Rows, Columns & Icons:

Now we’re talking, aren’t we?

Now is the time to add multiple widgets onto the screen at once. Because, up until now, our apps looked somewhat mundane to the eye. We’ll start with rows first:

A row means a horizontal arrangement of objects. Anything that we put into a row will be placed one beside the other. We need to add a widget called ‘Row’ to the body property. This widget has a property called ‘*children’* (Notice that the word children implies that it accepts more than one child—which is kind of the point of rows). This property accepts a *list of widgets.* As discussed in the DART primer, a list is nothing but an array in DART. It’s datatype is the objects of the Widget class.

Let’s add 3 widgets to the row:

1. A simple text widget
2. A flat button widget (we’ll learn about it in a moment, for now, just roll with it)
3. A container widget containing text

We’ll colour them differently in order to identify them. The code looks like:

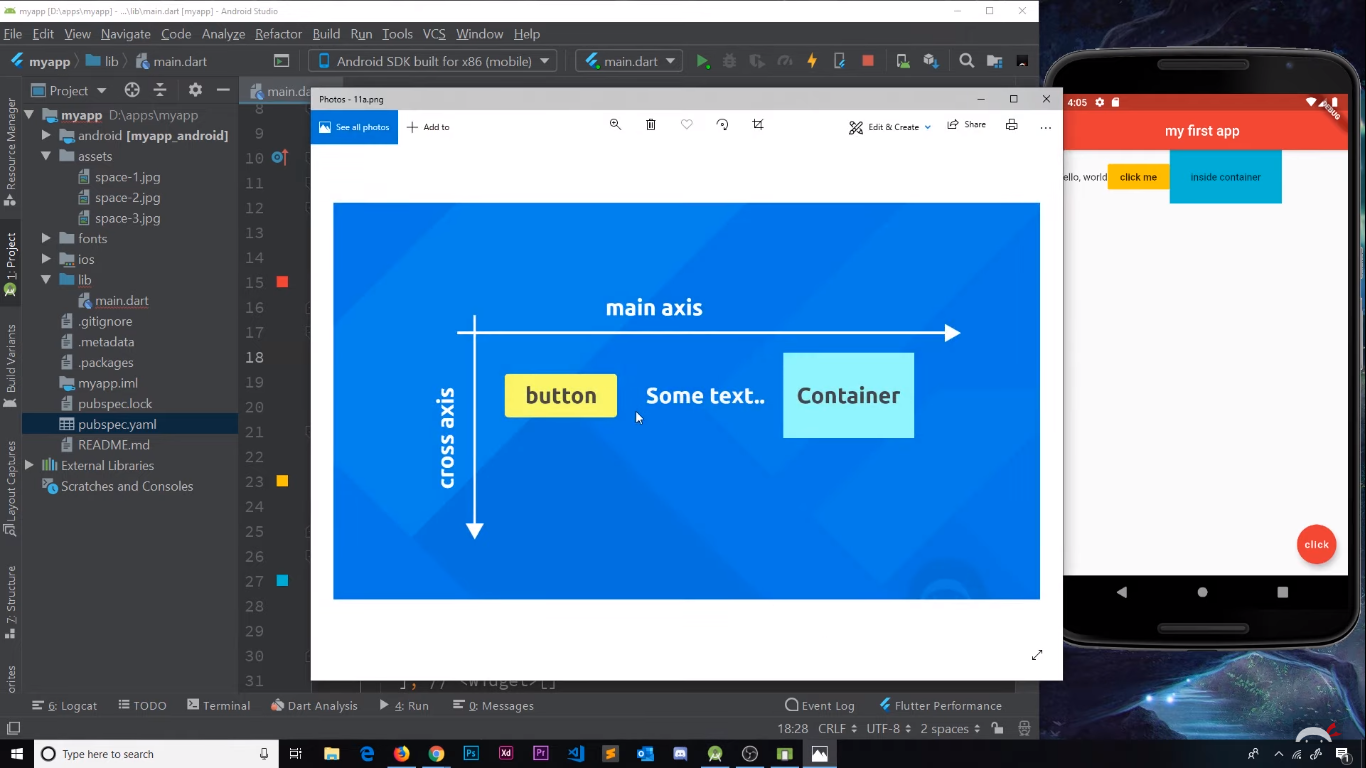
body: Row(  
 children: <Widget>[  
 Text(**'Your'**),  
 FlatButton(  
 onPressed: () {},  
 child: Text(**'Mom'**),  
 color: Colors.*orange*,  
 )  
 Container(  
 child: Text(**'Gay'**),  
 color: Colors.*cyan*,  
 padding: EdgeInsets.all(20.0),  
 )  
 ],  
),



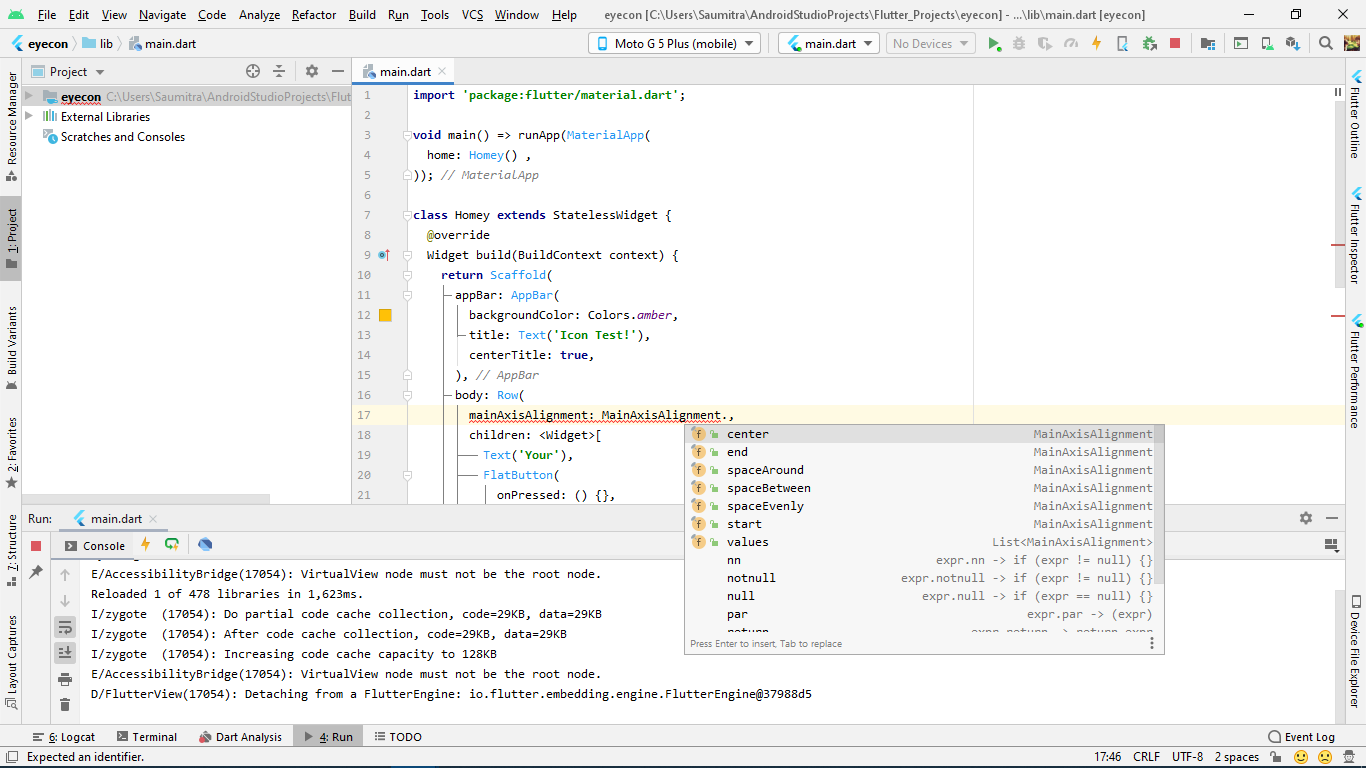
Et Voila! We now have multiple widgets on the screen!

These widgets however, look kind of bunched up or stacked together. In order to change the alignment of these widgets, we’ll use properties of the row widget called ‘MainAxisAlignment’ and ‘CrossAxisAlignment’.

In the case of rows, the horizontal axis is the main axis, and the axis that crosses them is the cross axis.

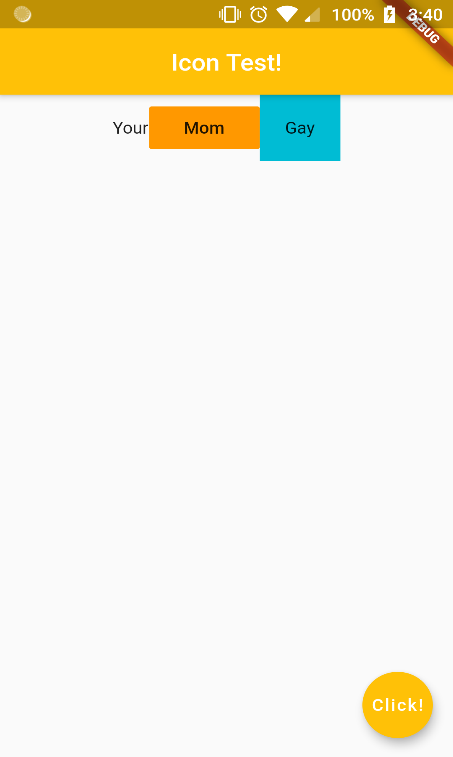


Let’s start with mainAxisAlignment. It has a value of MainAxisAlignment.<various options>.



1. center : Positions the elements at the centre of the row.

mainAxisAlignment: MainAxisAlignment.**center**



1. end : Bunches the elements to the right hand side of the screen (The “end” of the screen).

mainAxisAlignment: MainAxisAlignment.**end**



1. spaceAround : Similar to spaceBetween, the difference is that the space between the widgets is double that between the widget and the end of the screen. Essentially each widget has like a margin around it.

mainAxisAlignment: MainAxisAlignment.**spaceAround**



1. spaceBetween : Places elements with spaces in between, but no spaces at the ends. Start and end elements are stuck to the LHS and RHS of the screen.

mainAxisAlignment: MainAxisAlignment.**spaceBetween**,



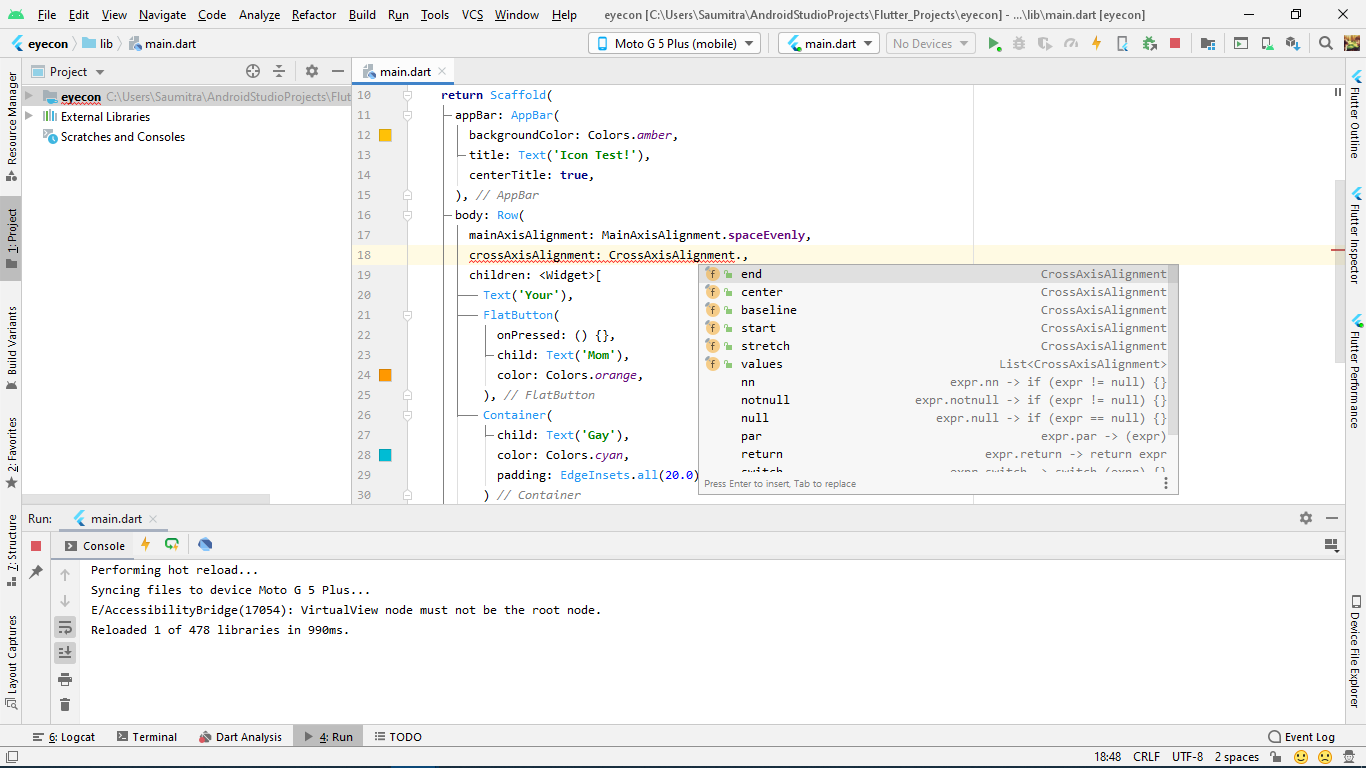
1. spaceEvenly : Spaces elements evenly across the entire row. Elements are also separated from the ends of the screen.

mainAxisAlignment: MainAxisAlignment.**spaceEvenly**

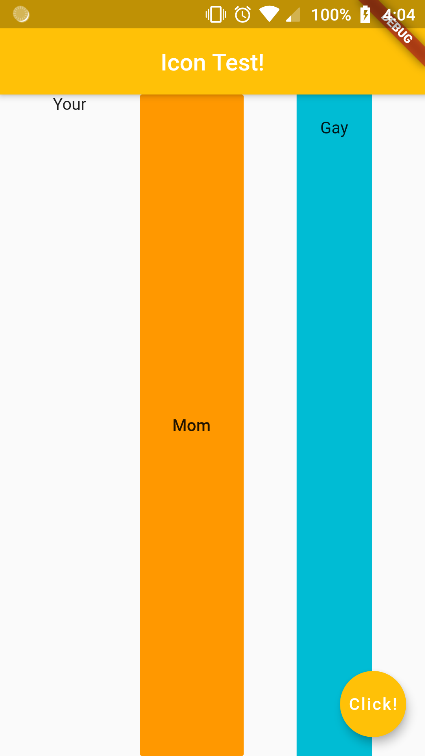


1. start: This the default setting. Elements are positioned from the start of the screen (LHS)

In a similar manner, we have the crossAxisAlignment property, with multiple options. This property adjusts how the elements look with respect to the vertical axis of the row.



1. Stretch: Stretches elements across the entire cross axis of the row. Because there is only one row, the cross axis is the entire screen height.



1. Center: The default setting, at the centre of the height of the tallest widget.
2. Start: Positions elements at the start of the cross axis



1. End: Positions elements at the bottom of the tallest widget

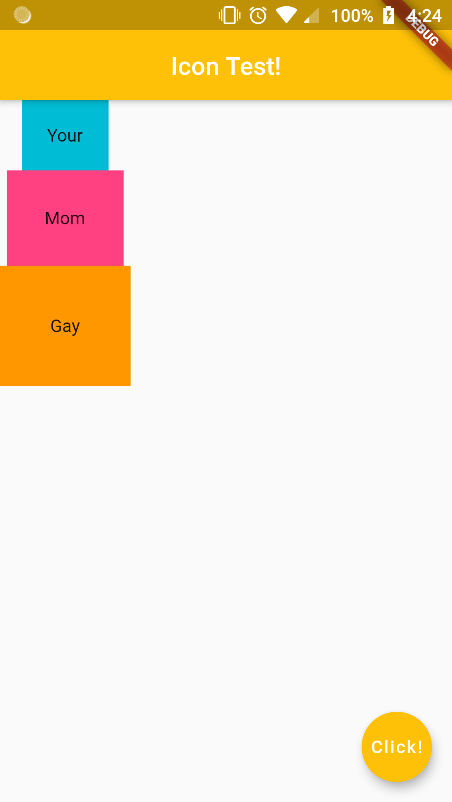


The entire code looks like:

body: Row(  
 mainAxisAlignment: MainAxisAlignment.**spaceEvenly**,  
 crossAxisAlignment: CrossAxisAlignment.**end**,  
 children: <Widget>[  
 Text(**'Your'**),  
 FlatButton(  
 onPressed: () {},  
 child: Text(**'Mom'**),  
 color: Colors.*orange*,  
 ),  
 Container(  
 child: Text(**'Gay'**),  
 color: Colors.*cyan*,  
 padding: EdgeInsets.all(20.0),  
 )  
 ],  
),

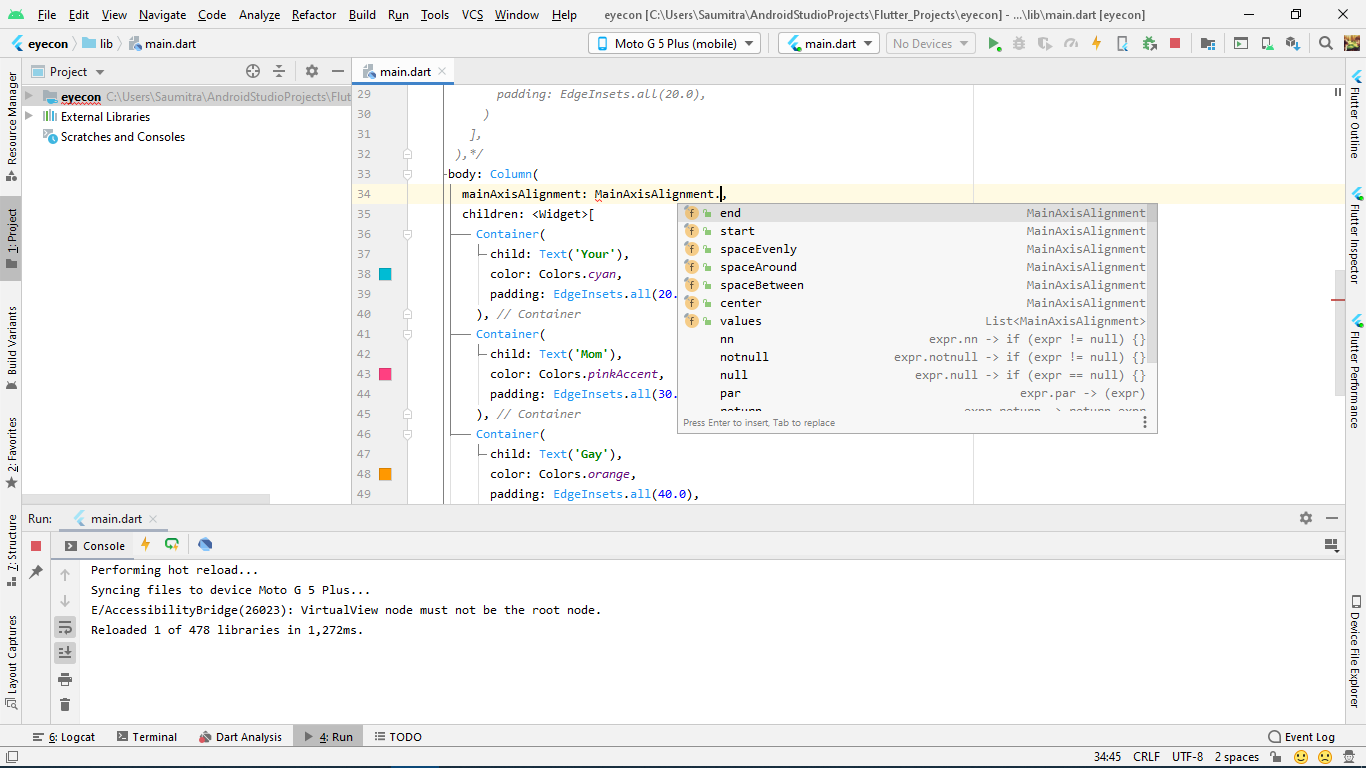
Moving on to columns, we have the exact similar format of children as that of rows. The code now looks like:

body: Column(  
 children: <Widget>[  
 Container(  
 child: Text(**'Your'**),  
 color: Colors.*cyan*,  
 padding: EdgeInsets.all(20.0),  
 ),  
 Container(  
 child: Text(**'Mom'**),  
 color: Colors.*pinkAccent*,  
 padding: EdgeInsets.all(30.0),  
 ),  
 Container(  
 child: Text(**'Gay'**),  
 color: Colors.*orange*,  
 padding: EdgeInsets.all(40.0),  
 )  
 ],  
),

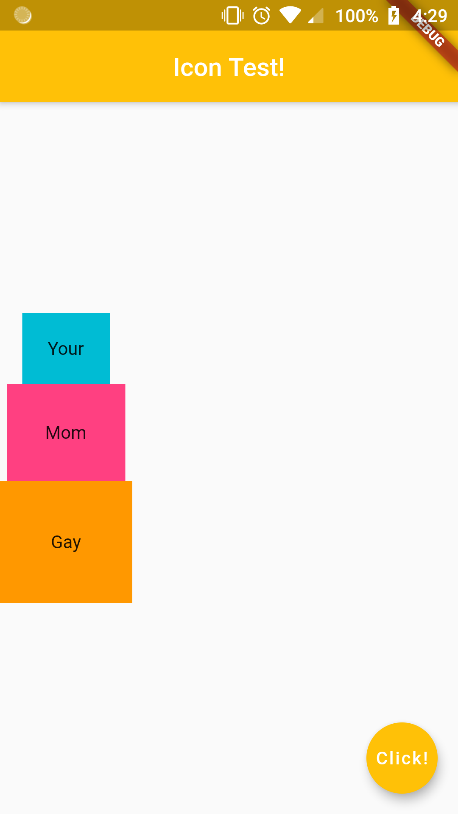


For the alignments, we need to understand that the ‘MainAxis’ for the column widget is VERTICAL, and the ‘CrossAxis’ is HORIZONTAL.

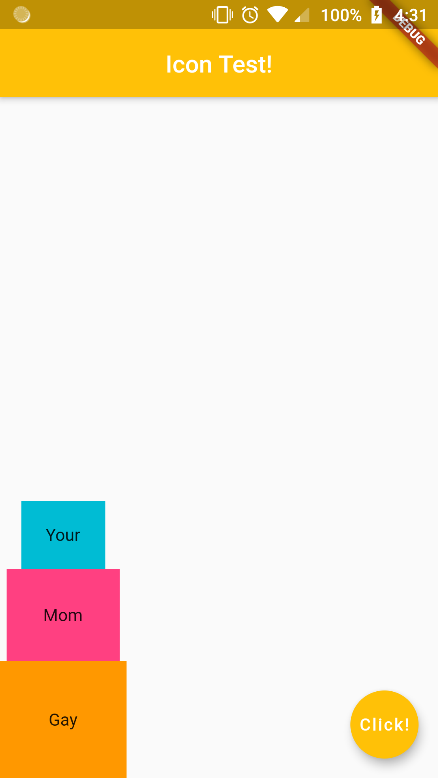
Let’s start with mainAxisAlignment. It has a value of MainAxisAlignment.<various options>.



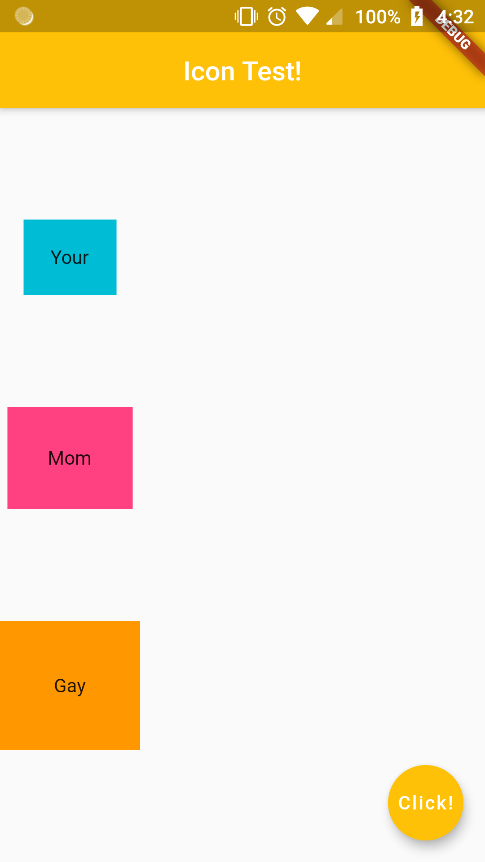
1. Center: Positions the elements at the centre of the column.



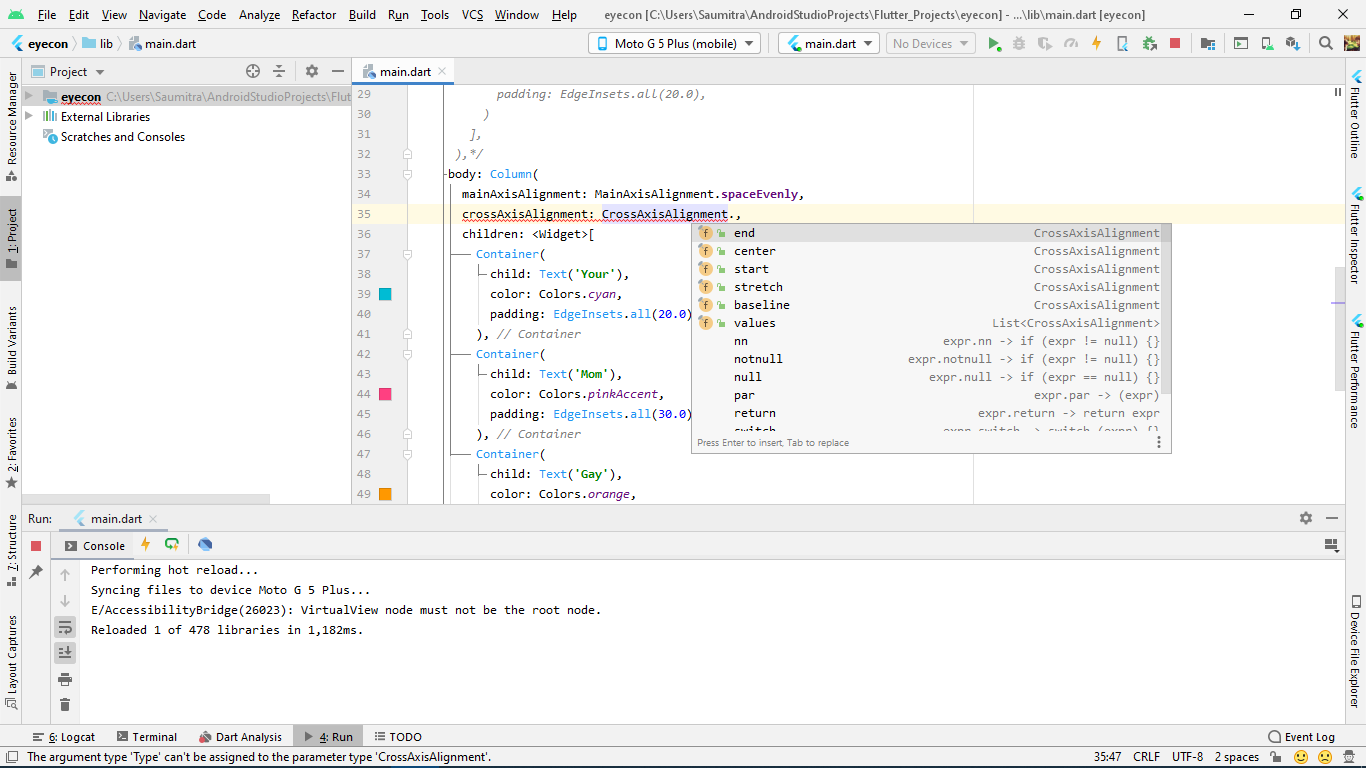
1. End: Bunches the elements to the bottom of the main axis (The “end” of the screen).



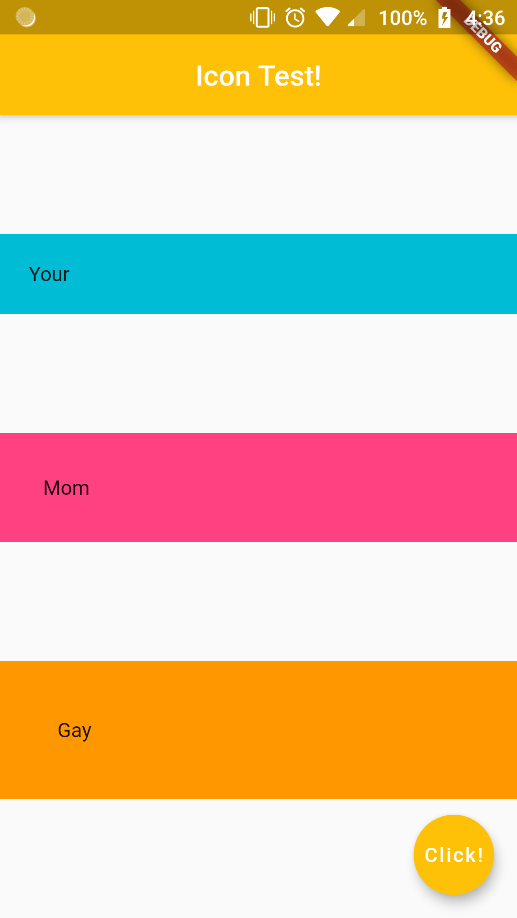
1. spaceEvenly : Spaces elements evenly across the entire column. Elements are also separated from the ends of the screen.



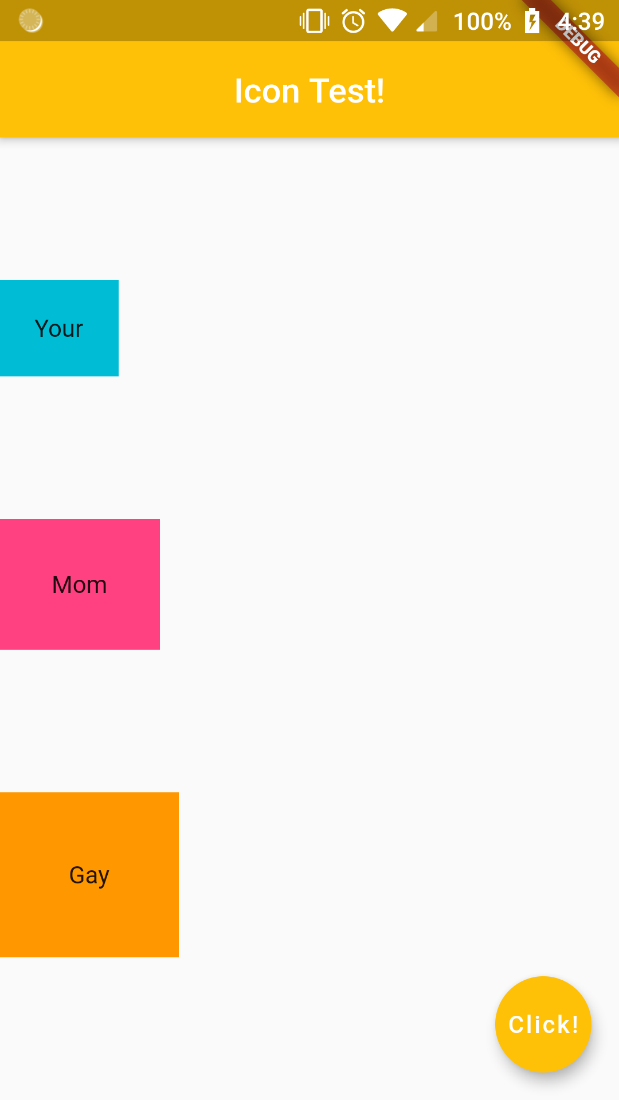
Now, let’s try the cross axis alignment…



1. Stretch: Stretches elements across the entire cross axis of the row. Because there is only one row, the cross axis is the entire screen width.



1. Center: The default setting, at the centre of the width of the widest widget.
2. Start: Positions elements at the start of the cross axis



1. End: Positions elements at the bottom of the widest widget.

\*Here we have also changed the mainAxisAlignment

mainAxisAlignment: MainAxisAlignment.**center**,  
crossAxisAlignment: CrossAxisAlignment.**end**

