Flutter

Step 01: Import Statements

Import statements are to written first in the top of the file, whenever we make a new dart file, we add the import statements firstly.

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

OR

Whenever we add a dependency from pub.dev to our pubspec.yaml, we have to import that package's import statement, in the file where we want to use the dependency.

Step 02: void main()

```
void main()
{
  runApp(const MaterialApp(home: MyApp()));
}
```

main() is the main function which runs firstly in the app. void is the datatype of the main() function. After import statements we write the main() function.

Step 03: runApp() -> inside main() function

Inside main() function we write the first function runApp(). runApp() function is responsible for rendering the app widgets.

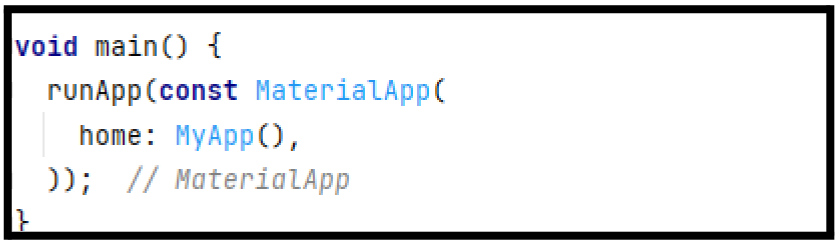
Step 04: MaterialApp() Or CupertinoApp() -> inside runApp()

Inside runApp(), you need to add MaterialApp() or CupertinoApp(), only one time, as this introduces the navigator and theme widgets required for material design of a app.

Step 05: Create a stateful or stateless widget outside void main().

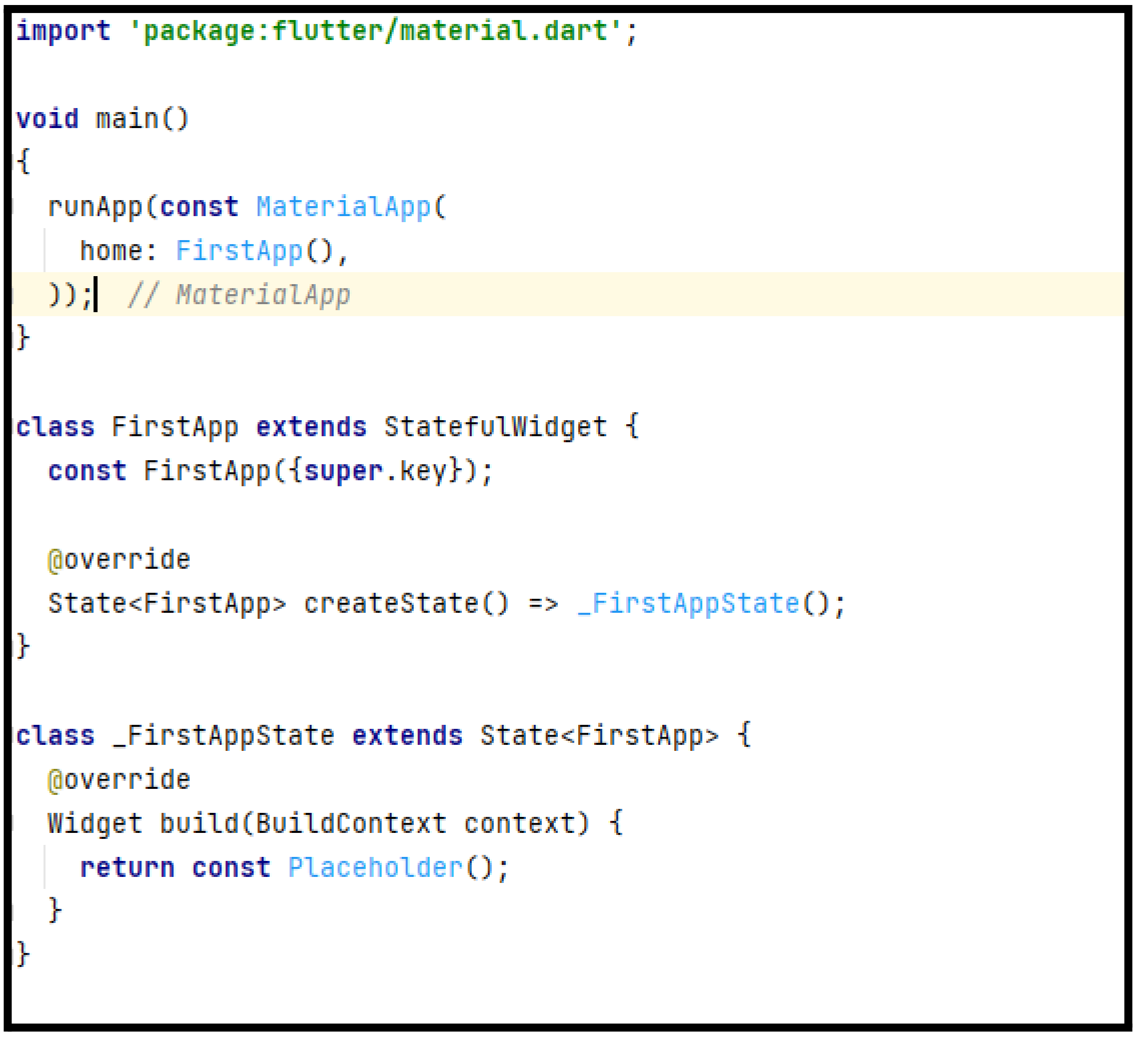
Create a stateful widget or stateless widget by just typing stf or stl outside the main function and then entering the name of the widget.

Whatever name you give to that widget add that widget name in the home property of the MaterialApp() like this



```dart
void main() {
  runApp(const MaterialApp(
    home: MyApp(),
  ));  // MaterialApp
}
```

Basic Code will look like



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

void main()
{
  runApp(const MaterialApp(
    home: FirstApp(),
  )); // MaterialApp
}

class FirstApp extends StatefulWidget {
  const FirstApp({super.key});

  @override
  State<FirstApp> createState() => _FirstAppState();
}

class _FirstAppState extends State<FirstApp> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

List Of Widgets (Kindly don't write new)

1. 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.

```
Text(
'Hello, How Are You!',
textAlign: TextAlign.center,
style: TextStyle(fontWeight: FontWeight.bold),
)
```

2. 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. In the above example, the onPressed property allows us to perform an action when you click the button,

```
//FlatButton Example
FlatButton(
  child: Text("Click here"),
  onPressed: () {
    // Do something here
  },
),



//RaisedButton Example
```

```
RaisedButton(
  child: Text("Click here"),
  elevation: 5.0,
  onPressed: () {
    // Do something here
  },
),
```

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.

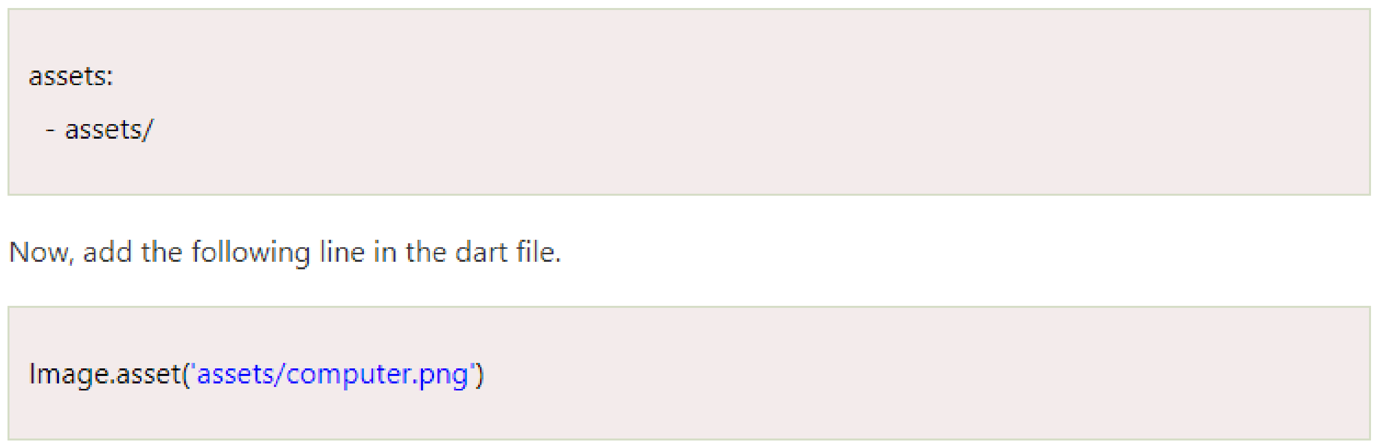
asset: It load image from your project asset folder.

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')
```

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.

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: <Widget>[
    Text(
      "VegElement",
    ),
    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.

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.

```
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.

State Management Widget

In Flutter, there are mainly two types of widget:

StatelessWidget

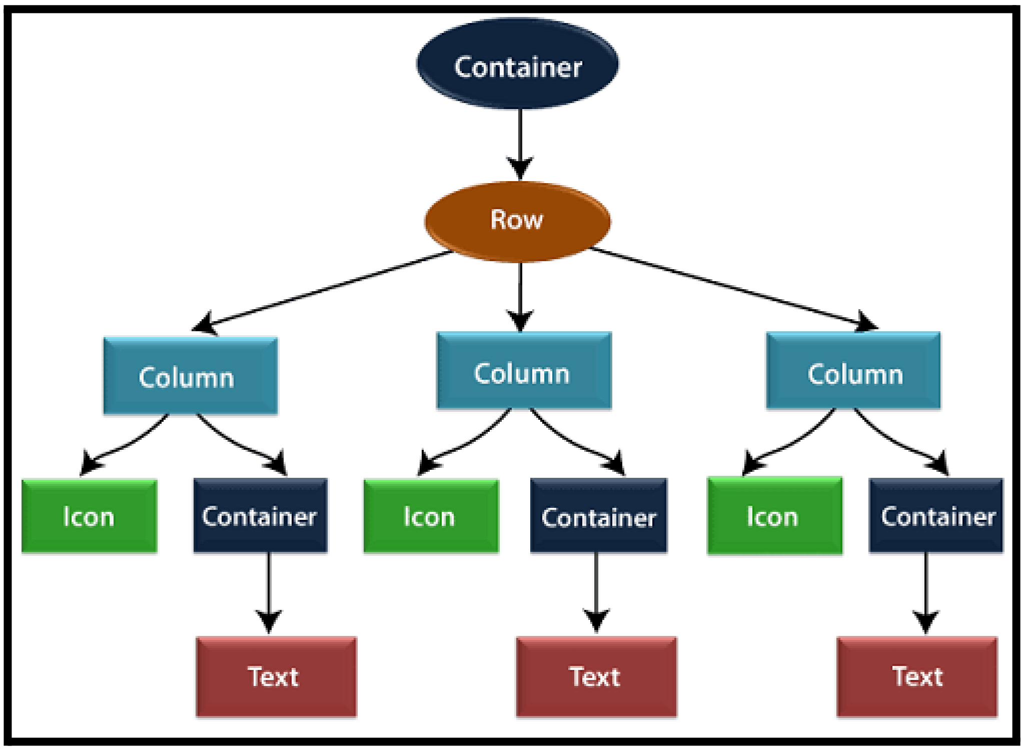
StatefulWidget

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.

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.



Types of Layout Widgets

We can categories the layout widget into two types:

Single Child Widget

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.

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

Gesture Detector

Flutter provides a widget that gives excellent support for all types of gestures by using the GestureDetector widget. The GestureWidget is non-visual widgets, which is primarily used for detecting the user's gesture. The basic idea of the gesture detector is a stateless widget that contains parameters in its constructor for different touch events.

In some situations, there might be multiple gesture detectors at a particular location on the screen, and then the framework disambiguates which gesture should be called. The GestureDetector widget decides which gesture is going to recognize based on which of its callbacks are non-null.

MediaQuery provides a higher-level view of the current app's screen size and can also give more detailed information about the device and its layout preferences. In practice, MediaQuery is always there. It can simply be accessed by calling MediaQuery.of in the build method.

From there you can look up all sorts of interesting information about the device you're running on, like the size of the screen, and build your layout accordingly. MediaQuery can also be used to check the current device's orientation or can be used to check if the user has modified the default font size. It can also be used to determine if parts of the screen are obscured by a system UI, similar to a safe area widget.

https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json