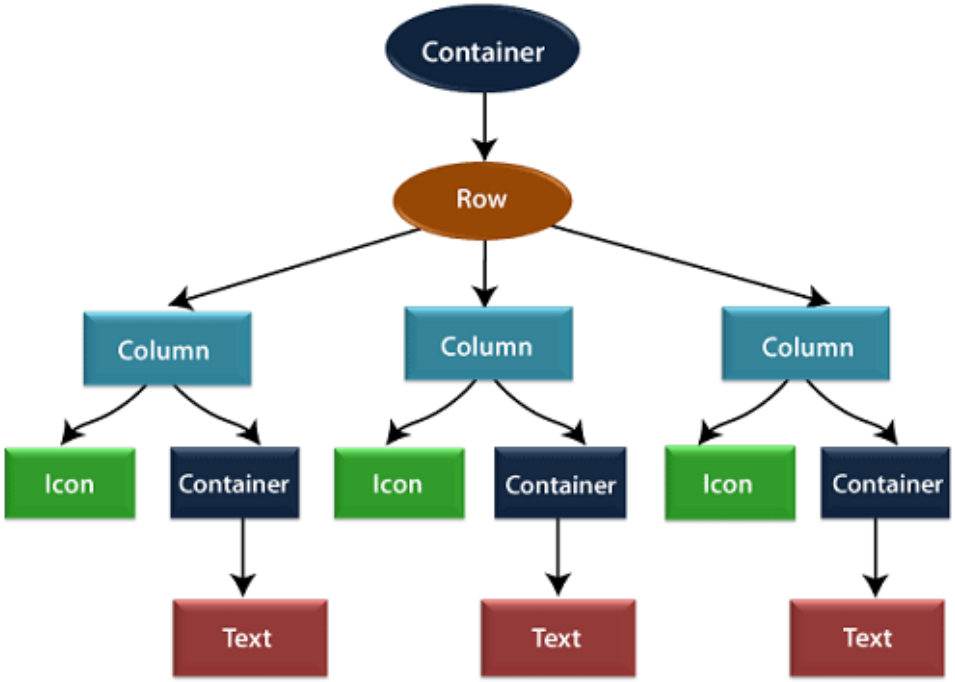
Flutter Layouts

The main concept of the layout mechanism is the widget. We know that flutter assume everything as a widget. So the image, icon, text, and even the layout of your app are all widgets. Here, some of the things you do not see on your app UI, such as rows, columns, and grids that arrange, constrain, and align the visible widgets are also the widgets.

Flutter allows us to create a layout by composing multiple widgets to build more complex widgets. **For example**, we can see the below image that shows three icons with a label under each one.

In the second image, we can see the visual layout of the above image. This image shows a row of three columns, and these columns contain an icon and label.



In the above image, the **container** is a widget class that allows us to customize the child widget. It is mainly used to add borders, padding, margins, background color, and many more. Here, the text widget comes under the container for adding margins. The entire row is also placed in a container for adding margin and padding around the row. Also, the rest of the UI is controlled by properties such as color, text.style, etc.

Layout a widget

Let us learn how we can create and display a simple widget. The following steps show how to layout a widget:

Step 1: First, you need to select a Layout widget.

Step 2: Next, create a visible widget.

Step 3: Then, add the visible widget to the layout widget.

Step 4: Finally, add the layout widget to the page where you want to display.

Types of Layout Widgets

We can categories the layout widget into two types:

1. Single Child Widget
2. Multiple Child Widget

Single Child Widgets

The single child layout widget is a type of widget, which can have only **one child widget** inside the parent layout widget. These widgets can also contain special layout functionality. Flutter provides us many single child widgets to make the app UI attractive. If we use these widgets appropriately, it can save our time and makes the app code more readable. The list of different types of single child widgets are:

Container: It is the most popular layout widget that provides customizable options for painting, positioning, and sizing of widgets.

```
1. Center(
2.   child: Container(
```

```
    margin: const EdgeInsets.all(15.0),
    color: Colors.blue,
    width: 42.0,
    height: 42.0,
  ),
)
```

Padding: It is a widget that is used to arrange its child widget by the given padding. It contains **EdgeInsets** and **EdgeInsets.fromLTRB** for the desired side where you want to provide padding.

```
const Greetings(
  child: Padding(
    padding: EdgeInsets.all(14.0),
    child: Text('Hello JavaTpoint!'),
  ),
)
```

Center: This widget allows you to center the child widget within itself.

Align: It is a widget, which aligns its child widget within itself and sizes it based on the child's size. It provides more control to place the child widget in the exact position where you need it.

```
Center(
  child: Container(
    height: 110.0,
    width: 110.0,
    color: Colors.blue,
    child: Align(
      alignment: Alignment.topLeft,
      child: FlutterLogo(
        size: 50,
      ),
    ),
  ),
),
)
```

SizedBox: This widget allows you to give the specified size to the child widget through all screens.

```
SizedBox(
  width: 300.0,
  height: 450.0,
  child: const Card(child: Text('Hello JavaTpoint!')),
)
```

AspectRatio: This widget allows you to keep the size of the child widget to a specified aspect ratio.

```
AspectRatio(
  aspectRatio: 5/3,
  child: Container(
    color: Colors.bluel,
  ),
),
```

Baseline: This widget shifts the child widget according to the child's baseline.

```
child: Baseline(
        baseline: 30.0,
        baselineType: TextBaseline.alphabetic,
        child: Container(
            height: 60,
            width: 50,
            color: Colors.blue,
        ),
)
```

ConstrainedBox: It is a widget that allows you to force the additional constraints on its child widget. It means you can force the child widget to have a specific constraint without changing the properties of the child widget.

```
ConstrainedBox(
  constraints: new BoxConstraints(
    minHeight: 150.0,
    minWidth: 150.0,
```

```
    maxHeight: 300.0,
    maxWidth: 300.0,
  ),
  child: new DecoratedBox(
    decoration: new BoxDecoration(color: Colors.red),
  ),
),
```

CustomSingleChildLayout: It is a widget, which defers from the layout of the single child to a delegate. The delegate decides to position the child widget and also used to determine the size of the parent widget.

FittedBox: It scales and positions the child widget according to the specified **fit**.

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // It is the root widget of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multiple Layout Widget',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        // This is the theme of your application.
        primarySwatch: Colors.green,
      ),
      home: MyHomePage(),
    );
  }
}
class MyHomePage extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```
25.       appBar: AppBar(title: Text("FittedBox Widget")),
26.       body: Center(
27.       child: FittedBox(child: Row(
28.         children: <Widget>[
29.           Container(
30.             child: Image.asset('assets/computer.png'),
31.             ),
32.             Container(
33.               child: Text("This is a widget"),
34.             )
35.           ],
36.         ),
37.         fit: BoxFit.contain,
38.       )
39.     ),
40.   );
41.  }
42.}
```

Output

FractionallySizedBox: It is a widget that allows to sizes of its child widget according to the fraction of the available space.

IntrinsicHeight and IntrinsicWidth: They are a widget that allows us to sizes its child widget to the child's intrinsic height and width.

LimitedBox: This widget allows us to limits its size only when it is unconstrained.

Offstage: It is used to measure the dimensions of a widget without bringing it on to the screen.

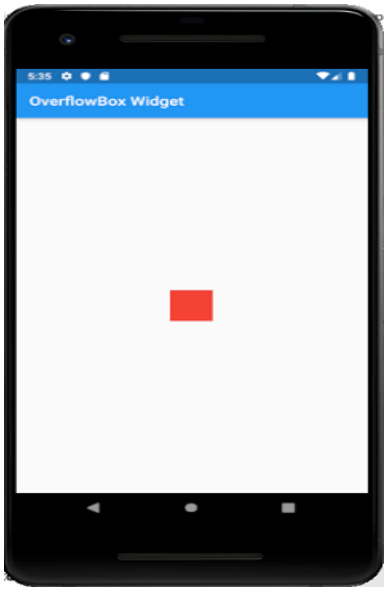
OverflowBox: It is a widget, which allows for imposing different constraints on its child widget than it gets from a parent. In other words, it allows the child to overflow the parent widget.

Example

```
1. import 'package:flutter/material.dart';
2.
3. void main() => runApp(MyApp());
4.
5. class MyApp extends StatelessWidget {
6.   // It is the root widget of your application.
7.   @override
8.   Widget build(BuildContext context) {
9.     return MaterialApp(
10.      title: 'Single Layout Widget',
11.      debugShowCheckedModeBanner: false,
12.      theme: ThemeData(
13.        // This is the theme of your application.
14.        primarySwatch: Colors.blue,
15.      ),
16.      home: MyHomePage(),
17.    );
18. }
19.}
20.class MyHomePage extends StatelessWidget {
21.
22. @override
```

```
23.  Widget build(BuildContext context) {
24.    return Scaffold(
25.      appBar: AppBar(
26.        title: Text("OverflowBox Widget"),
27.      ),
28.      body: Center(
29.      child: Container(
30.        height: 50.0,
31.        width: 50.0,
32.        color: Colors.red,
33.        child: OverflowBox(
34.          minHeight: 70.0,
35.          minWidth: 70.0,
36.          child: Container(
37.            height: 50.0,
38.            width: 50.0,
39.            color: Colors.blue,
40.            ),
41.          ),
42.        ),
43.      ),
44.    );
45.  }
46.}
```

Output



Multiple Child widgets

The multiple child widgets are a type of widget, which contains **more than one child widget**, and the layout of these widgets are **unique**. For example, Row widget laying out of its child widget in a horizontal direction, and Column widget laying out of its child widget in a vertical direction. If we combine the Row and Column widget, then it can build any level of the complex widget.

Here, we are going to learn different types of multiple child widgets:

Row: It allows to arrange its child widgets in a horizontal direction.

Example

```
1.  import 'package:flutter/material.dart';
2.
3.  void main() => runApp(MyApp());
4.
5.  class MyApp extends StatelessWidget {
6.    // It is the root widget of your application.
7.    @override
8.    Widget build(BuildContext context) {
9.      return MaterialApp(
10.       title: 'Multiple Layout Widget',
11.       debugShowCheckedModeBanner: false,
12.       theme: ThemeData(
```

```
      // This is the theme of your application.
      primarySwatch: Colors.blue,
    ),
    home: MyHomePage(),
  );
 }
}
class MyHomePage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
  return Center(
   child: Container(
    alignment: Alignment.center,
    color: Colors.white,
    child: Row(
     children: <Widget>[
      Expanded(
       child: Text('Peter', textAlign: TextAlign.center),
      ),
      Expanded(
       child: Text('John', textAlign: TextAlign.center ),

      ),
      Expanded(
       child: FittedBox(
        fit: BoxFit.contain, // otherwise the logo will be tiny
        child: const FlutterLogo(),
       ),
      ),
     ],
    ),
   ),
  );
 }
}
```

Output

Column: It allows to arrange its child widgets in a vertical direction.

ListView: It is the most popular scrolling widget that allows us to arrange its child widgets one after another in scroll direction.

GridView: It allows us to arrange its child widgets as a scrollable, 2D array of widgets. It consists of a repeated pattern of cells arrayed in a horizontal and vertical layout.

Expanded: It allows to make the children of a Row and Column widget to occupy the maximum possible area.

Table: It is a widget that allows us to arrange its children in a table based widget.

Flow: It allows us to implements the flow-based widget.

Stack: It is an essential widget, which is mainly used for overlapping several children widgets. It allows you to put up the multiple layers onto the screen. The following example helps to understand it.

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

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // It is the root widget of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multiple Layout Widget',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        // This is the theme of your application.
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(),
    );
  }
}
class MyHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        alignment: Alignment.center,
        color: Colors.white,
        child: Stack(
          children: <Widget>[
            // Max Size
            Container(
              color: Colors.blue,
            ),
            Container(
              color: Colors.pink,
              height: 400.0,
              width: 300.0,
            ),
```

```
38.             Container(
39.               color: Colors.yellow,
40.               height: 220.0,
41.               width: 200.0,
42.             )
43.           ],
44.         ),
45.       ),
46.     );
47.   }
48. }
```

Output