

**SEB-312 Mobile Application Development**

**LAB # 03**

**LAB Title**

|  |
| --- |
| Experimenting with Stateless and Stateful widgets, Building a basic UI with Text, Text field, Button and Container widgets. Experimenting with Stateless and Stateful widgets, Building a basic UI with Text, Text field, Button and Container widgets |

**Assessment of CLO: 03, PLO: 05**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** |  | | |
| **Roll No.** |  | | |
| **Semester** |  | **Session** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Perf. Level**  **Criteria** | **Excellent**  **(2.5)** | **Good**  **(2)** | **Satisfactory**  **(1.5)** | **Needs Improvement**  **(0 ~ 1)** | **Marks Obtained** |
| **1** | Project Execution & Implementation | Fully functional, optimized, and well-structured. | Minor errors, mostly functional. | Some errors, requires guidance. | Major errors, non-functional, or not Performed. |  |
| **2** | Results & Debugging  Or Troubleshooting | Accurate results with effective debugging  Or Troubleshooting. | Mostly correct, some debugging Or Troubleshooting needed. | Partial results, minimal debugging  Or Troubleshooting. | Incorrect results, no debugging Or Troubleshooting, or not attempted. |  |
| **3** | Problem-Solving & Adaptability  (VIVA) | Creative approach, efficiently solves challenges. | Adapts well, minor struggles. | Some adaptability, needs guidance. | Lacks innovation or no innovation, unable to solve problems. |  |
| **4** | Report Quality & Documentation | Clear, structured, with detailed visuals. | Mostly clear, minor gaps. | Some clarity issues, missing details. | Poorly structured, lacks clarity, or not submitted. |  |
| **Total Marks Obtained Out of 10** | | | | | |  |

**Experiment evaluated by**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor’s Name** | **Sidra Khatoon** | | |
| **Date** |  | **Signature** |  |

**Objective**

The purpose of this lab session is to practice Flutter basics. Create a application on flutter basic widgets.

**Instructions**

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

**Flutter Widgets:**

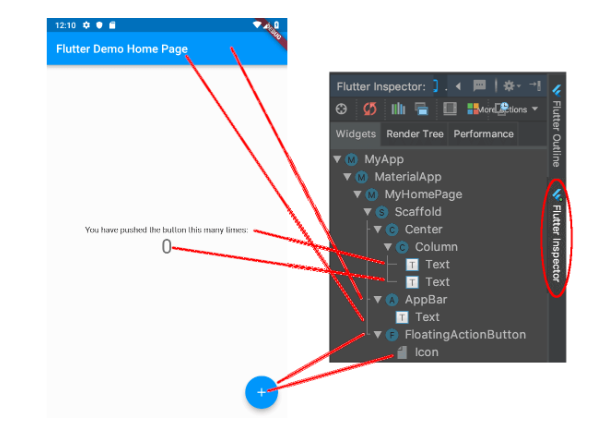
**What is Widget:**

Widgets are just pieces of your user interface. Text is a widget. Buttons are widgets. Check boxes are widgets. Images are widgets. And the list goes on. In fact, everything in the UI is a widget. Even the app itself is a widget! A widget is a blueprint. Flutter uses these blueprints to build the view elements under the hood and render them to the screen.

When you understand that widgets are almost anything that affects how the UI looks and behaves, then it makes sense that there are a lot more widgets than just structural elements like buttons, text, and images. For example, padding is a widget. Layout columns are widgets. Styles are widgets. Even gesture detectors are widgets.

**Widget trees:**

The widgets are arranged into a tree of parent and child widgets. The entire widget tree is what forms the layout that you see on the screen. For example, here is the widget tree for the default demo app when you start a new project. The visible widgets are marked with red lines. (The other widgets in this tree are used for layout and adding functionality.)



**Types of Widgets:**

Widgets are immutable. That is, they cannot be changed. Any properties that they contain are final and can only be set when the widget is initialized. This keeps them lightweight so that it's inexpensive to recreate them when the widget tree changes.

There are two types of widgets: stateless and stateful. Stateless widgets are widgets that don't store any state. That is, they don't store values that might change. For example, an Icon is stateless; you set the icon image when you create it and then it doesn't change any more. A Text widget is also stateless. You might say, "But wait, you can change the text value." True, but if you want to change the text value, you just create a whole new widget with new text. The Text widget doesn't store a text property that can be changed.

The second type of widget is called a stateful widget. That means it can keep track of changes and update the UI based on those changes. Now you might say, "But you said that widgets are immutable! How can they keep track of changes?" Yes, the stateful widget itself is immutable, but it creates a State object that keeps track of the changes. When the values in the State object change, it creates a whole new widget with the updated values. So the lightweight widget (blueprint) gets recreated but the state persists across changes.

**What are Stateful Widgets?**

A Stateful Widget has its own mutable state that it needs to track. It is modified according to the user’s input. A Stateful Widget looks after two things primarily, the changed state based on its previous state and an updated view of the user interface. A track of the previous state value has to be looked at because there is a need to self-rebuild the widget to show the new changes made to your application. A Stateful Widget triggers a build method for creating its children widgets and the subclass of the state holds the related data. It is often used in cases where redrawing of a widget is needed. A Stateful Widget can change when:

* There is a User Input included
* There are some User Interaction
* There are some Dynamic Changes

Given below is the basic structure of a Stateful Widget-

****

Special Methods Associated with Stateful Widget

There are various methods provided by the Stateful class to work with:

1. BuildContext: It provides information regarding which widget is to be built/re-build and where it will be located after re-building. Thus, BuildContext is the widget associated with the state.

Widget build(BuildContext context) {

return Container();

}

2. SetState(): A State object is used to modify the user interface. It executes the code for a particular callback and repaints the widgets that rely on that state for configuration.

setState(setState(() {

});

3. initState(): This method is the entry point of a widget. The initState() method initializes all the methods that the build method will depend upon. This is called only once in a widget’s lifetime and mostly overridden the rest of the time.

initState() {

//...

super.init();

}

4. didChangeDependencies(): It is used for loading the dependencies required for execution of a state. The didChangeDependencies() is called immediately after the initState() is called for the first time and before the triggering of the build method.

void didChangeDependencies() {

}

5. dispose(): This method is used for removing an object permanently from the widget tree. It is used when we need to clear up the memory by invoking super.dispose().

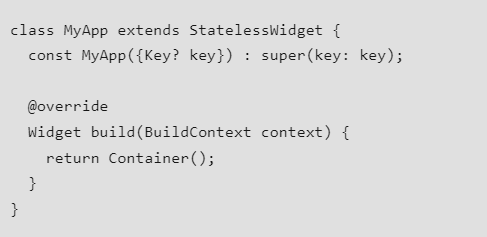
void dispose(){

//...

super.dispose();

}

**What are Stateless Widgets?**

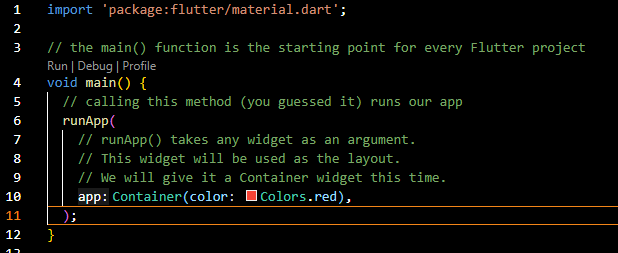
A Stateless Widget is one that does not have any mutable state that it needs to track. The only area of focus of a stateless widget is the information displayed and the user interface. They deal with situations that are independent of the user’s input. A Stateless Widget does not tell the framework when to remove it or rebuild it, it gets a command from the framework itself. They create user interfaces that do not change dynamically upon updation in the nearby values. We use a stateless widget when we create an application that does not require redrawing a widget again and again. Given below is the basic structure of a stateless widget known as GreenFrog. 

The MyApp class is a Stateless Widget and Widget build is a method with BuildContext as a parameter that returns widgets. Every widget while entering the Widget build(Build Context context) has to override it in order to enter the widget tree. Some Examples of a stateless widget are Text, IconButton, AppBar, etc. Build Method is called inside a stateless widget in only three cases:

* Initially in the start when it is built for the very first time while starting the application.
* If the parent widget is changed
* If the inherited widget is changed

**Container Widgets:**

The first widget we are going to play with is called a Container. As you might have guessed from the name, it's a holder for other widgets. But we aren't going to put anything else in it to start with. We will just play with its color property.

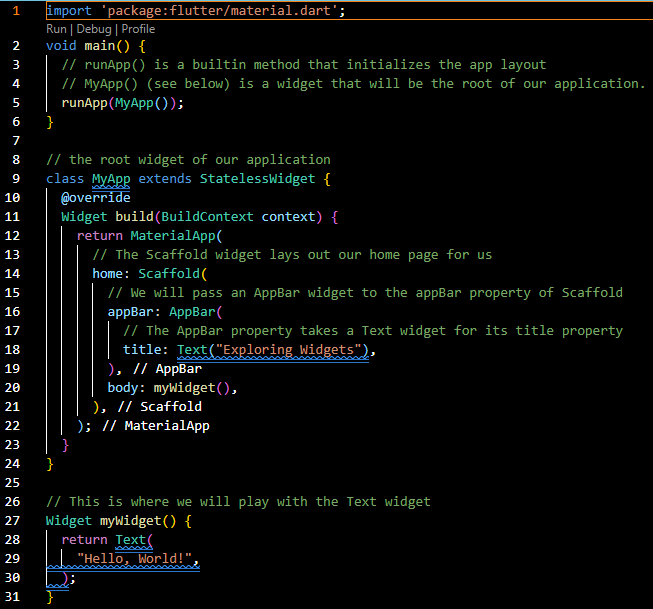


Now, replace Colors.green with other values.

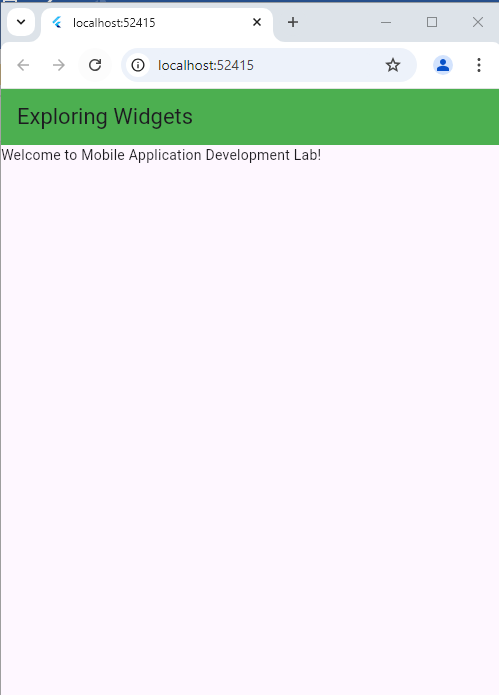
* Colors.red
* Colors.blueAccent
* Colors.deepPurple

**Text Widgets:**

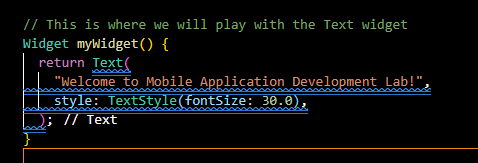
Probably every single app that you make will have text, so the Text widget is definitely one that we need to look at.



Change the text from "Hello, World!" to "Welcome to Mobile Application Development Lab!" and then do a hot reload.

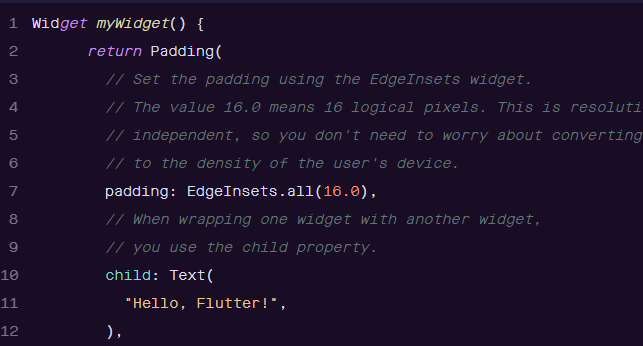


If you want to increase the font size, you can add a TextStyle widget to the style property of Text. Replace the myWidget() method above with the following:



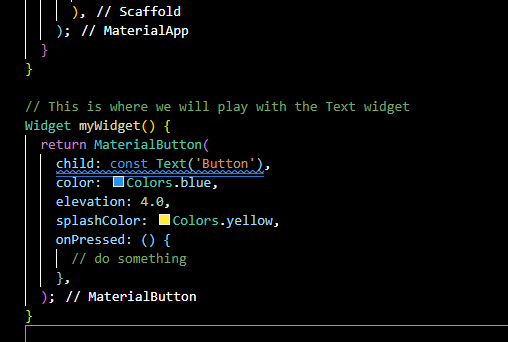
There are lots of other changes you can make with the TextStyle widget, like color, font, shadows, and spacing to name a few.

If you want to add padding, you don't change a property. Instead, you wrap the Text widget with a Padding widget. In Flutter lots of layout related tasks use widgets instead of setting properties. Remember, a widget is a blueprint that affects how the UI looks.



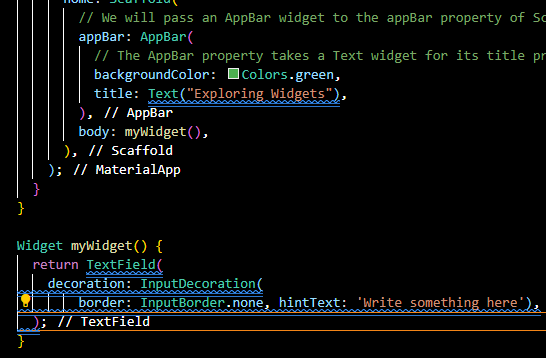
**Button Widgets:**

Buttons are another common need and Flutter has several types of button widgets. Although we are not doing anything in response to the button click in this tutorial, you can see in the code below where you could do something.



**TextFied Widgets:**

Button Widgets: For accepting user text input you use a TextField widget. Now that you already have experience with the widgets above, this one is simple. You just click in the TextField and the system keyboard automatically pops up.



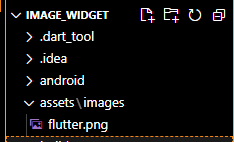
**Image Widgets:**

In this section, we are going to see how we can display images in Flutter. When you create an app in Flutter, it includes both code and assets (resources). An asset is a file, which is bundled and deployed with the app and is accessible at runtime. The asset can include static data, configuration files, icons, and images. The Flutter supports many image formats, such as JPEG, WebP, PNG, GIF, animated WebP/GIF, BMP, and WBMP.

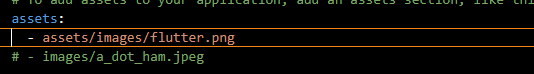
How to display the image in Flutter?

To display an image in Flutter, do the following steps:

* Step 1: First, we need to create a new folder inside the root of the Flutter project and named it assets. We can also give it any other name if you want.
* Step 2: Next, inside this folder, add one image manually.

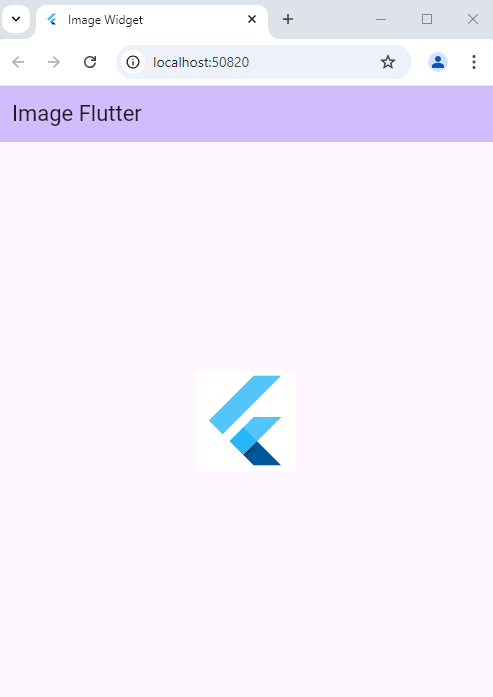


* Step 3: Update the pubspec.yaml file. Suppose the image name is tablet.png, then pubspec.yaml file is:



* Step 4: Finally, open the main.dart file and insert the following code.





**Flutter Row and Column:**

Row and Column are the two most important and powerful widgets in Flutter. These widgets let you align children horizontally and vertically as per the requirement. As we know that when we design any UI(User Interface) in a flutter, we need to arrange its content in the Row and Column manner so these Row and Column widgets are required when designing UI.

Key Points:

1. Row and Column widgets are the most commonly used layout patterns in the Flutter application.
2. Both may take several child widgets.
3. A child widget can also be a row or column widget.
4. We can stretch or constrain a particular children's widget.
5. Flutter also allows developers to specify how child widgets can use row and column widgets' available space.

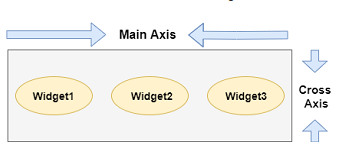
**Row Widgets:**

This widget arranges its children in a horizontal direction on the screen. In other words, it will expect child widgets in a horizontal array. If the child widgets need to fill the available horizontal space, we must wrap the children widgets in an Expanded widget.

A row widget does not appear scrollable because it displays the widgets within the visible view. So it is considered wrong if we have more children in a row which will not fit in the available space. If we want to make a scrollable list of row widgets, we need to use the ListView widget.

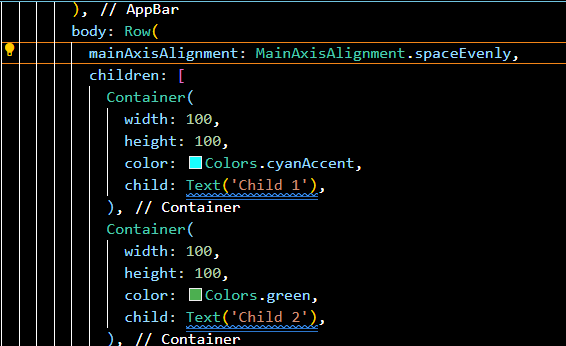
****

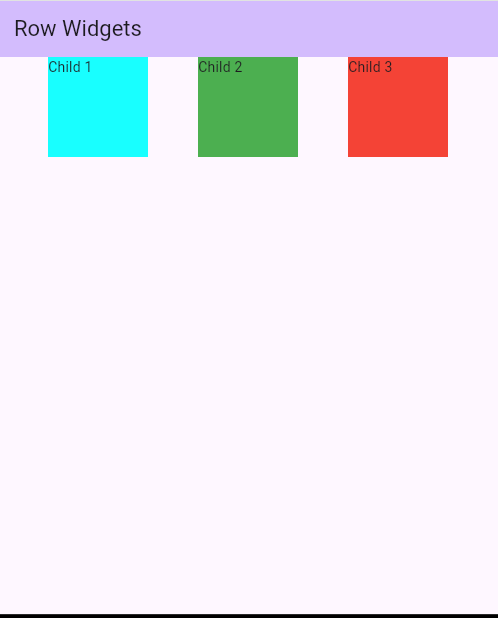
We can control how a row widget aligns its children based on our choice using the property crossAxisAlignment and mainAxisAlignment. The row's cross-axis will run vertically, and the main axis will run horizontally. See the below visual representation to understand it more clearly.



We can align the row's children widget with the help of the following properties:

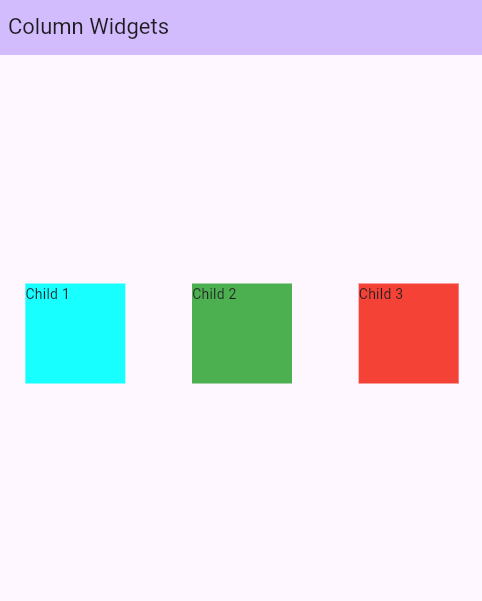
* start: It will place the children from the starting of the main axis.
* end: It will place the children at the end of the main axis.
* center: It will place the children in the middle of the main axis.
* spaceBetween: It will place the free space between the children evenly.
* spaceAround: It will place the free space between the children evenly and half of that space before and after the first and last children widget.
* spaceEvenly: It will place the free space between the children evenly and before and after the first and last children widget.

Main axis alignment of box around the width



Cross axis alignment of box around the height

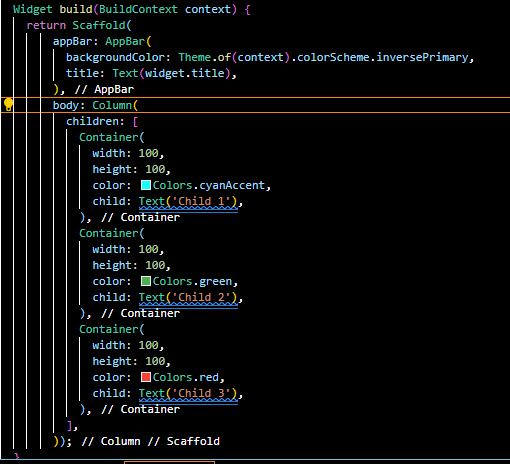




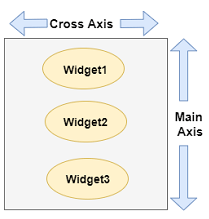
**Column Widgets:**

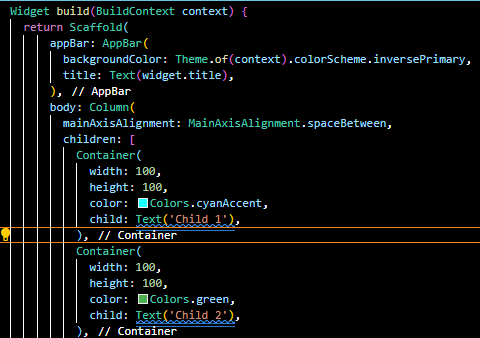
This widget arranges its children in a vertical direction on the screen. In other words, it will expect a vertical array of children widgets. If the child widgets need to fill the available vertical space, we must wrap the children widgets in an Expanded widget.

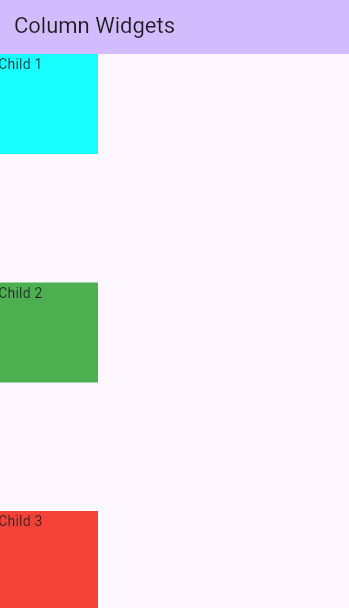
A column widget does not appear scrollable because it displays the widgets within the visible view. So it is considered wrong if we have more children in a column which will not fit in the available space. If we want to make a scrollable list of column widgets, we need to use the ListView Widget.



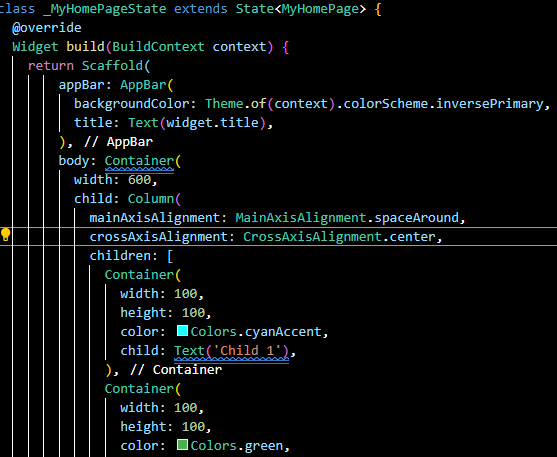
We can also control how a column widget aligns its children using the property mainAxisAlignment and crossAxisAlignment. The column's cross-axis will run horizontally, and the main axis will run vertically. The below visual representation explains it more clearly.

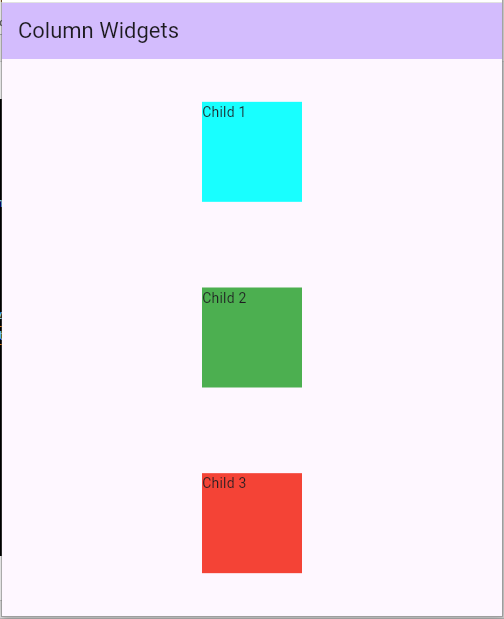


Main axis alignment of box around the height



Cross axis alignment of box around the width





**Assessment:**

1: You have to create a colorful container in which text is center align. Container is also center aligned by using padding widget.

2: Create a simple app in which use column and row widget to align 3 or 4 images in row and column. Create 2 app one for column and other for column. Also add app bar in application. Submit me code and screenshot of output.