

Extra widgets, used in this layout . We haven't covered all this! (Look it up yourself):

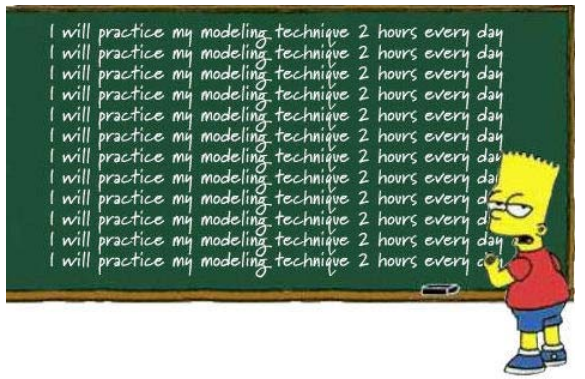
- `ListView.builder()` – to wrap the individual items
- `ListTile()` – to create items in the list
- `Divider()` – to create the horizonal line and some room between items
- `Icons.more_vert` – to create the three dots

`PuppyList.dart` – Desktop view



Workshop – Home work assignment

- (Re)create these previous 3 layouts
- Experiment with the ways to layout the Containers, Rows and Columns
- Create at a minimum 1 lay-out yourself
  - For your work, hobby, family, etc.
- Demonstrate it next day, or to a colleague



Checkpoint

- You know about the Container widget
- You can divide the lay-out of the app in Rows and Columns and know the widgets for this
- You are aware of the different axis alignments for Rows and Columns
- You know the Expanded widget to take up available space
- You can create different lay-outs by combining all previous knowledge