**Course: 405-02: Mobile Application Development – 2**

**Unit-4: Introduction of Flutter:**

4.1 Fundamentals of Flutter:

**What is Flutter?**

In general, creating a mobile application is a very complex and challenging task. There are many frameworks available, which provide excellent features to develop mobile applications. For developing mobile apps, Android provides a native framework based on Java and Kotlin language, while iOS provides a framework based on Objective-C/Swift language. Thus, we need two different languages and frameworks to develop applications for both OS. Today, to overcome form this complexity, there are several frameworks have introduced that support both OS along with desktop apps. These types of the framework are known as **cross-platform** development tools.

The cross-platform development framework has the ability to write one code and can deploy on the various platform (Android, iOS, and Desktop). It saves a lot of time and development efforts of developers. There are several tools available for cross-platform development, including web-based tools, such as Ionic from Drifty Co. in 2013, Phonegap from Adobe, Xamarin from Microsoft, and React Native form Facebook. Each of these frameworks has varying degrees of success in the mobile industry. In recent, a new framework has introduced in the cross-platform development family named **Flutter** developed from Google.

Flutter is a UI toolkit for creating fast, beautiful, natively compiled applications for mobile, web, and desktop with one programing language and single codebase. It is free and open-source. It was initially developed from **Google** and now manages by an **ECMA** standard. Flutter apps use Dart programming language for creating an app. The **dart programming** shares several same features as other programming languages, such as Kotlin and Swift, and can be trans-compiled into JavaScript code.

Flutter is mainly optimized for 2D mobile apps that can run on both Android and iOS platforms. We can also use it to build full-featured apps, including camera, storage, geolocation, network, third-party SDKs, and more.

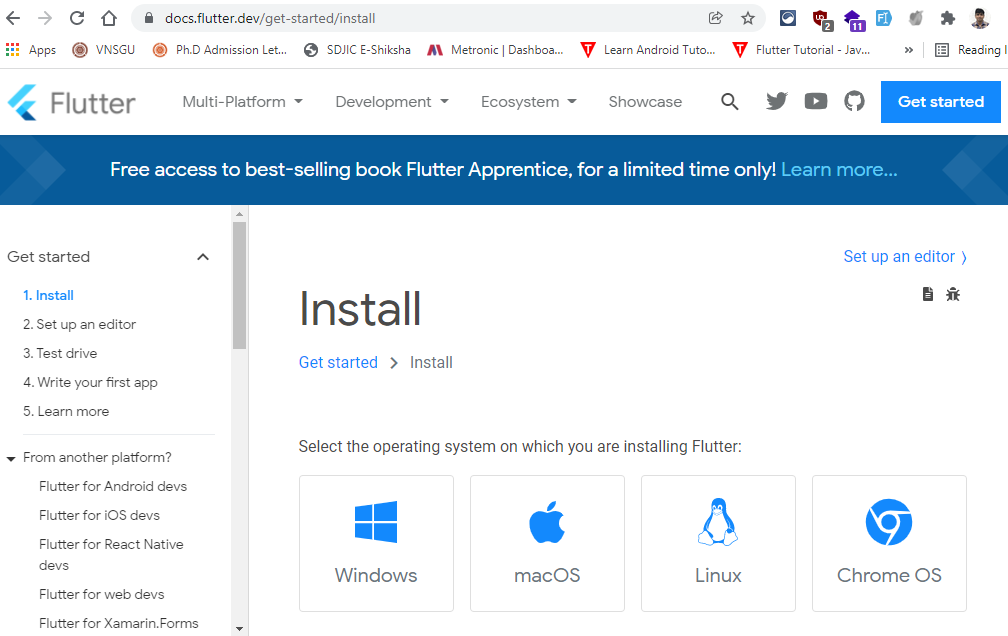
**What makes Flutter unique?**

Flutter is different from other frameworks because it neither uses **WebView** nor the **OEM** widgets that shipped with the device. Instead, it uses its own high-performance rendering engine to draw widgets. It also implements most of its systems such as animation, gesture, and widgets in Dart programing language that allows developers to read, change, replace, or remove things easily. It gives excellent control to the developers over the system.

4.1.1 Installation and Architecture of Flutter

Install the Flutter SDK

**Step 1:** Download the installation bundle of the Flutter Software Development Kit for windows. To download Flutter SDK, Go to its official website**(https://flutter.dev/)**, click on Get started button, you will get the following screen.



**Get the Flutter SDK**

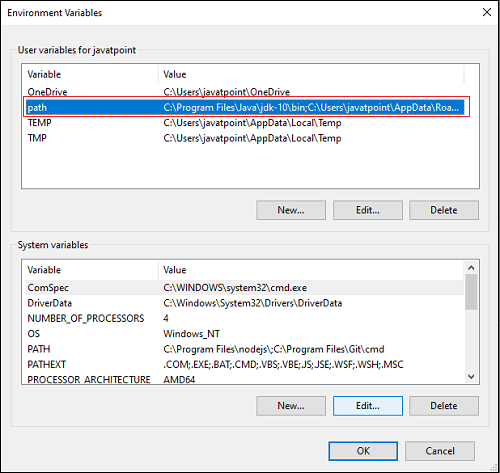
**Step 2:**Download the following installation bundle to get the latest stable release of the Flutter SDK:

*https://storage.googleapis.com/flutter\_infra\_release/releases/stable/windows/flutter\_windows\_2.8.1-stable.zip*

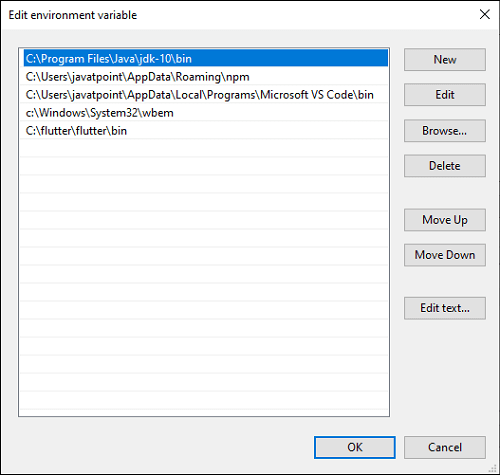
**Step 3:**Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\Users\<your-user-name>\Documents).

**Step 4:** To run the Flutter command in regular windows console, you need to update the system path to include the flutter bin directory. The following steps are required to do this:

**Step 4.1:** Go to MyComputer properties -> advanced tab -> environment variables. You will get the following screen.



**Step 4.2:** Now, select path -> click on edit. The following screen appears.

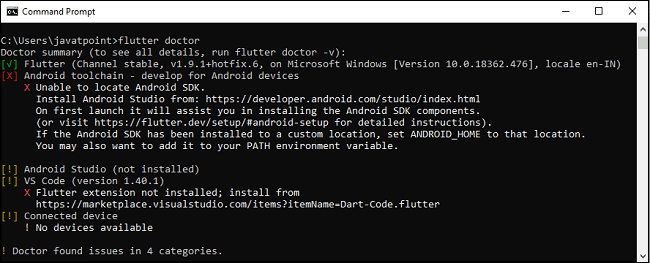


**Step 4.3:** In the above window, click on New->write path of Flutter bin folder in variable value -> ok -> ok -> ok.

**Step 5:** Now, run the $ **flutter doctor** command. This command checks for all the requirements of Flutter app development and displays a report of the status of your Flutter installation.

$ flutter doctor

**Step 6:** When you run the above command, it will analyze the system and show its report, as shown in the below image. Here, you will find the details of all missing tools, which required to run Flutter as well as the development tools that are available but not connected with the device.

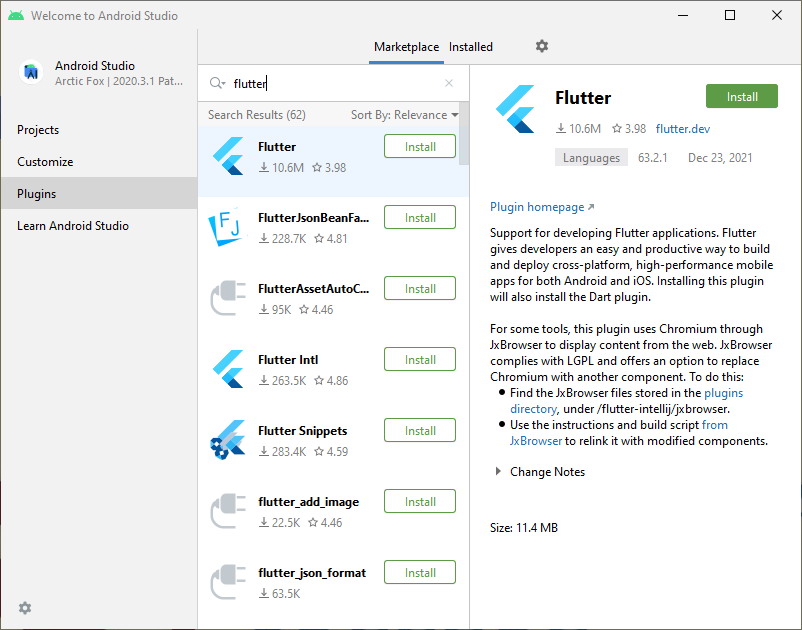


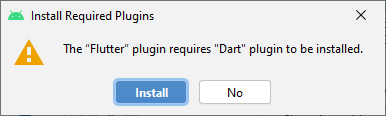
**Install Flutter/Dart Plugins for Android Studio**

Install Flutter and Dart plugin for building Flutter application in Android Studio. These plugins provide a template to create a Flutter application, give an option to run and debug Flutter application in the Android Studio itself. Do the following steps to install these plugins.

Open the Android Studio and then go to File->Settings->Plugins.

Now search the Flutter plugin. If found, select Flutter plugin and click install. When you click on install, it will ask you to install Dart plugin as below screen. Click yes to proceed.





**Set Environment Variables**

Create two System/User Environment Variable and set following values.

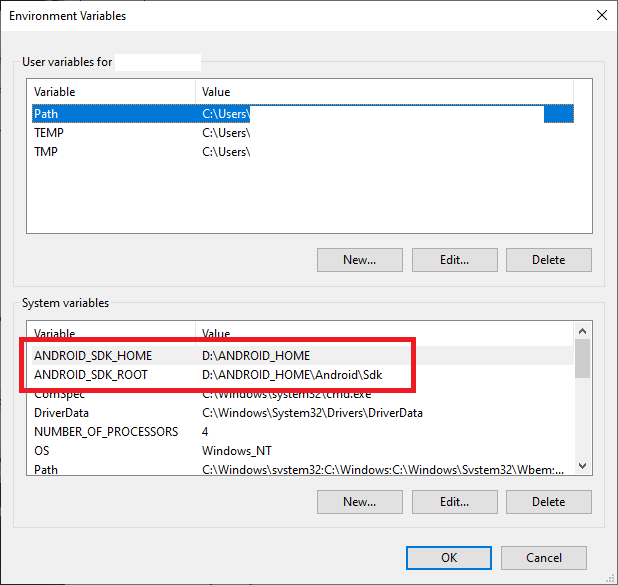
Variable Name: ANDROID\_SDK\_HOME

Value: C:\Users\{YOUR\_USER\_NAME}

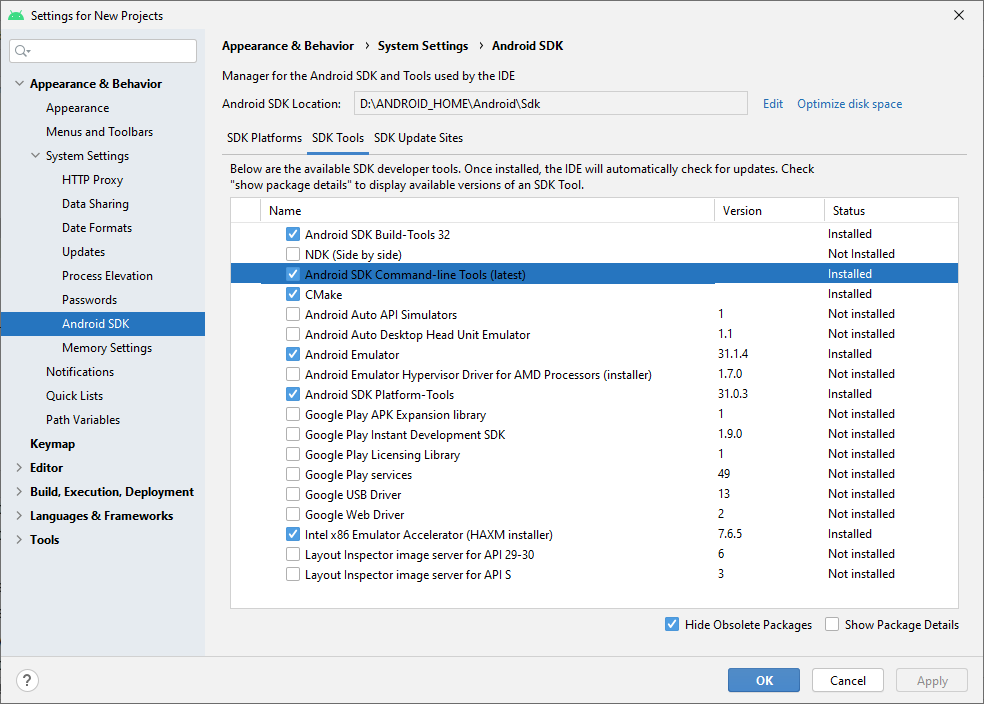
(Or provide any directory where you want to keep all the android libraries and Virtual devices)

Variable Name: ANDROID\_SDK\_ROOT

Value: C:\Users\{YOUR\_USER\_NAME}\AppData\Local\Android\Sdk



**Install Android SDK command line tools**

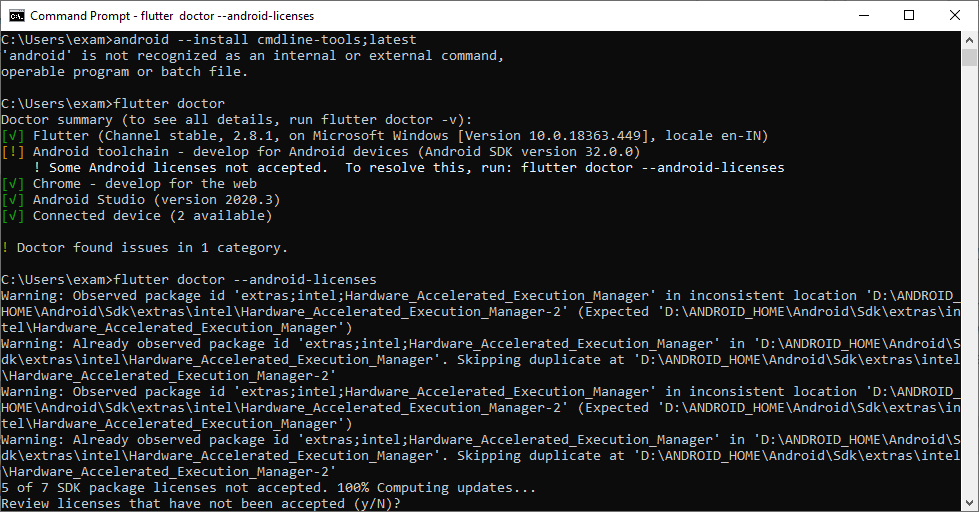


**Accept All Android Licenses:**

We need to accept all the android licenses to run the flutter applications built over Android SDK. To accept all the licenses, we need to run following command in the terminal window.

$ flutter doctor --android-licenses

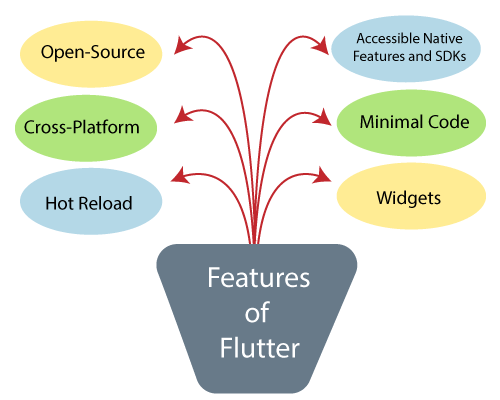
Read the licenses if you want and keep pressing “Y” to accept all the licenses.



Now we are ready to build our first Flutter Application using Android Studio and Android SDK.

4.1.2 Features of Flutter

Flutter gives easy and simple methods to start building beautiful mobile and desktop apps with a rich set of material design and widgets. Here, we are going to discuss its main features for developing the mobile framework.



**Open-Source:** Flutter is a free and open-source framework for developing mobile applications.

**Cross-platform:** This feature allows Flutter to write the code once, maintain, and can run on different platforms. It saves the time, effort, and money of the developers.

**Hot Reload:** Whenever the developer makes changes in the code, then these changes can be seen instantaneously with Hot Reload. It means the changes immediately visible in the app itself. It is a very handy feature, which allows the developer to fix the bugs instantly.

**Accessible Native Features and SDKs:** This feature allows the app development process easy and delightful through Flutter's native code, third-party integration, and platform APIs. Thus, we can easily access the SDKs on both platforms.

**Minimal code:** Flutter app is developed by Dart programming language, which uses JIT and AOT compilation to improve the overall start-up time, functioning and accelerates the performance. JIT enhances the development system and refreshes the UI without putting extra effort into building a new one.

**Widgets:** The Flutter framework offers widgets, which are capable of developing customizable specific designs. Most importantly, Flutter has two sets of widgets: Material Design and Cupertino widgets that help to provide a glitch-free experience on all platforms.

Advantage of Flutter

Flutter fulfills the custom needs and requirements for developing mobile applications. It also offers many advantages, which are listed below.

* It makes the app development process extremely fast because of the hot-reload feature. This feature allows us to change or update the code are reflected as soon as the alterations are made.
* It provides the smoother and seamless scrolling experiences of using the app without much hangs or cuts, which makes running applications faster in comparison to other mobile app development frameworks.
* Flutter reduces the time and efforts of testing. As we know, flutter apps are cross-platform so that testers do not always need to run the same set of tests on different platforms for the same app.
* It has an excellent user interface because it uses a design-centric widget, high-development tools, advanced APIs, and many more features.
* It is similar to a reactive framework where the developers do not need to update the UI content manually.
* It is suitable for MVP (Minimum Viable Product) apps because of its speedy development process and cross-platform nature.

Disadvantages of Flutter

We have seen earlier that the Flutter has many advantages, but it also contains some disadvantages, which are given below.

* The Flutter is a comparatively new language that needs continuous integration support through the maintenance of scripts.
* It provides very limited access to SDK libraries. It means a developer does not have a lot of functionalities to create a mobile application. Such types of functionalities need to be developed by the Flutter developer themselves.
* The Flutter apps do not support the browser. It only supports Android and iOS platforms.
* It uses Dart programming for coding, so a developer needs to learn new technologies. However, it is easy to learn for developers.

**History of Flutter**

Flutter is a free and open-source UI software development kit introduced by Google. It is used to build applications for Android, iOS, Windows, and the web. The first version of Flutter was announced in the year **2015** at the **Dart Developer Summit**. It was initially known as codename **"Sky"** and can run on the Android OS. After the announcement of Flutter, the first Flutter Alpha version (v-0.06) was released in May **2017**.

Later, during the keynote of Google Developer days in Shanghai, Google launched the second preview of Flutter in **September 2018** that was the last big release before Flutter 1.0 version. On **December 4, 2018**, the first stable version of the Flutter framework was released at the Flutter Live event, denoting Flutter 1.0. The current stable release of the framework is Flutter v1.9.1+hotfix.6 on October 24, 2019.

**Advantages of Flutter**

Flutter comes with beautiful and customizable widgets for high performance and outstanding mobile application. It fulfills all the custom needs and requirements. Besides these, Flutter offers many more advantages as mentioned below −

* Dart has a large repository of software packages which lets you to extend the capabilities of your application.
* Developers need to write just a single code base for both applications (both Android and iOS platforms). *Flutter* may to be extended to other platform as well in the future.
* Flutter needs lesser testing. Because of its single code base, it is sufficient if we write automated tests once for both the platforms.
* Flutter’s simplicity makes it a good candidate for fast development. Its customization capability and extendibility makes it even more powerful.
* With Flutter, developers has full control over the widgets and its layout.
* Flutter offers great developer tools, with amazing hot reload.

Disadvantages of Flutter

Despite its many advantages, flutter has the following drawbacks in it −

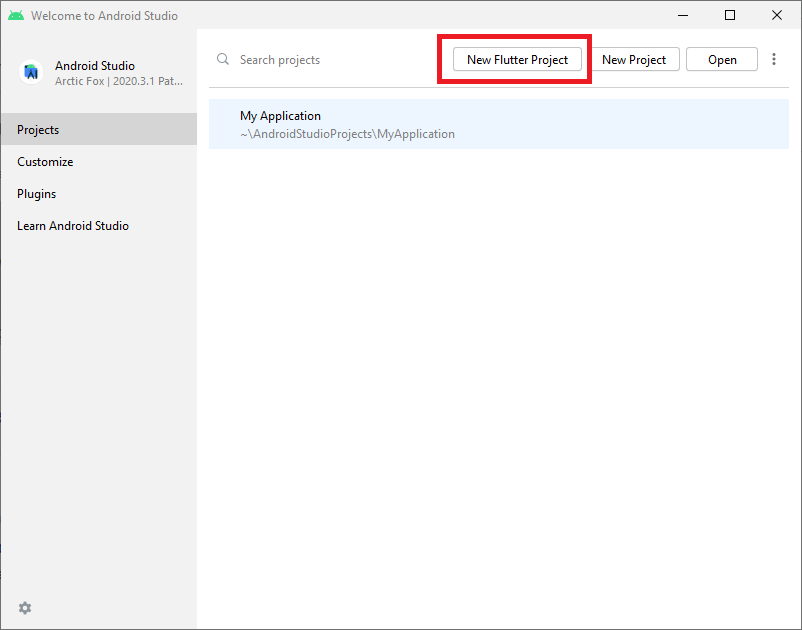
* Since it is coded in Dart language, a developer needs to learn new language (though it is easy to learn).
* Modern framework tries to separate logic and UI as much as possible but, in Flutter, user interface and logic is intermixed. We can overcome this using smart coding and using high level module to separate user interface and logic.
* Flutter is yet another framework to create mobile application. Developers are having a hard time in choosing the right development tools in hugely populated segment.

**4.1.3 Creating basic flutter project using Android Studio**

In this section, we are going to learn how to create a simple application in Android Studio to understand the basics of the Flutter application. To create Flutter application, do the following steps:

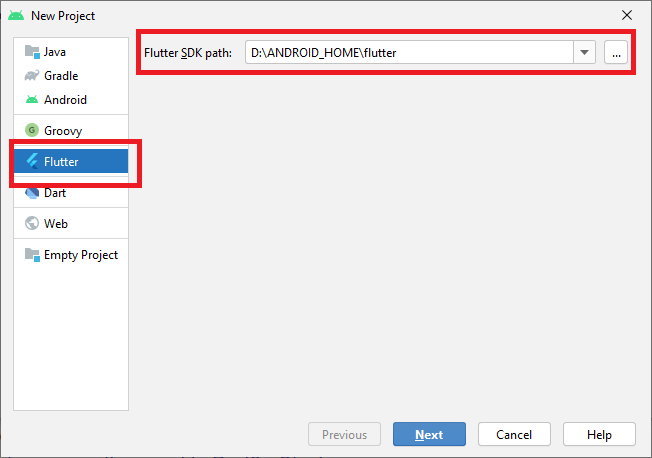
**Step 1:** Open the Android Studio.

**Step 2:** Create the Flutter project. To create a project, go to File-> New->New Flutter Project. The following screen helps to understand it more clearly.

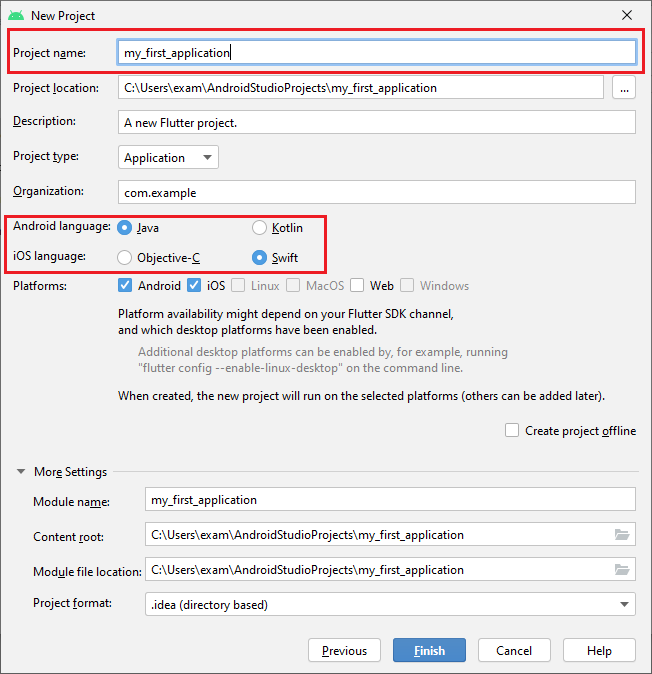


**Step 3:** Select Flutter SDK path if not set. In following image Flutter SDK stored in following location.

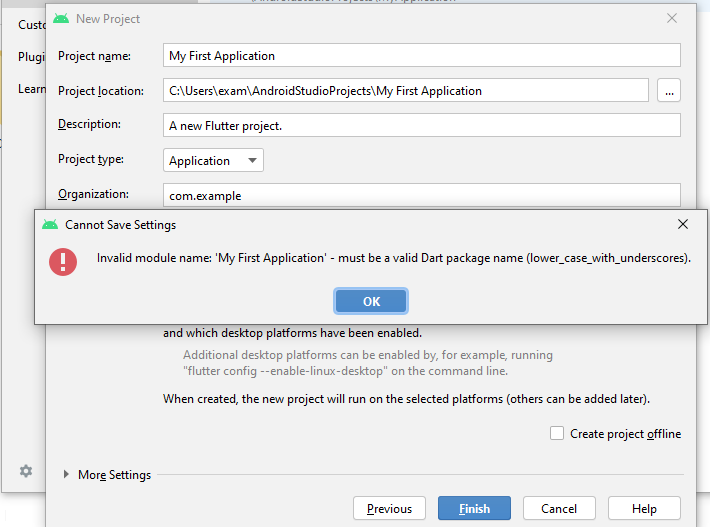
D:\ANDROID\_HOME\flutter



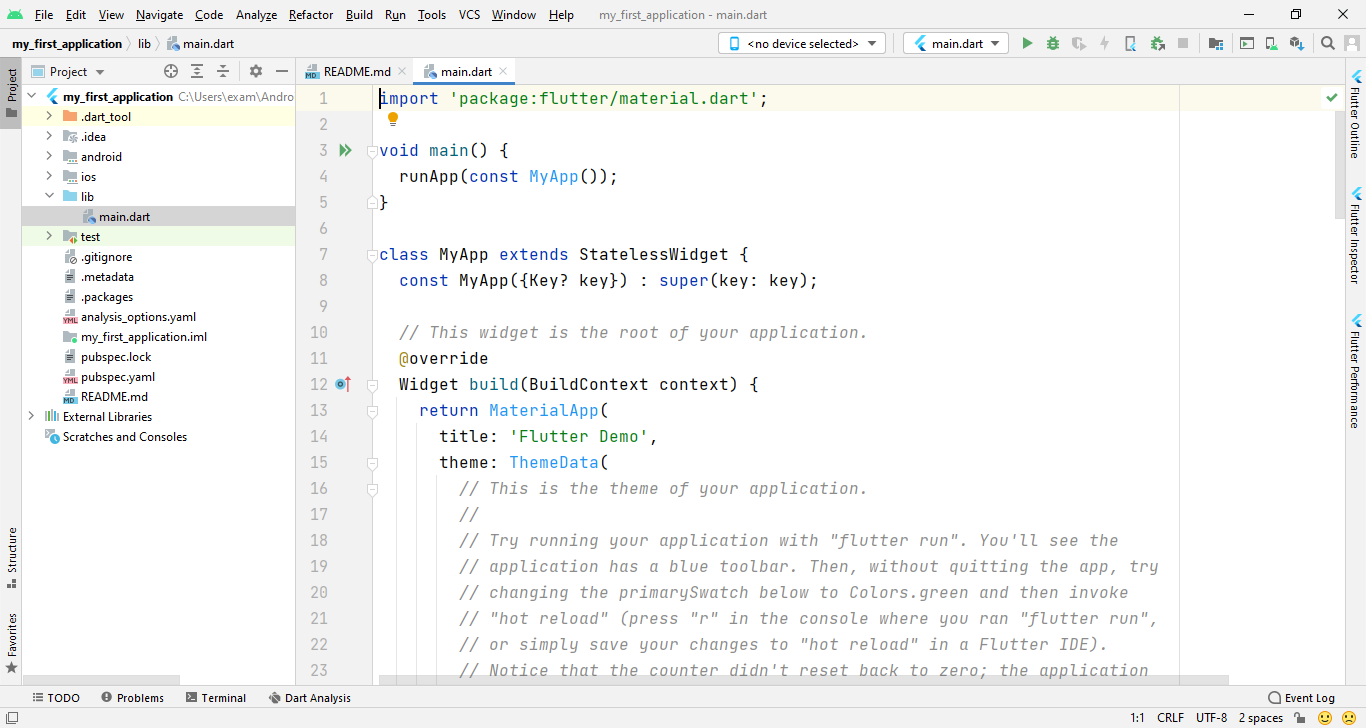
**Step 4:** Assign a Project name by using following settings. Project name should be in lowercase separated by underscore sign.

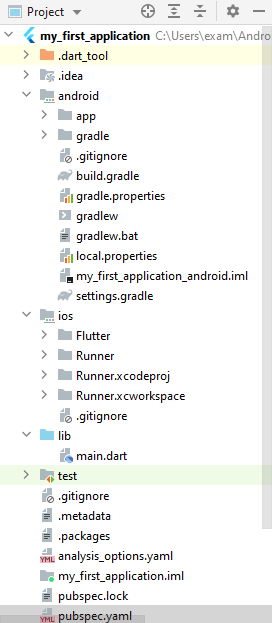


Android studio will throw error while given any invalid project name.



**STEP 5:** Android Studio will show you project file and folders in Project windows. If its not showing all the files as following image, in the Project window click on “Android” > Select “Project” and it will open a tree view as shown below image.





**.idea:** This folder is at the very top of the project structure, which holds the configuration for Android Studio. It doesn't matter because we are not going to work with Android Studio so that the content of this folder can be ignored.

**.android:** This folder holds a complete Android project and used when you build the Flutter application for Android. When the Flutter code is compiled into the native code, it will get injected into this Android project, so that the result is a native Android application. **For Example:** When you are using the Android emulator, this Android project is used to build the Android app, which further deployed to the Android Virtual Device.

**.ios:** This folder holds a complete Mac project and used when you build the Flutter application for iOS. It is similar to the android folder that is used when developing an app for Android. When the Flutter code is compiled into the native code, it will get injected into this iOS project, so that the result is a native iOS application. Building a Flutter application for iOS is only possible when you are working on macOS.

**.lib:** It is an essential folder, which stands for the library. It is a folder where we will do our 99 percent of project work. Inside the lib folder, we will find the Dart files which contain the code of our Flutter application. By default, this folder contains the file **main.dart**, which is the entry file of the Flutter application.

**.test:** This folder contains a Dart code, which is written for the Flutter application to perform the automated test when building the app. It won't be too important for us here.

We can also have some default files in the Flutter application. In 99.99 percent of cases, we don't touch these files manually. These files are:

**.gitignore:** It is a text file containing a list of files, file extensions, and folders that tells Git which files should be ignored in a project. Git is a version-control file for tracking changes in source code during software development Git.

**.metadata:** It is an auto-generated file by the flutter tools, which is used to track the properties of the Flutter project. This file performs the internal tasks, so you do not need to edit the content manually at any time.

**.packages:** It is an auto-generated file by the Flutter SDK, which is used to contain a list of dependencies for your Flutter project.

**flutter\_demoapp.iml:** It is always named according to the Flutter project's name that contains additional settings of the project. This file performs the internal tasks, which is managed by the Flutter SDK, so you do not need to edit the content manually at any time.

**pubspec.yaml:** It is the project's configuration file that will use a lot during working with the Flutter project. It allows you how your application works. This file contains:

* Project general settings such as name, description, and version of the project.
* Project dependencies.
* Project assets (e.g., images).

**pubspec.lock:** It is an auto-generated file based on the **.yaml** file. It holds more detail setup about all dependencies.

**README.md:** It is an auto-generated file that holds information about the project. We can edit this file if we want to share information with the developers.

Open the **main.dart** file and replace the code with the following code snippets.

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
5. **class** MyApp **extends** StatelessWidget {
6. // This widget is the root of your application.
7. @override
8. Widget build(BuildContext context) {
9. **return** MaterialApp(
10. title: 'Hello World Flutter Application',
11. theme: ThemeData(
12. // This is the theme of your application.
13. primarySwatch: Colors.blue,
14. ),
15. home: MyHomePage(title: 'Home page'),
16. );
17. }
18. }
19. **class** MyHomePage **extends** StatelessWidget {
20. MyHomePage({Key key, **this**.title}) : **super**(key: key);
21. // This widget is the home page of your application.
22. **final** String title;
24. @override
25. Widget build(BuildContext context) {
26. **return** Scaffold(
27. appBar: AppBar(
28. title: Text(**this**.title),
29. ),
30. body: Center(
31. child: Text('Hello World'),
32. ),
33. );
34. }
35. }

**Step 8:**Let us understand the above code snippet line by line.

* To start Flutter programming, you need first to import the Flutter package. Here, we have imported a **Material package**. This package allows you to create user interface according to the Material design guidelines specified by Android.
* The second line is an entry point of the Flutter applications similar to the main method in other programming languages. It calls the **runApp** function and pass it an object of **MyApp** The primary purpose of this function is to attach the given widget to the screen.
* Line 5 to 18 is a widget used for creating UI in the Flutter framework. Here, the **StatelessWidget** does not maintain any state of the widget. MyApp extends StatelessWidget that overrides its **build** The build method is used for creating a part of the UI of the application. In this block, the build method uses MaterialApp, a widget to create the root level UI of the application and contains three properties - title, theme, and home.
  1. **Title:** It is the title of the Flutter application.
  2. **Theme:** It is the theme of the widget. By default, it set the blue as the overall color of the application.
  3. **Home:** It is the inner UI of the application, which sets another widget (MyHomePage) for the application.
* Line 19 to 35, the **MyHomePage** is similar to MyApp, except it will return the **Scaffold** Scaffold widget is a top-level widget after the MaterialApp widget for creating the user interface. This widget contains two properties **appBar** and **body**. The appBar shows the header of the app, and body property shows the actual content of the application. Here, **AppBar** render the header of the application, **Center** widget is used to center the child widget, and **Text** is the final widget used to show the text content and displays in the center of the screen.

**Step 9:** Now, run the application. To do this, go to Run->Run main.dart, as shown in the below screen.

4.2 Flutter Widget:

4.2.1 Types of flutter widget:

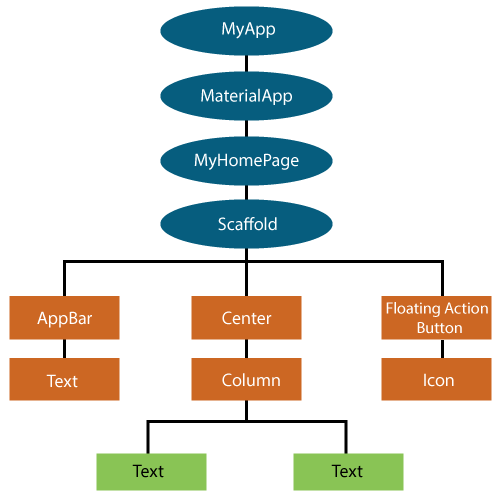
Flutter Widgets

In this section, we are going to learn the concept of a widget, how to create it, and their different types of widgets are available in the Flutter framework. We have learned earlier that everything in Flutter is a widget.

Whenever you are going to code for building anything in Flutter, it will be inside a widget. The central purpose is to build the app out of widgets. It describes how your app view should look like with their current configuration and state. When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.

Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.

The below image is a simple visual representation of the widget tree.



import 'package:flutter/material.dart';

class MyHomePage extends StatelessWidget {

MyHomePage({Key key, this.title}) : super(key: key);

// This widget is the home page of your application.

final String title;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(this.title),

),

body: Center(

child: Text('Hello World'),

),

);

}

}

4.2.1.1 Visible and Invisible

**Types of Widget**

We can split the Flutter widget into two categories:

1. Visible (Output and Input)
2. Invisible (Layout and Control)
   1. **Visible widget**

The visible widgets are related to the user input and output data. Some of the important types of this widget are:

**Text**

A Text widget holds some text to display on the screen. We can align the text widget by using textAlign property, and style property allow the customization of Text that includes font, font weight, font style, letter spacing, color, and many more. We can use it as like below code snippets.

new Text(

'Hello, Students!',

textAlign: TextAlign.center,

style: new TextStyle(fontWeight: FontWeight.bold),

)

**Button**

This widget allows you to perform some action on click. Flutter does not allow you to use the Button widget directly; instead, it uses a type of buttons like a FlatButton and a RaisedButton. We can use it as like below code snippets.

//FlatButton Example

new FlatButton(

child: Text("Click here"),

onPressed: () {

// Do something here

},

),

//RaisedButton Example

new RaisedButton(

child: Text("Click here"),

elevation: 5.0,

onPressed: () {

// Do something here

},

),

In the above example, the onPressed property allows us to perform an action when you click the button, and elevation property is used to change how much it stands out.

**Image**

This widget holds the image which can fetch it from multiple sources like from the asset folder or directly from the URL. It provides many constructors for loading image, which are given below:

Image: It is a generic image loader, which is used by ImageProvider.

asset: It load image from your project asset folder.

file: It loads images from the system folder.

memory: It load image from memory.

network: It loads images from the network.

To add an image in the project, you need first to create an assets folder where you keep your images and then add the below line in pubspec.yaml file.

assets:

- assets/

Now, add the following line in the dart file.

Image.asset('assets/computer.png')

The complete source code for adding an image is shown below in the hello world example.

class MyHomePage extends StatelessWidget {

MyHomePage({Key key, this.title}) : super(key: key);

// This widget is the home page of your application.

final String title;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text(this.title),

),

body: Center(

child: Image.asset('assets/computer.png'),

),

);

}

}

When you run the app, it will give the following output.



**Icon**

This widget acts as a container for storing the Icon in the Flutter. The following code explains it more clearly.

new Icon(

Icons.add,

size: 34.0,

)

* 1. **Invisible widget**

The invisible widgets are related to the layout and control of widgets. It provides controlling how the widgets actually behave and how they will look onto the screen. Some of the important types of these widgets are:

**Column**

A column widget is a type of widget that arranges all its children's widgets in a vertical alignment. It provides spacing between the widgets by using the mainAxisAlignment and crossAxisAlignment properties. In these properties, the main axis is the vertical axis, and the cross axis is the horizontal axis.

Example

The below code snippets construct two widget elements vertically.

new Column(

mainAxisAlignment: MainAxisAlignment.center,

children: <Widget>[

new Text(

"VegElement",

),

new Text(

"Non-vegElement"

),

],

),

**Row**

The row widget is similar to the column widget, but it constructs a widget horizontally rather than vertically. Here, the main axis is the horizontal axis, and the cross axis is the vertical axis.

Example

The below code snippets construct two widget elements horizontally.

new Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: <Widget>[

new Text(

"VegElement",

),

new Text(

"Non-vegElement"

),

],

),

**Center**

This widget is used to center the child widget, which comes inside it. All the previous examples contain inside the center widget.

Example

Center(

child: new clumn(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: <Widget>[

new Text(

"VegElement",

),

new Text(

"Non-vegElement"

),

],

),

)

**Padding**

This widget wraps other widgets to give them padding in specified directions. You can also provide padding in all directions. We can understand it from the below example that gives the text widget padding of 6.0 in all directions.

Example

Padding(

padding: const EdgeInsets.all(6.0),

child: new Text(

"Element 1",

),

),

**Scaffold**

This widget provides a framework that allows you to add common material design elements like AppBar, Floating Action Buttons, Drawers, etc.

**Stack**

It is an essential widget, which is mainly used for overlapping a widget, such as a button on a background gradient.

4.2.1.2 StatelessWidget, StatefulWidget

State Management Widget

In Flutter, there are mainly two types of widget:

* StatelessWidget
* StatefulWidget

**StatefulWidget**

A StatefulWidget has state information. It contains mainly two classes: the **state object** and the **widget**. It is dynamic because it can change the inner data during the widget lifetime. This widget does not have a **build()** method. It has **createState()** method, which returns a class that extends the Flutters State Class. The examples of the StatefulWidget are Checkbox, Radio, Slider, InkWell, Form, and TextField.

**Example**

1. **class** Car **extends** StatefulWidget {
2. **const** Car({ Key key, **this**.title }) : **super**(key: key);
4. @override
5. \_CarState createState() => \_CarState();
6. }
8. **class** \_CarState **extends** State<Car> {
9. @override
10. Widget build(BuildContext context) {
11. **return** Container(
12. color: **const** Color(0xFEEFE),
13. child: Container(
14. child: Container( //child: Container() )
15. )
16. );
17. }
18. }

**StatelessWidget**

The StatelessWidget does not have any state information. It remains static throughout its lifecycle. The examples of the StatelessWidget are Text, Row, Column, Container, etc.

**Example**

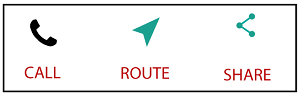
1. **class** MyStatelessCarWidget **extends** StatelessWidget {
2. **const** MyStatelessCarWidget ({ Key key }) : **super**(key: key);
4. @override
5. Widget build(BuildContext context) {
6. **return** Container(color: **const** Color(0x0xFEEFE));
7. }
8. }

4.2.1.3 Single child widget and Multiple child widget

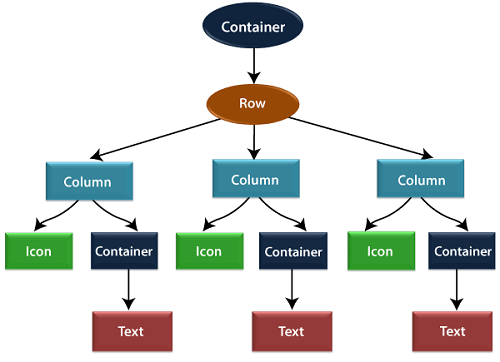
Flutter Layouts

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(
3. margin: **const** EdgeInsets.all(15.0),
4. color: Colors.blue,
5. width: 42.0,
6. height: 42.0,
7. ),
8. )

**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.

1. **const** Greetings(
2. child: Padding(
3. padding: EdgeInsets.all(14.0),
4. child: Text('Hello Students!'),
5. ),
6. )

**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.

1. Center(
2. child: Container(
3. height: 110.0,
4. width: 110.0,
5. color: Colors.blue,
6. child: Align(
7. alignment: Alignment.topLeft,
8. child: FlutterLogo(
9. size: 50,
10. ),
11. ),
12. ),
13. )

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

1. SizedBox(
2. width: 300.0,
3. height: 450.0,
4. child: **const** Card(child: Text('Hello JavaTpoint!')),
5. )

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

1. AspectRatio(
2. aspectRatio: 5/3,
3. child: Container(
4. color: Colors.bluel,
5. ),
6. ),

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

1. child: Baseline(
2. baseline: 30.0,
3. baselineType: TextBaseline.alphabetic,
4. child: Container(
5. height: 60,
6. width: 50,
7. color: Colors.blue,
8. ),
9. )

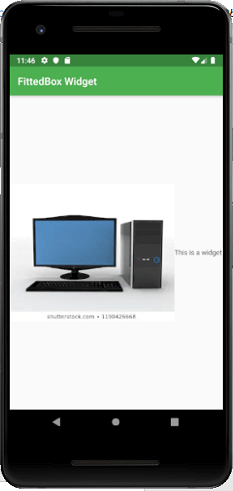
**ConstrainedBox:** It is a widget that allows you to force the additional constraints on its child widget. It means you can force the child widget to have a specific constraint without changing the properties of the child widget.

1. ConstrainedBox(
2. constraints: **new** BoxConstraints(
3. minHeight: 150.0,
4. minWidth: 150.0,
5. maxHeight: 300.0,
6. maxWidth: 300.0,
7. ),
8. child: **new** DecoratedBox(
9. decoration: **new** BoxDecoration(color: Colors.red),
10. ),
11. ),

**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**.

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
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(
13. // This is the theme of your application.
14. primarySwatch: Colors.green,
15. ),
16. home: MyHomePage(),
17. );
18. }
19. }
20. **class** MyHomePage **extends** StatelessWidget {
22. @override
23. Widget build(BuildContext context) {
24. **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. }



**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.

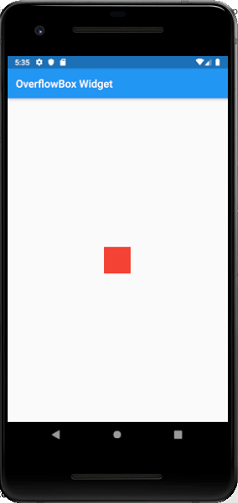
**Offstage:** It is used to measure the dimensions of a widget without bringing it on to the screen.

**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';
3. **void** main() => runApp(MyApp());
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 {
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**

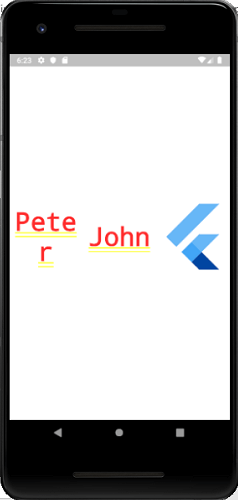
The multiple child widgets are a type of widget, which contains **more than one child widget**, and the layout of these widgets are **unique**. For example, Row widget laying out of its child widget in a horizontal direction, and Column widget laying out of its child widget in a vertical direction. If 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';
3. **void** main() => runApp(MyApp());
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(
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. @override
22. Widget build(BuildContext context) {
23. **return** Center(
24. child: Container(
25. alignment: Alignment.center,
26. color: Colors.white,
27. child: Row(
28. children: <Widget>[
29. Expanded(
30. child: Text('Peter', textAlign: TextAlign.center),
31. ),
32. Expanded(
33. child: Text('John', textAlign: TextAlign.center ),
35. ),
36. Expanded(
37. child: FittedBox(
38. fit: BoxFit.contain, // otherwise the logo will be tiny
39. child: **const** FlutterLogo(),
40. ),
41. ),
42. ],
43. ),
44. ),
45. );
46. }
47. }



**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.

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
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(
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. @override
22. Widget build(BuildContext context) {
23. **return** Center(
24. child: Container(
25. alignment: Alignment.center,
26. color: Colors.white,
27. child: Stack(
28. children: <Widget>[
29. // Max Size
30. Container(
31. color: Colors.blue,
32. ),
33. Container(
34. color: Colors.pink,
35. height: 400.0,
36. width: 300.0,
37. ),
38. Container(
39. color: Colors.yellow,
40. height: 220.0,
41. width: 200.0,
42. )
43. ],
44. ),
45. ),
46. );
47. }
48. }



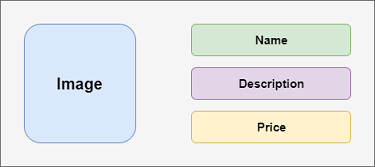
Building Complex Layout

In this section, we are going to learn how you can create a complex user interface using both single and multiple child layout widgets. The layout framework allows you to create a complex user interface layout by nesting the rows and columns inside of rows and columns.

Let us see an example of a complex user interface by creating the **product list**. For this purpose, you need first to replace the code of **main.dart** file with the following code snippet.

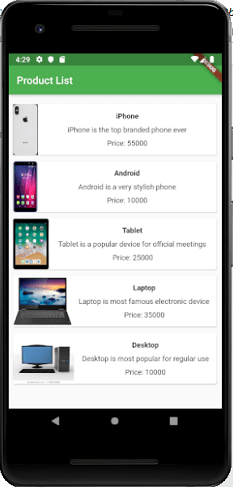
1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
5. **class** MyApp **extends** StatelessWidget {
6. // It is the root widget of your application.
7. @override
8. Widget build(BuildContext context) {
9. **return** MaterialApp(
10. title: 'Flutter Demo Application', theme: ThemeData(
11. primarySwatch: Colors.green,),
12. home: MyHomePage(title: 'Complex layout example'),
13. );
14. }
15. }
16. **class** MyHomePage **extends** StatelessWidget {
17. MyHomePage({Key key, **this**.title}) : **super**(key: key);
18. **final** String title;
20. @override
21. Widget build(BuildContext context) {
22. **return** Scaffold(
23. appBar: AppBar(title: Text("Product List")),
24. body: ListView(
25. padding: **const** EdgeInsets.fromLTRB(3.0, 12.0, 3.0, 12.0),
26. children: <Widget>[
27. ProductBox(
28. name: "iPhone",
29. description: "iPhone is the top branded phone ever",
30. price: 55000,
31. image: "iphone.png"
32. ),
33. ProductBox(
34. name: "Android",
35. description: "Android is a very stylish phone",
36. price: 10000,
37. image: "android.png"
38. ),
39. ProductBox(
40. name: "Tablet",
41. description: "Tablet is a popular device for official meetings",
42. price: 25000,
43. image: "tablet.png"
44. ),
45. ProductBox(
46. name: "Laptop",
47. description: "Laptop is most famous electronic device",
48. price: 35000,
49. image: "laptop.png"
50. ),
51. ProductBox(
52. name: "Desktop",
53. description: "Desktop is most popular for regular use",
54. price: 10000,
55. image: "computer.png"
56. ),
57. ],
58. )
59. );
60. }
61. }
62. **class** ProductBox **extends** StatelessWidget {
63. ProductBox({Key key, **this**.name, **this**.description, **this**.price, **this**.image}) :
64. **super**(key: key);
65. **final** String name;
66. **final** String description;
67. **final** **int** price;
68. **final** String image;
70. Widget build(BuildContext context) {
71. **return** Container(
72. padding: EdgeInsets.all(2),
73. height: 110,
74. child: Card(
75. child: Row(
76. mainAxisAlignment: MainAxisAlignment.spaceEvenly,
77. children: <Widget>[
78. Image.asset("assets/" + image),
79. Expanded(
80. child: Container(
81. padding: EdgeInsets.all(5),
82. child: Column(
83. mainAxisAlignment: MainAxisAlignment.spaceEvenly,
84. children: <Widget>[
85. Text(
86. **this**.name, style: TextStyle(
87. fontWeight: FontWeight.bold
88. )
89. ),
90. Text(**this**.description), Text(
91. "Price: " + **this**.price.toString()
92. ),
93. ],
94. )
95. )
96. )
97. ]
98. )
99. )
100. );
101. }
102. }

In the above code, we create widget **ProductBox** that contains the details of the product, such as image, name, price, and description. In the ProductBox widget, we use the following child widgets: Container, Row, Column, Expanded, Card, Text, Image, etc. This widget contains the following layout:



**Output**

Now, when we run the dart file in the android emulator, it will give the following output.



4.2.2 Visible widget(Constructor and Properties): Text, Image, Button, Icon

**Flutter Text**

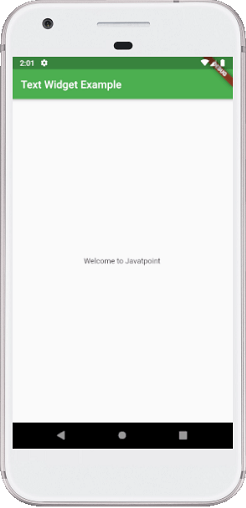
A Text is a widget in Flutter that allows us to **display a string of text with a single line in our application**. Depending on the layout constraints, we can break the string across multiple lines or might all be displayed on the same line. If we do not specify any styling to the text widget, it will use the closest **DefaultTextStyle** class style. This class does not have any explicit style. In this article, we are going to learn how to use a Text widget and how to style it in our application.

Here is a simple example to understand this widget. This example shows our **project's title** in the application bar and a **message** in the application's body.

1. **import** 'package:flutter/material.dart';
3. **void** main() { runApp(MyApp()); }
5. **class** MyApp **extends** StatelessWidget {
6. @override
7. Widget build(BuildContext context) {
8. **return** MaterialApp(
9. theme: ThemeData(
10. primarySwatch: Colors.green,
11. ),
12. home: MyTextPage()
13. );
14. }
15. }
16. **class** MyTextPage **extends** StatelessWidget {
17. @override
18. Widget build(BuildContext context) {
19. **return** Scaffold(
20. appBar: AppBar(
21. title:Text("Text Widget Example")
22. ),
23. body: Center(
24. child:Text("Welcome to Javatpoint")
25. ),
26. );
27. }
28. }

In the above code, we have used a **MaterialApp** widget that calls the home screen using the **MyTextPage()** class. This class contains the **scaffold** widget, which has **appBar** and **body** where we have used the **Text** widget to display the title and body, respectively. It is a simple scenario of Text widget where we have to pass the string that we want to display on our page.

When we run this application in the emulator or device, we should get the UI similar to the below screenshot:



**Text Widget Constructor:**

The text widget constructor used to make the custom look and feel of our text in Flutter:

1. **const** Text(String data,{
2. Key key,
3. TextStyle style,
4. StrutStyle strutStyle,
5. TextAlign textAlign,
6. TextDirection textDirection,
7. TextOverflow overflow,
8. bool softWrap,
9. **double** textScaleFactor,
10. **int** maxLines,
11. String semanticsLabel,
12. TextWidthBasis textWidthBasis,
13. TextHeightBehavior textHeightBehavior
14. }
15. )

The following are the essential properties of the Text widget used in our application:

**TextAlign:** It is used to specify how our text is aligned horizontally. It also controls the text location.

**TextDirection:** It is used to determine how textAlign values control the layout of our text. Usually, we write text from left to right, but we can change it using this parameter.

**Overflow:** It is used to determine when the text will not fit in the available space. It means we have specified more text than the available space.

**TextScaleFactor:** It is used to determine the scaling to the text displayed by the Text widget. Suppose we have specified the text scale factor as 1.5, then our text will be 50 percent larger than the specified font size.

**SoftWrap:** It is used to determine whether or not to show all text widget content when there is not enough space available. If it is true, it will show all content. Otherwise, it will not show all content.

**MaxLines:** It is used to determine the maximum number of lines displayed in the text widget.

**TextWidthBasis:** It is used to control how the text width is defined.

**TextHeightBehavior:** It is used to control how the paragraph appears between the first line and descent of the last line.

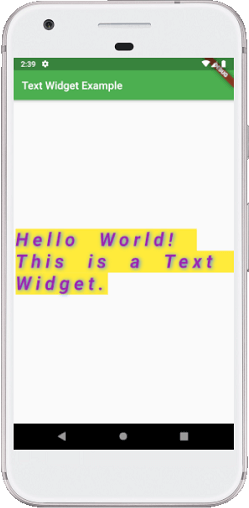
**Style:** It is the most common property of this widget that allows developers to styling their text. It can do styling by specifying the foreground and background color, font size, font weight, letter and word spacing, locale, shadows, etc. See the table to understand it more easily:

|  |  |
| --- | --- |
| **Attributes** | **Descriptions** |
| foreground | It determines the paint as a foreground for the text. |
| background | It determines the paint as a background for the text. |
| fontWeight | It determines the thickness of the text. |
| fontSize | It determines the size of the text. |
| fontFamily | It is used to specify the typeface for the font. For this, we need to download a typeface file in our project, and then keep this file into the assets/font folder. Finally, config the pubspec.yaml file to use it in the project. |
| fontStyle | It is used to style the font either in bold or italic form. |
| Color | It is used to determine the color of the text. |
| letterSpacing | It is used to determine the distance between the characters of the text. |
| wordSpacing | It is used to specify the distance between two words of the text. |
| shadows | It is used to paint underneath the text. |
| decoration | We use this to decorate text using the three parameters: decoration, decorationColor, decorationStyle. The decoration determines the location, decorationColor specify the color, decorationStyle determine the shape. |

1. **import** 'package:flutter/material.dart';
3. **void** main() { runApp(MyApp()); }
5. **class** MyApp **extends** StatelessWidget {
6. @override
7. Widget build(BuildContext context) {
8. **return** MaterialApp(
9. theme: ThemeData(
10. primarySwatch: Colors.green,
11. ),
12. home: MyTextPage()
13. );
14. }
15. }
16. **class** MyTextPage **extends** StatelessWidget {
17. @override
18. Widget build(BuildContext context) {
19. **return** Scaffold(
20. appBar: AppBar(
21. title:Text("Text Widget Example")
22. ),
23. body: Center(
24. child:Text(
25. "Hello World! This is a Text Widget.",
26. style: TextStyle(
27. fontSize: 35,
28. color: Colors.purple,
29. fontWeight: FontWeight.w700,
30. fontStyle: FontStyle.italic,
31. letterSpacing: 8,
32. wordSpacing: 20,
33. backgroundColor: Colors.yellow,
34. shadows: [
35. Shadow(color: Colors.blueAccent, offset: Offset(2,1), blurRadius:10)
36. ]
37. ),
38. )
39. ),
40. );
41. }
42. }

**Output:**

When we run this application in the emulator or device, we should get the UI similar to the below screenshot:



**Flutter Images**

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.

Displaying images is the fundamental concept of most of the mobile apps. Flutter has an Image widget that allows displaying different types of images in the mobile application.

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:

1. assets:
2. - assets/tablet.png
3. - assets/background.png

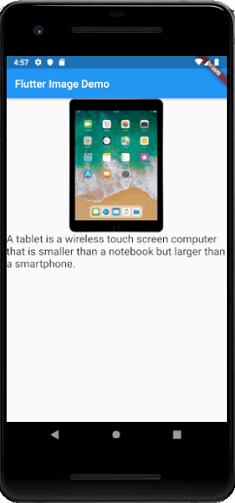
If the assets folder contains more than one image, we can include it by specifying the directory name with the **slash (/)** character at the end.

1. flutter:
2. assets:
3. - assets/

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

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
5. **class** MyApp **extends** StatelessWidget {
6. @override
7. Widget build(BuildContext context) {
8. **return** MaterialApp(
9. home: Scaffold(
10. appBar: AppBar(
11. title: Text('Flutter Image Demo'),
12. ),
13. body: Center(
14. child: Column(
15. children: <Widget>[
16. Image.asset('assets/tablet.png'),
17. Text(
18. 'A tablet is a wireless touch screen computer that is smaller than a notebook but larger than a smartphone.',
19. style: TextStyle(fontSize: 20.0),
20. )
21. ],
22. ),
23. ),
24. ),
25. );
26. }
27. }

**Step 5:** Now, run the app. You will get something like the screen below.



**Display images from the internet**

Displaying images from the internet or network is very simple. Flutter provides a built-in method **Image.network** to work with images from a URL. The Image.network method also allows you to use some optional properties, such as height, width, color, fit, and many more. We can use the following syntax to display an image from the internet.

1. Image.network(
2. 'https://picsum.photos/250?image=9',
3. )

The Imag.Network gives one useful thing that supports animated gifs. We can use the following syntax for displaying gifs from the internet.

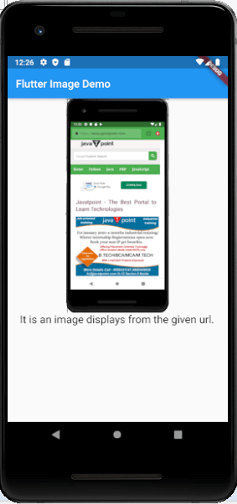
1. Image.network(
2. 'https://github.com/flutter/plugins/raw/master/packages/video\_player/doc/demo\_ipod.gif?raw=true',
3. );

Let us understand how to display an image from the network with the following example:

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
5. **class** MyApp **extends** StatelessWidget {
6. @override
7. Widget build(BuildContext context) {
8. **return** MaterialApp(
9. home: Scaffold(
10. appBar: AppBar(
11. title: Text('Flutter Image Demo'),
12. ),
13. body: Center(
14. child: Column(
15. children: <Widget>[
16. Image.network(
17. 'https://static.javatpoint.com/tutorial/flutter/images/flutter-creating-android-platform-specific-code3.png',
18. height: 400,
19. width: 250
20. ),
21. Text(
22. 'It is an image displays from the given url.',
23. style: TextStyle(fontSize: 20.0),
24. )
25. ],
26. ),
27. ),
28. ),
29. );
30. }
31. }

**Output**

When you run the app in Android Emulator, the following screen appears. Here, you can see the image of the given url.



**Flutter Buttons**

Buttons are the graphical control element that **provides a user to trigger an event** such as taking actions, making choices, searching things, and many more. They can be placed anywhere in our UI like dialogs, forms, cards, toolbars, etc.

Buttons are the Flutter widgets, which is a part of the material design library. Flutter provides several types of buttons that have different shapes, styles, and features.

Features of Buttons

The standard features of a button in Flutter are given below:

1. We can easily apply themes on buttons, shapes, color, animation, and behavior.
2. We can also theme icons and text inside the button.
3. Buttons can be composed of different child widgets for different characteristics.

Types of Flutter Buttons

Following are the different types of button available in [Flutter](https://www.javatpoint.com/flutter):

* Flat Button
* Raised Button
* Floating Button
* Drop Down Button
* Icon Button
* Inkwell Button
* PopupMenu Button
* Outline Button

Let us discuss each button in detail.

1. Flat Button

It is a **text label button** that does not have much decoration and displayed **without any elevation**. The flat button has two required properties that are: **child and onPressed()**. It is mostly used in toolbars, dialogs, or inline with other content. By default, the flat button has no color, and its text is black. But, we can use color to the button and text using **color and textColor** attributes, respectively.

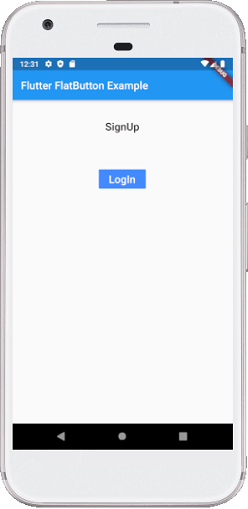
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MyApp());
5. }
7. class MyApp extends StatefulWidget {
8. @override
9. \_MyAppState createState() =**>** \_MyAppState();
10. }
12. class \_MyAppState extends State**<MyApp>** {
13. @override
14. Widget build(BuildContext context) {
15. return MaterialApp(
16. home: Scaffold(
17. appBar: AppBar(
18. title: Text('Flutter FlatButton Example'),
19. ),
20. body: Center(child: Column(children: **<Widget>**[
21. Container(
22. margin: EdgeInsets.all(25),
23. child: FlatButton(
24. child: Text('SignUp', style: TextStyle(fontSize: 20.0),),
25. onPressed: () {},
26. ),
27. ),
28. Container(
29. margin: EdgeInsets.all(25),
30. child: FlatButton(
31. child: Text('LogIn', style: TextStyle(fontSize: 20.0),),
32. color: Colors.blueAccent,
33. textColor: Colors.white,
34. onPressed: () {},
35. ),
36. ),
37. ]
38. ))
39. ),
40. );
41. }
42. }

**Output:**

If we run this app, we will see the following screen:



2. Raised Button

It is a button, which is based on the material widget and has a **rectangular body**. It is similar to a flat button, but it **has an elevation** that will increases when the button is pressed. It adds dimension to the UI along Z-axis. It has several properties like text color, shape, padding, button color, the color of a button when disabled, animation time, elevation, etc.

This button has **two callback functions**.

**onPressed():** It is triggered when the button is pressed.

**onLongPress():** It is triggered when the button is long pressed.

It is to note that this button is in a **disabled state** if onPressed() and onLongPressed() callbacks are not specified.

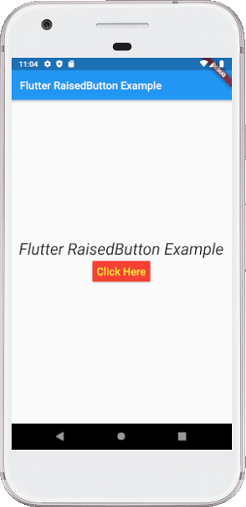
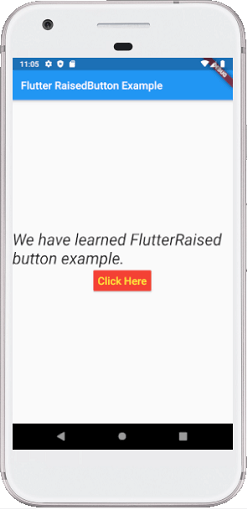
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MyApp());
5. }
7. class MyApp extends StatefulWidget {
8. @override
9. \_MyAppState createState() =**>** \_MyAppState();
10. }
12. class \_MyAppState extends State**<MyApp>** {
13. String msg = 'Flutter RaisedButton Example';
14. @override
15. Widget build(BuildContext context) {
16. return MaterialApp(
17. home: Scaffold(
18. appBar: AppBar(
19. title: Text('Flutter RaisedButton Example'),
20. ),
21. body: Container(
22. child: Center(
23. child: Column(
24. mainAxisAlignment: MainAxisAlignment.center,
25. children: [
26. Text(msg, style: TextStyle(fontSize: 30, fontStyle: FontStyle.italic),),
27. RaisedButton(
28. child: Text("Click Here", style: TextStyle(fontSize: 20),),
29. onPressed: \_changeText,
30. color: Colors.red,
31. textColor: Colors.yellow,
32. padding: EdgeInsets.all(8.0),
33. splashColor: Colors.grey,
34. )
35. ],
36. ),
37. ),
38. ),
39. ),
40. );
41. }
42. \_changeText() {
43. setState(() {
44. if (msg.startsWith('F')) {
45. msg = 'We have learned FlutterRaised button example.';
46. } else {
47. msg = 'Flutter RaisedButton Example';
48. }
49. });
50. }
51. }

**Output:**

When we run this example, it will give the below screenshot. If we click on the "**Click Here**" button, it will change the text message. Show the second screenshot.

3. Floating Action Button

A FAB button is a **circular icon button** that triggers the primary action in our application. It is the most used button in today's applications. We can use this button for adding, refreshing, or sharing the content. Flutter suggests using at most one FAB button per screen. There are two types of Floating Action Button:

**FloatingActionButton:** It creates a simple circular floating button with a child widget inside it. It must have a child parameter to display a widget.

**FloatingActionButton.extended:** It creates a wide floating button along with an icon and a label inside it. Instead of a child, it uses labels and icon parameters.

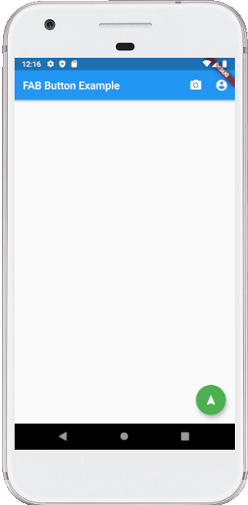
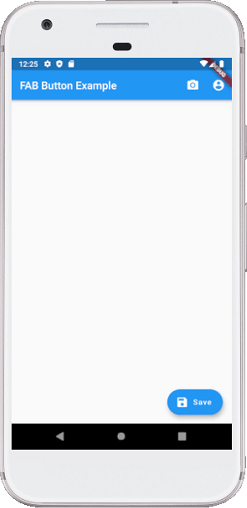
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MyApp());
5. }
7. class MyApp extends StatefulWidget {
8. @override
9. \_MyAppState createState() =**>** \_MyAppState();
10. }
12. class \_MyAppState extends State**<MyApp>** {
13. @override
14. Widget build(BuildContext context) {
15. return MaterialApp(home: Scaffold(
16. appBar: AppBar(
17. title: Text("FAB Button Example"),
18. backgroundColor: Colors.blue,
19. actions: **<Widget>**[
20. IconButton(icon: Icon(Icons.camera\_alt), onPressed: () =**>** {}),
21. IconButton(icon: Icon(Icons.account\_circle), onPressed: () =**>** {})
22. ],
23. ),
24. floatingActionButton: FloatingActionButton(
25. child: Icon(Icons.navigation),
26. backgroundColor: Colors.green,
27. foregroundColor: Colors.white,
28. onPressed: () =**>** {},
29. ),
30. /\*floatingActionButton:FloatingActionButton.extended(
31. onPressed: () {},
32. icon: Icon(Icons.save),
33. label: Text("Save"),
34. ), \*/
35. ),
36. );
37. }
38. }

**Output:**

Run the application in android emulator, and it will give the UI similar to the following screenshot. The second image is an output of the **FAB.extended** button. Its coding can be seen in the above code's comment section.

4. DropDown Button

A drop-down button is used to create a nice overlay on the screen that allows the user to select any item from multiple options. Flutter allows a simple way to implement a drop-down box or drop-down button. This button shows the currently selected item and an arrow that opens a menu to select an item from multiple options.

Flutter provides a **DropdownButton widget** to implement a drop-down list. We can place it anywhere in our app.

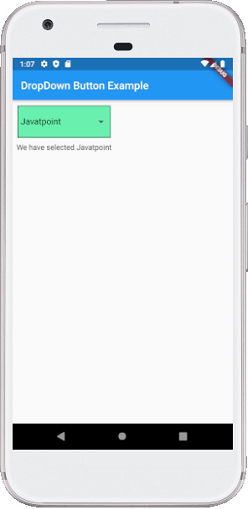
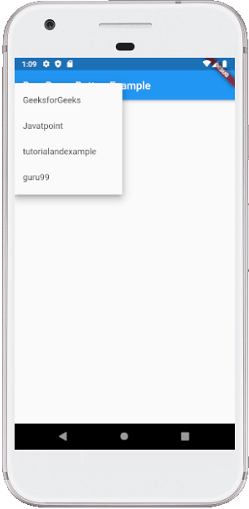
**Example**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() =**>** runApp(MaterialApp(
4. home: MyApp(),
5. ));
7. class MyApp extends StatefulWidget {
8. @override
9. \_MyAppState createState() =**>** \_MyAppState();
10. }
12. class \_MyAppState extends State**<MyApp>** {
13. List**<ListItem>** \_dropdownItems = [
14. ListItem(1, "GeeksforGeeks"),
15. ListItem(2, "Javatpoint"),
16. ListItem(3, "tutorialandexample"),
17. ListItem(4, "guru99")
18. ];
20. List**<DropdownMenuItem<ListItem>>** \_dropdownMenuItems;
21. ListItem \_itemSelected;
23. void initState() {
24. super.initState();
25. \_dropdownMenuItems = buildDropDownMenuItems(\_dropdownItems);
26. \_itemSelected = \_dropdownMenuItems[1].value;
27. }
29. List**<DropdownMenuItem<ListItem>>** buildDropDownMenuItems(List listItems) {
30. List**<DropdownMenuItem<ListItem>>** items = List();
31. for (ListItem listItem in listItems) {
32. items.add(
33. DropdownMenuItem(
34. child: Text(listItem.name),
35. value: listItem,
36. ),
37. );
38. }
39. return items;
40. }
42. @override
43. Widget build(BuildContext context) {
44. return Scaffold(
45. appBar: AppBar(
46. title: Text("DropDown Button Example"),
47. ),
48. body: Column(
49. children: **<Widget>**[
50. Padding(
51. padding: const EdgeInsets.all(10.0),
52. child: Container(
53. padding: const EdgeInsets.all(5.0),
54. decoration: BoxDecoration(
55. color: Colors.greenAccent,
56. border: Border.all()),
57. child: DropdownButtonHideUnderline(
58. child: DropdownButton(
59. value: \_itemSelected,
60. items: \_dropdownMenuItems,
61. onChanged: (value) {
62. setState(() {
63. \_itemSelected = value;
64. });
65. }),
66. ),
67. ),
68. ),
69. Text("We have selected ${\_itemSelected.name}"),
70. ],
71. ),
72. );
73. }
74. }
76. class ListItem {
77. int value;
78. String name;
80. ListItem(this.value, this.name);
81. }

**Output**

Run the application in android emulator, and it will give the UI similar to the following screenshot. The second image is an output of the list contains in the drop drown button.

5. Icon Button

An IconButton is a **picture printed** on the Material widget. It is a useful widget that gives the Flutter UI a material design feel. We can also customize the look and feel of this button. In simple terms, it is an icon that reacts when the user will touch it.

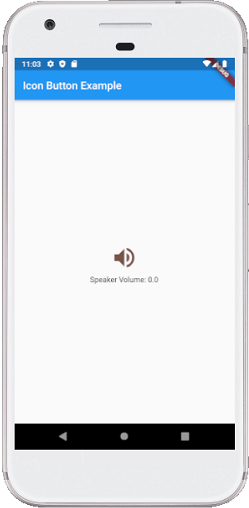
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() =**>** runApp(MyApp());
5. class MyApp extends StatelessWidget {
6. @override
7. Widget build(BuildContext context) {
8. return MaterialApp(
9. home: Scaffold(
10. appBar: AppBar(
11. title: Text("Icon Button Example"),
12. ),
13. body: Center(
14. child: MyStatefulWidget(),
15. ),
16. ),
17. );
18. }
19. }
20. double \_speakervolume = 0.0;
22. class MyStatefulWidget extends StatefulWidget {
23. MyStatefulWidget({Key key}) : super(key: key);
25. @override
26. \_MyStatefulWidgetState createState() =**>** \_MyStatefulWidgetState();
27. }
29. class \_MyStatefulWidgetState extends State**<MyStatefulWidget>** {
30. Widget build(BuildContext context) {
31. return Column(
32. mainAxisSize: MainAxisSize.min,
33. children: **<Widget>**[
34. IconButton(
35. icon: Icon(Icons.volume\_up),
36. iconSize: 50,
37. color: Colors.brown,
38. tooltip: 'Increase volume by 5',
39. onPressed: () {
40. setState(() {
41. \_speakervolume += 5;
42. });
43. },
44. ),
45. Text('Speaker Volume: $\_speakervolume')
46. ],
47. );
48. }
49. }

**Output:**

Run the application in android emulator, and it will give the UI similar to the following screenshot. When we press the **volume button**, it will always increase by 5.



6. Inkwell Button

InkWell button is a material design concept, which is used for **touch response**. This widget comes under the Material widget where the ink reactions are actually painted. It creates the app UI interactive by adding gesture feedback. It is mainly used for adding **splash ripple effect**.

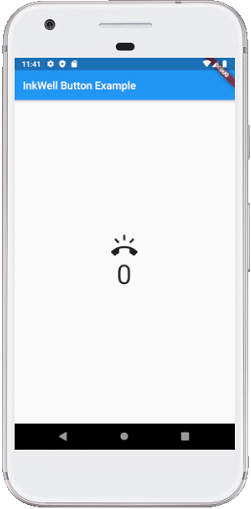
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() =**>** runApp(MyApp());
5. class MyApp extends StatefulWidget {
6. @override
7. \_MyAppState createState() =**>** \_MyAppState();
8. }
10. class \_MyAppState extends State**<MyApp>** {
11. int \_volume = 0;
13. @override
14. Widget build(BuildContext context) {
15. return MaterialApp(
16. home: Scaffold(
17. appBar: AppBar(
18. title: Text('InkWell Button Example'),
19. ),
20. body: Center(
21. child: new Column(
22. mainAxisAlignment: MainAxisAlignment.center,
23. children: **<Widget>**[
24. InkWell(
25. splashColor: Colors.green,
26. highlightColor: Colors.blue,
27. child: Icon(Icons.ring\_volume, size: 50),
28. onTap: () {
29. setState(() {
30. \_volume += 2;
31. });
32. },
33. ),
34. Text (
35. \_volume.toString(),
36. style: TextStyle(fontSize: 50)
37. ),
38. ],
39. ),
40. ),
41. ),
42. );
43. }
44. }

**Output:**

Run the application in android emulator, and it will give the UI similar to the following screenshot. Every time we press the ring volume button, it will increase the volume by 2.



7. PopupMenu Button

It is a button that **displays the menu** when it is pressed and then calls the **onSelected** method the menu is dismissed. It is because the item from the multiple options is selected. This button contains a text and an image. It will mainly use with **Settings** menu to list all options. It helps in making a great user experience.

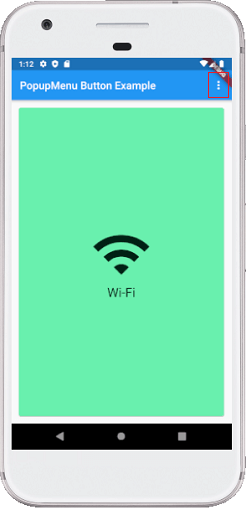
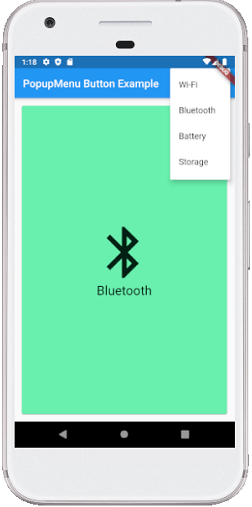
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() { runApp(MyApp());}
5. class MyApp extends StatefulWidget {
6. @override
7. \_MyAppState createState() =**>** \_MyAppState();
8. }
10. class \_MyAppState extends State**<MyApp>** {
11. Choice \_selectedOption = choices[0];
13. void \_select(Choice choice) {
14. setState(() {
15. \_selectedOption = choice;
16. });
17. }
18. @override
19. Widget build(BuildContext context) {
20. return MaterialApp(
21. home: Scaffold(
22. appBar: AppBar(
23. title: const Text('PopupMenu Button Example'),
24. actions: **<Widget>**[
25. PopupMenuButton**<Choice>**(
26. onSelected: \_select,
27. itemBuilder: (BuildContext context) {
28. return choices.skip(0).map((Choice choice) {
29. return PopupMenuItem**<Choice>**(
30. value: choice,
31. child: Text(choice.name),
32. );
33. }).toList();
34. },
35. ),
36. ],
37. ),
38. body: Padding(
39. padding: const EdgeInsets.all(10.0),
40. child: ChoiceCard(choice: \_selectedOption),
41. ),
42. ),
43. );
44. }
45. }
47. class Choice {
48. const Choice({this.name, this.icon});
49. final String name;
50. final IconData icon;
51. }
53. const List**<Choice>** choices = const **<Choice>**[
54. const Choice(name: 'Wi-Fi', icon: Icons.wifi),
55. const Choice(name: 'Bluetooth', icon: Icons.bluetooth),
56. const Choice(name: 'Battery', icon: Icons.battery\_alert),
57. const Choice(name: 'Storage', icon: Icons.storage),
58. ];
60. class ChoiceCard extends StatelessWidget {
61. const ChoiceCard({Key key, this.choice}) : super(key: key);
63. final Choice choice;
65. @override
66. Widget build(BuildContext context) {
67. final TextStyle textStyle = Theme.of(context).textTheme.headline;
68. return Card(
69. color: Colors.greenAccent,
70. child: Center(
71. child: Column(
72. mainAxisSize: MainAxisSize.min,
73. crossAxisAlignment: CrossAxisAlignment.center,
74. children: **<Widget>**[
75. Icon(choice.icon, size: 115.0, color: textStyle.color),
76. Text(choice.name, style: textStyle),
77. ],
78. ),
79. ),
80. );
81. }
82. }

**Output:**

Run the application in android emulator, and it will give the UI similar to the following screenshot. When we click the **three dots** shown at the top left corner of the screen, it will pop up the multiple options. Here, we can select any option, and it will keep it in the card, as shown in the second image.

8. Outline Button

It is similar to the flat button, but it contains a thin grey rounded rectangle border. Its outline border is defined by the shape attribute.

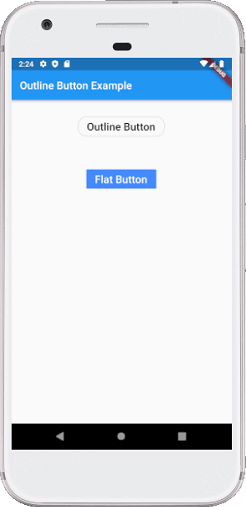
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MyApp());
5. }
7. class MyApp extends StatefulWidget {
8. @override
9. \_MyAppState createState() =**>** \_MyAppState();
10. }
12. class \_MyAppState extends State**<MyApp>** {
13. @override
14. Widget build(BuildContext context) {
15. return MaterialApp(
16. home: Scaffold(
17. appBar: AppBar(
18. title: Text('Outline Button Example'),
19. ),
20. body: Center(child: Column(children: **<Widget>**[
21. Container(
22. margin: EdgeInsets.all(25),
23. child: OutlineButton(
24. child: Text("Outline Button", style: TextStyle(fontSize: 20.0),),
25. highlightedBorderColor: Colors.red,
26. shape: RoundedRectangleBorder(
27. borderRadius: BorderRadius.circular(15)),
28. onPressed: () {},
29. ),
30. ),
31. Container(
32. margin: EdgeInsets.all(25),
33. child: FlatButton(
34. child: Text('Flat Button', style: TextStyle(fontSize: 20.0),),
35. color: Colors.blueAccent,
36. textColor: Colors.white,
37. onPressed: () {},
38. ),
39. ),
40. ]
41. ))
42. ),
43. );
44. }
45. }

**Output:**

Run the application in android emulator, and it will give the UI similar to the following screenshot.



Button Bar

Flutter provides the flexibility to **arrange the buttons in a bar or a row**. ButtonBar widget contains three properties: **alignment, children, and mainAxisSize**.

* Alignment is used to present the aligning option to the entire button bar widget.
* Children attribute is used to take the number of buttons in a bar.
* mainAxisSize attribute is used to provide the horizontal space for the button bar.

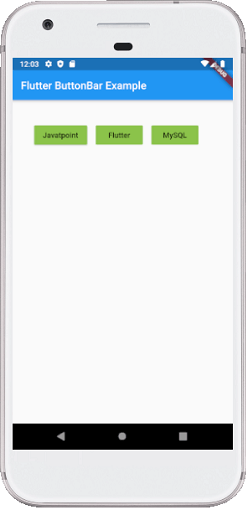
**Example:**

Open the **main.dart** file and replace it with the below code.

1. import 'package:flutter/material.dart';
3. void main() {
4. runApp(MaterialApp( home: MyApp(),));
5. }
7. class MyApp extends StatefulWidget {
8. @override
9. \_State createState() =**>** \_State();
10. }
12. class \_State extends State**<MyApp>** {
13. @override
14. Widget build(BuildContext context) {
15. return Scaffold(
16. appBar: AppBar(
17. title: Text('Flutter ButtonBar Example'),
18. ),
19. body: Padding(
20. padding: EdgeInsets.all(10),
21. child: Column(
22. children: **<Widget>**[
23. Padding(
24. padding: EdgeInsets.all(15),
25. child: new ButtonBar(
26. mainAxisSize: MainAxisSize.min,
27. children: **<Widget>**[
28. RaisedButton(
29. child: new Text('Javatpoint'),
30. color: Colors.lightGreen,
31. onPressed: () {/\*\* \*/},
32. ),
33. FlatButton(
34. child: Text('Flutter'),
35. color: Colors.lightGreen,
36. onPressed: () {/\*\* \*/},
37. ),
38. FlatButton(
39. child: Text('MySQL'),
40. color: Colors.lightGreen,
41. onPressed: () {/\*\* \*/},
42. ),
43. ],
44. ),
45. ),
46. ],
47. )
48. )
49. );
50. }
51. }

**Output:**

Run the application in android emulator, and it will give the UI similar to the following screenshot. Here, we can see that the three buttons are placed in a horizontal bar or row.



**Flutter Icons**

An icon is a **graphic image** representing an application or any specific entity containing meaning for the user. It can be selectable and non-selectable. **For example**, the company's logo is non-selectable. Sometimes it also contains a **hyperlink** to go to another page. It also acts as a sign in place of a detailed explanation of the actual entity.

[Flutter](https://www.javatpoint.com/flutter) provides an **Icon Widget** to create icons in our applications. We can create icons in Flutter, either using inbuilt icons or with the custom icons. Flutter provides the list of all icons in the **Icons class**. In this article, we are going to learn how to use Flutter icons in the application.

Icon Widget Properties

Flutter icons widget has different properties for customizing the icons. These properties are explained below:

|  |  |
| --- | --- |
| **Property** | **Descriptions** |
| icon | It is used to specify the icon name to display in the application. Generally, Flutter uses material design icons that are symbols for common actions and items. |
| color | It is used to specify the color of the icon. |
| size | It is used to specify the size of the icon in pixels. Usually, icons have equal height and width. |
| textDirection | It is used to specify to which direction the icon will be rendered. |

Let us understand Flutter icons using different examples.

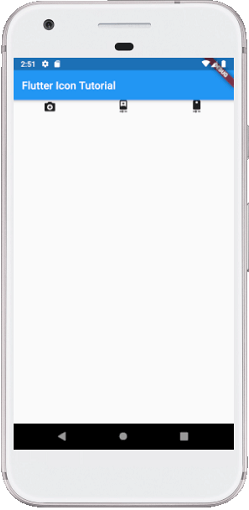
**Example 1:**

In this example, we will see the basic icon widget that has default values. First, create a project in the IDE, navigate to the **lib folder**, and then open the **main.dart** file. Now, replace the below code in the main.dart file:

1. **import** 'package:flutter/material.dart';
3. **void** main() => runApp(MyApp());
5. **class** MyApp **extends** StatelessWidget {
6. // This widget is the root of your application.
7. @override
8. Widget build(BuildContext context) {
9. **return** MaterialApp(
10. theme: ThemeData(
11. primarySwatch: Colors.blue,
12. ),
13. home: MyIconPage(),
14. );
15. }
16. }
18. **class** MyIconPage **extends** StatefulWidget {
19. @override
20. \_MyIconPageState createState() => \_MyIconPageState();
21. }
23. **class** \_MyIconPageState **extends** State<MyIconPage> {
24. @override
25. Widget build(BuildContext context) {
26. **return** Scaffold(
27. appBar: AppBar(
28. title: Text('Flutter Icon Tutorial'),
29. ),
30. body: Row(
31. mainAxisAlignment: MainAxisAlignment.spaceAround,
32. children: <Widget>[
33. Icon(Icons.camera\_enhance),
34. Icon(Icons.camera\_front),
35. Icon(Icons.camera\_rear),
36. ]),
37. );
38. }
39. }

**Output:**

When we run this project, it will show the UI similar to the following screenshot in the emulator or device we are using:



4.3.3 Invisible widget(Constructor and Properties): column, row, center, padding, scaffold, stack