Chapter-1

Flutter: Introduction to Widgets

Widgets:

- Flutter builds UIs entirely with widgets.
- Widgets are like Lego blocks - you combine them to create complex layouts.
- Each widget describes a part of the UI (its looks, layout, and behavior).
- They are immutable - their state cannot change after creation.
- Widgets are arranged in a hierarchy (like a tree) - parent widgets control their children.

Gestures:

- User interactions like taps, swipes, drags, etc.
- Specific widgets (e.g., GestureDetector) handle gestures.
- These widgets detect gestures and trigger actions within your app.

Concept of State:

- Some widgets (StatefulWidgets) can hold data that changes (state).
- When the state changes, the widget rebuilds to reflect the update.
- Stateless widgets are simpler and faster but can't hold state (ideal for static UI elements).

Layers:

- **Widgets Layer**: Constructs the UI using widgets.
- **Rendering Layer**: Handles layout and painting of the widgets.
- **Painting Layer**: Draws the UI on the screen.
- **Gestures Layer**: Detects and responds to user gestures.
- **Services Layer**: Provides core services such as plugins and platform-specific functionality.

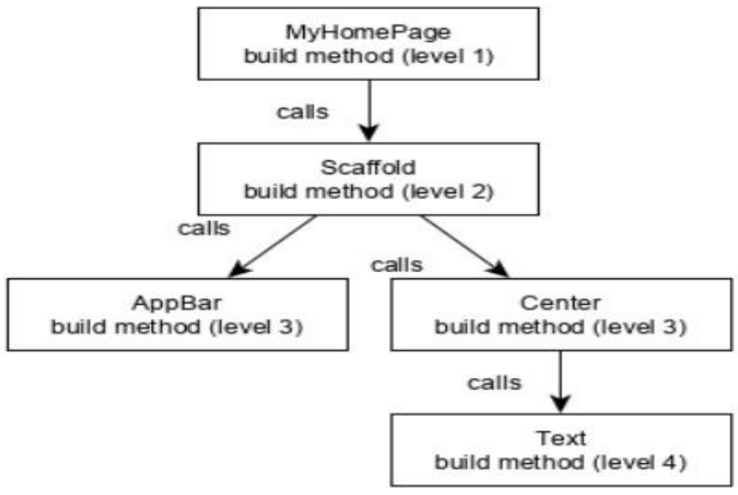
Widget Build Method Visualization (Hello World App):

Let us check the Hello World application’s MyHomePage widget. The code for this purpose is as given below:

```
import “package:flutter/material.dart”;
void main()
{
 runApp(MyApp());
}


class MyApp
{
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Hello World App'),
      ),
      body: Center(
        child: Text('Hello, World!'),
      ),
    ),
  );
}
```

For a better understanding, the visual representation of the same is given below:



Widget Build Visualization

Explanation:

- The runApp function is the starting point, it takes the root widget (MaterialApp) and sets it as the main widget for the app.
- The MaterialApp widget's build method is called first. It builds the overall application structure.
- Inside MaterialApp, the home property defines the child widget, which is a Scaffold widget in this case.
- The Scaffold widget's build method is called next. It builds the layout structure for the app (usually a body and optional app bar etc.).
- Inside Scaffold, the body property defines the child widget, which is a Center widget here.
- The Center widget's build method positions its child widget in the center of the available space.
- Finally, the Text widget's build method is called. It uses the provided text string ("Hello, world!") to render the actual text on the screen.

This hierarchy shows how each widget's build method is responsible for building its own UI and potentially calling the build methods of its child widgets. It continues down the tree until the final widget (Text) builds the actual visual element.

Chapter-2

Platform Specific Widgets, Types of Layout Widgets

Platform Specific Widgets:

Platform Specific Widgets are like tools that are specially designed for a particular type of device or operating system. Think of them as clothes that fit perfectly for a particular person.

Android: Imagine these widgets are designed to look and feel like Android apps. They follow the Material Design guidelines by Google.

Examples:

- **FloatingActionButton:** A round button that floats above the content, often used for primary actions.
- **SnackBar:** A small banner that appears at the bottom of the screen to show a quick message.
- **BottomNavigationBar:** A bar at the bottom of the screen for navigating between different sections of an app.

IOS: These widgets look and feel like iOS apps. They follow Apple's Cupertino design guidelines.

Examples:

- **CupertinoButton:** A button that looks like an iOS button.
- **CupertinoSlider:** A slider that looks like an iOS slider.
- **CupertinoNavigationBar:** A navigation bar that looks like an iOS navigation bar.

Advantages

- Feels natural and familiar to users of that platform.
- Better performance and integration with the operating system.

Disadvantages

- More work to create different versions for each platform.
- Less flexibility for customization.

Platform Independent Widgets:

Platform Independent Widgets are like universal tools that work the same way no matter what type of device you use. Think of them as clothes that fit anyone, regardless of size or shape.

Examples:

- **Text:** Displays a piece of text.
- **Image:** Shows an image.
- **Container:** A versatile box that can contain other widgets and add padding, margins, borders, and background colors.
- **Row:** Arranges child widgets in a horizontal line.
- **Column:** Arranges child widgets in a vertical line.

Advantages

- One set of code works on multiple platforms (Android, iOS, etc.).
- Easier to manage and update since you don't need to create multiple versions.

Disadvantages

- Might not look exactly like native apps on each platform.
- Sometimes there can be minor performance issues compared to platform-specific widgets.

Layout Widgets

Layout Widgets are like the layout plans of a room where you decide where to place furniture. In your app, layout widgets help you arrange other widgets (like text, buttons, images) on the screen.

Types of Layout Widgets

1. **Single Child Widget**
2. **Multiple Child Widget**

Single Child Widget

Single Child Widgets are designed to contain only one widget inside them. Think of them as boxes that hold one item and maybe do something special to it, like adding padding or centering it.

Examples

- **Container:** A versatile box that can add padding, margins, borders, and background colors to one child widget. It’s like a frame for your picture.
- **Padding:** Adds space around a single child widget. It’s like putting extra space around a picture in a frame.
- **Align:** Aligns one child widget within its boundaries. Imagine positioning a picture to the left, right, or center of a frame.
- **Center:** Centers one child widget within itself. It’s like putting a picture exactly in the middle of a frame.
- **SizedBox:** Gives a specific size to its single child. It’s like putting a picture in a frame of a fixed size.

Usage: These are used when you need to apply specific properties to one widget. For example, centering a text or adding padding around an image.

Multiple Child Widget

Multiple Child Widgets can contain multiple widgets inside them. Think of them as shelves where you can place multiple items in a particular order.

Examples:

- **Row:** Arranges its children in a horizontal line, like placing items side by side on a shelf.
- **Column:** Arranges its children in a vertical line, like stacking items on top of each other.
- **Stack:** Allows its children to overlap each other, like placing pictures one on top of another.
- **ListView:** A scrollable list of widgets arranged in a single column. Think of a list you can scroll through.
- **GridView:** A scrollable grid of widgets arranged in rows and columns, like a photo gallery.

Usage: These are used for more complex layouts where you need to arrange several widgets. For example, creating a row of buttons or a column of text fields.

Chapter-3
MaterialApp, Scaffold, Center Widget

StatelessWidget

A **stateless widget** in Flutter is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. It does not have any mutable state of its own, meaning once it is built, it cannot change its appearance or behavior.

Key Characteristics:

- **Immutability**: Stateless widgets cannot change their state after they are built. If the UI needs to update, a new widget must be created.
- **Performance**: Because they are immutable, stateless widgets are lightweight and simple, making them more efficient.

MaterialApp

Description:

The MaterialApp widget is the starting point for any Flutter app that uses Material Design. It sets up various settings and configurations for the app, such as the title, theme, locale, and home screen.

Key Properties:

1. **home:** The default route of the app, which is usually the first screen the user sees.
2. **title:** A one-line description used by the device to identify the app for the user.
3. **theme:** The theme data for the app, which allows you to customize the colors and styles.
4. **debugShowCheckedModeBanner:** If true, shows a debug banner on the top right corner of the screen (useful for development).

Example:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
    title: 'My First Flutter App',
    home: Scaffold(
      appBar: AppBar(
        title: Text('Home Page'),
      ),
      body: Center(
        child: Text('Hello, Flutter!'),
      ),
    ),
  ));
}
```

In this example **MaterialApp** is used to set up the app. The **home** property is set to a Scaffold widget, which contains an AppBar and a centered Text widget.

Scaffold

Description:

The Scaffold widget provides a structure for the visual interface of a Material Design app. It implements the basic visual layout structure, including an app bar, body, bottom navigation bar, floating action button, and drawer.

Key Properties:

1. **appBar:** A horizontal bar typically shown at the top of the app, containing the app's title and other actions.
2. **body:** The primary content of the Scaffold.
3. **floatingActionButton:** A button that floats above the content.
4. **drawer:** A panel that slides in from the side of the app.

Example:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Scaffold Example'),
      ),
      body: Center(
        child: Text('This is the body of the Scaffold.'),
      ),
    ),
  ));
}
```

In this example the **Scaffold** provides a basic layout with an **AppBar** and a centered text in the **body**.

Center Widget

Description:

The Center widget is a simple layout widget that centers its child within itself. It is useful when you want to center a single child widget in the middle of its parent.

Key Properties:

1. **child:** The widget to be centered.

Example:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: Text('Center Widget Example'),
      ),
      body: Center(
        child: Text('Centered Text'),
      ),
    ),
  ));
}
```

In this example the **Center** widget is used to center the **Text** widget within the body of the **Scaffold.**

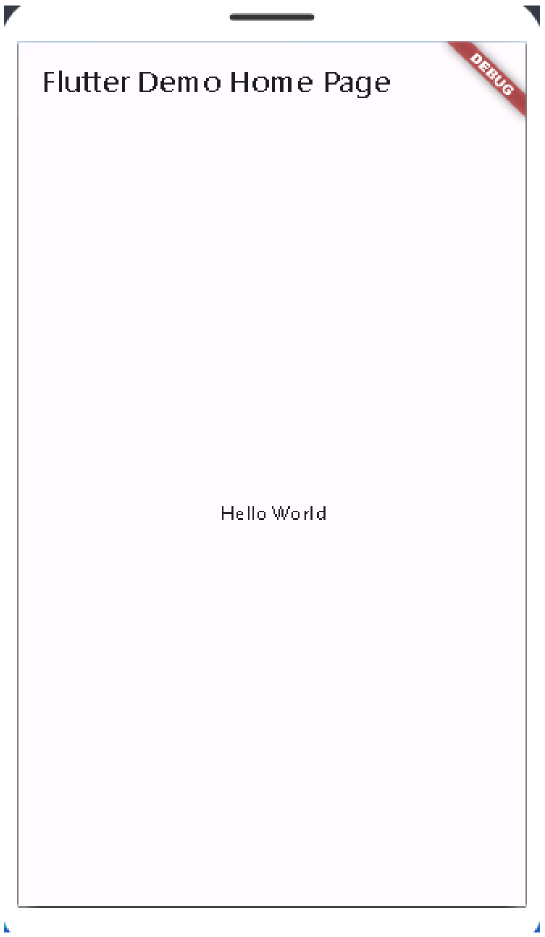
Putting It All Together

Here's a complete example that combines all the above widgets:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Basics',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Flutter Demo Home Page'),
        ),
        body: Center(
          child: Text('Welcome to Flutter!'),
        ),
      ),
    );
  }
}
```

Output Screenshot:



Flutter Demo Home Page

Hello World

Chapter-4
Text Widget

Text Widget

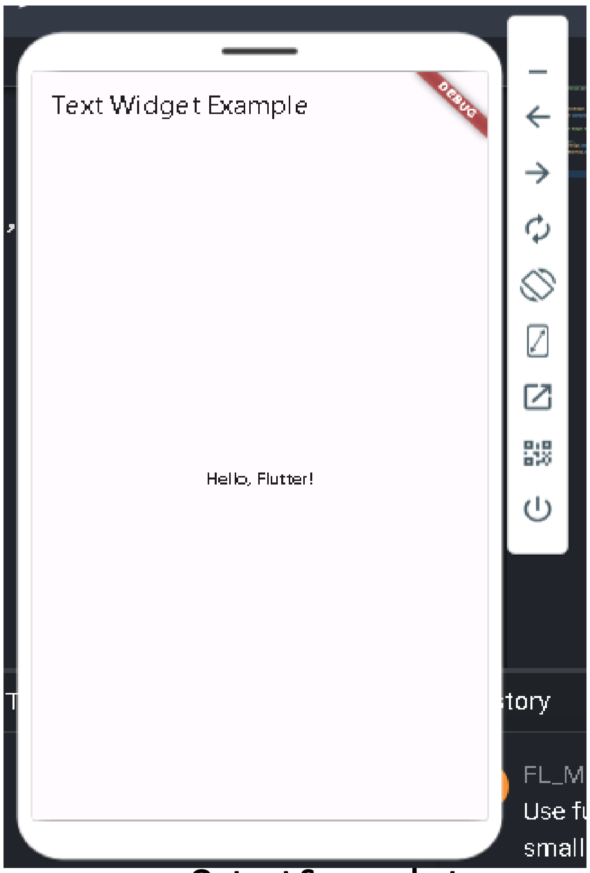
The `Text` widget in Flutter is used to display a string of text in the application. It supports various customization options to control its appearance and behavior, making it versatile for different text presentation needs.

The `Text` widget in Flutter is used to display a string of text with a single style.

Properties:

- `data`: The string to display.
- `style`: Defines the style of the text (e.g., color, font size, font weight).
- `textAlign`: Aligns the text within the widget (e.g., left, right, center).
- `maxLines`: The maximum number of lines for the text.
- `overflow`: Defines how to handle text overflow (e.g., ellipsis, fade).

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Text Widget Example'),
        ),
        body: Center(
          child: Text(
            'Hello, Flutter!',
            textAlign: TextAlign.center,
            maxLines: 2,
            overflow: TextOverflow.ellipsis,
          ),
        ),
      ),
    );
  }
}
```



Output Screenshot

Explanation of Properties:

- **textAlign**: This property aligns the text within its container. In this example, `TextAlign.center` centers the text horizontally.
- **maxLines**: This property specifies the maximum number of lines the text can occupy. Here, `maxLines: 2` allows the text to span up to two lines.
- **overflow**: This property handles how the text should be truncated or displayed when it exceeds the specified number of lines. In this example, `TextOverflow.ellipsis` adds an ellipsis ("...") at the end of the text if it overflows.

These properties help control the alignment, line limit, and overflow behavior of the text within the `Text` widget.

Style Property

The `style` property of the `Text` widget in Flutter allows you to customize the appearance of the text. It accepts a `TextStyle` object, which provides various options to style the text in different ways.

`TextStyle` Properties:

- **color**: Sets the color of the text.
- **fontSize**: Sets the size of the text.
- **fontWeight**: Sets the weight (thickness) of the text. Common values are `FontWeight.bold`, `FontWeight.normal`, etc.
- **fontStyle**: Sets the style of the text, such as normal or italic (`FontStyle.italic`).
- **letterSpacing**: Sets the space between letters.
- **wordSpacing**: Sets the space between words.
- **textBaseline**: Aligns the text to the alphabetic or ideographic baseline.
- **height**: Sets the height of the line (line height), which can be used to increase or decrease the space between lines of text.
- **backgroundColor**: Sets the background color of the text.

- **decoration**: Adds decoration to the text, such as underline, overline, or line-through. It uses `TextDecoration`.
- **decorationColor**: Sets the color of the text decoration.
- **decorationStyle**: Sets the style of the text decoration (e.g., solid, dashed, dotted).
- **fontFamily**: Sets the font family of the text.
- **shadows**: Adds shadows to the text.

Example:

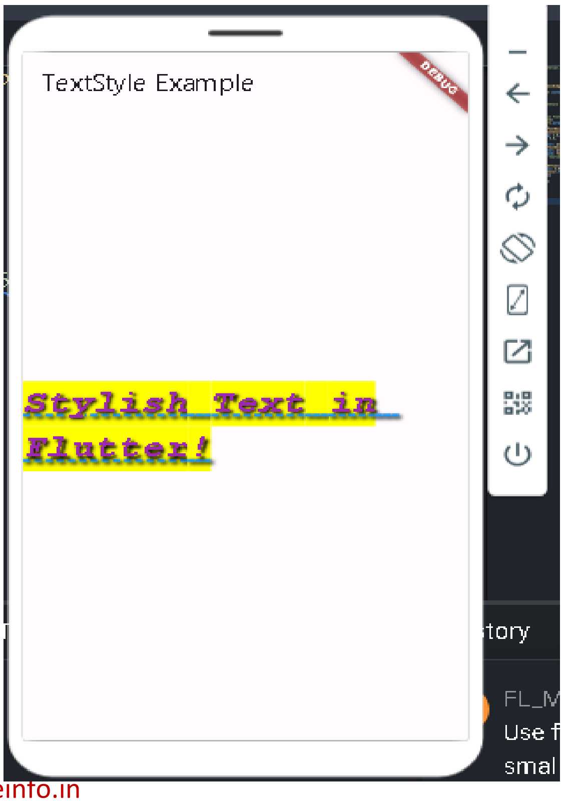
Let's look at an example that uses various `TextStyle` properties to customize the appearance of the text.

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('TextStyle Example'),
        ),
        body: Center(
          child: Text(
            'Stylish Text in Flutter!',
            style: TextStyle(
              color: Colors.purple,
              fontSize: 28,
              fontWeight: FontWeight.bold,
              fontStyle: FontStyle.italic,
              letterSpacing: 2.0,
              wordSpacing: 4.0,
              height: 1.5,
              backgroundColor: Colors.yellowAccent,
              decoration: TextDecoration.underline,
              decorationColor: Colors.blue,
              decorationStyle: TextDecorationStyle.dashed,
              fontFamily: 'Courier',
              shadows: [
                Shadow(
                  offset: Offset(2.0, 2.0),
                  blurRadius: 3.0,
                  color: Color.fromARGB(255, 0, 0, 0),
                ),
                ],
              ),
            ),
          ),
        ),
      );
    }
}
```

Breakdown of the Example:

- **color**: Sets the text color to purple.
- **fontSize**: Sets the text size to 28.
- **fontWeight**: Makes the text bold.
- **fontStyle**: Makes the text italic.
- **letterSpacing**: Adds 2 pixels of space between each letter.
- **wordSpacing**: Adds 4 pixels of space between each word.
- **height**: Sets the line height to 1.5 times the font size.
- **backgroundColor**: Sets the background color of the text to yellow.
- **decoration**: Underlines the text.
- **decorationColor**: Sets the underline color to blue.
- **decorationStyle**: Makes the underline dashed.
- **fontFamily**: Changes the font to Courier.
- **shadows**: Adds a shadow with an offset, blur radius, and color.

Output ScreenShot



Chapter-5
Image and Icon Widgets

Image Widget

The `Image` widget in Flutter is used to display images in the application. It supports loading images from various sources such as assets, network, and files, allowing for flexible and dynamic image rendering.

The `Image` widget in Flutter is used to display images in the application.

Properties:

- `image`: Specifies the image to display (from assets, network, or file).
- `width` and `height`: Define the dimensions of the image.
- `fit`: Defines how the image should fit within the bounds (e.g., fill, contain, cover).

Example

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Image Widget Example'),
        ),
        body: Center(
          child: Image.network(
'https:  cdn.pixabay.com/photo/2023/11/09/19/36/zoo-8378189_1280.jpg',
            width: 200,
            height: 200,
            fit: BoxFit.cover,
          ),
        ),
      ),
    );
  }
}
```

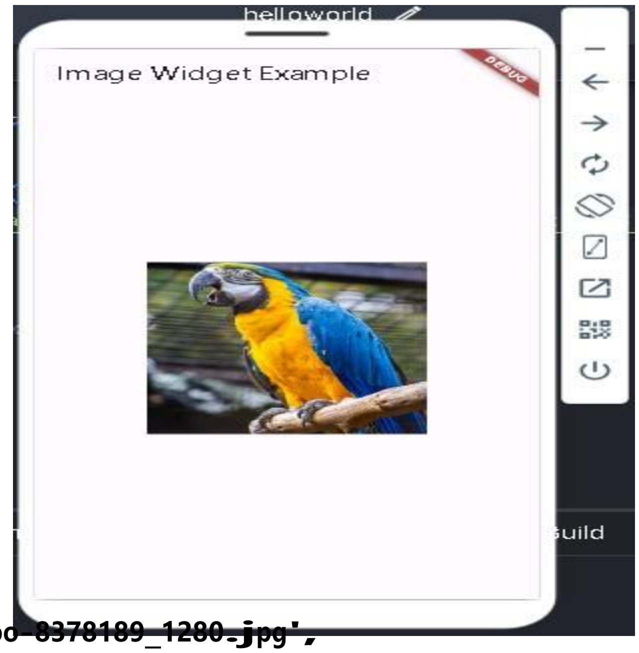


Image Widget Example

Output Screenshot

Steps to Add an Image to Assets

1. **Prepare the Image:**
   - Ensure the image you want to add is available and in a supported format (e.g., PNG, JPEG).
2. **Add the Image to the Project Directory:**
   - Create a directory in your Flutter project to store your images. A common convention is to create an `assets` directory and a subdirectory like `images`.

```
your_project/
├── assets/
│   └── images/
│       └── your_image.png
└── lib/
    └── main.dart
```

Update `pubspec.yaml`:

- Open the `pubspec.yaml` file in the root directory of your project.
- Add the path to the assets section. Ensure the asset path is correctly indented.

```
flutter:
  assets:
    - assets/images/your_image.png
```

3. Load the Image in Your Code:

- Use the `Image.asset` widget to display the image in your Flutter app.

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Image from Assets Example'),
        ),
        body: Center(
          child: Image.asset('assets/images/your_image.png'),
        ),
      ),
    );
  }
}
```

Icon Widget

The `Icon` widget in Flutter is used to display graphical icons. It provides a wide range of predefined icons from Material Design and custom icons, enabling consistent and visually appealing iconography in the app.

Description:

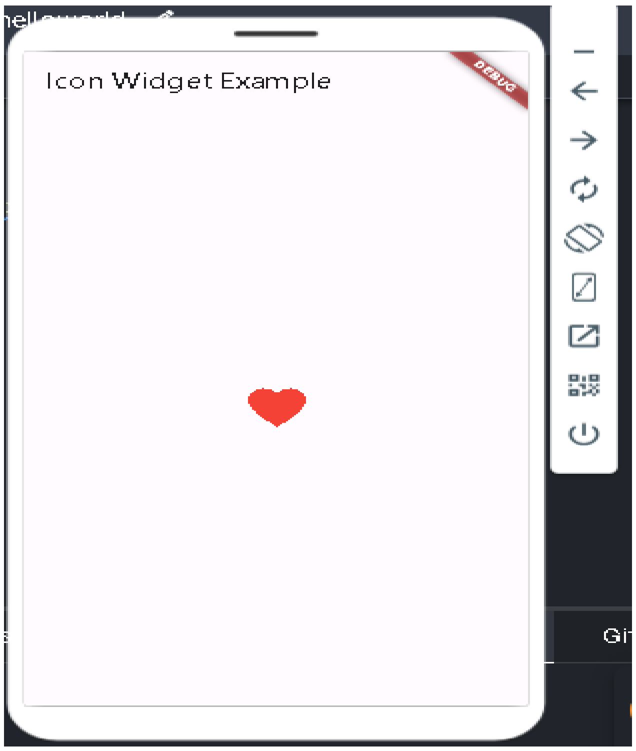
The `Icon` widget in Flutter is used to display a graphical icon.

Properties:

- `icon`: Specifies the icon to display (e.g., from Material Icons or custom icons).
- `color`: Sets the color of the icon.
- `size`: Sets the size of the icon.

Example:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Icon Widget Example'),
        ),
        body: Center(
          child: Icon(
            Icons.favorite,
            color: Colors.red,
            size: 50,
          ),
        ),
      ),
    );
  }
}
```



Output Screen Shot

Chapter-6

Row, Column and SizedBox Widgets

Row Widget

Description: The `Row` widget arranges its children in a horizontal sequence.

Properties

- **Main Axis: Horizontal (left to right by default).**
- **Cross Axis: Vertical (top to bottom by default).**

Use Case: Ideal for arranging widgets side by side, such as buttons, text fields, or icons.

Example with Text and Image Widget:

Column Widget

Description: The `Column` widget arranges its children in a vertical sequence.

- **Main Axis: Vertical (top to bottom by default).**
- **Cross Axis: Horizontal (left to right by default).**

Use Case: Suitable for stacking widgets vertically, like lists, forms, or detailed views.

MainAxisAlignment (Main Axis Alignment)

- **Determines how children are aligned along the main axis (horizontal for `Row`, vertical for `Column`):**

1. **start: Aligns children at the beginning of the main axis.**
2. **end: Aligns children at the end of the main axis.**
3. **center: Centers children along the main axis.**
4. **spaceBetween: Distributes space evenly between children, but not before the first or after the last child.**
5. **spaceAround: Distributes space evenly around all children.**

6. spaceEvenly: Distributes space evenly between and around children.

CrossAxisAlignment (Cross Axis Alignment)

- Determines how children are aligned along the cross axis (vertical for `Row`, horizontal for `Column`):

1. start: Aligns children at the start of the cross axis.
2. end: Aligns children at the end of the cross axis.
3. center: Centers children along the cross axis.
4. stretch: Stretches children to fill the cross axis.
5. baseline: Aligns children such that their baselines align.

SizedBox Widget

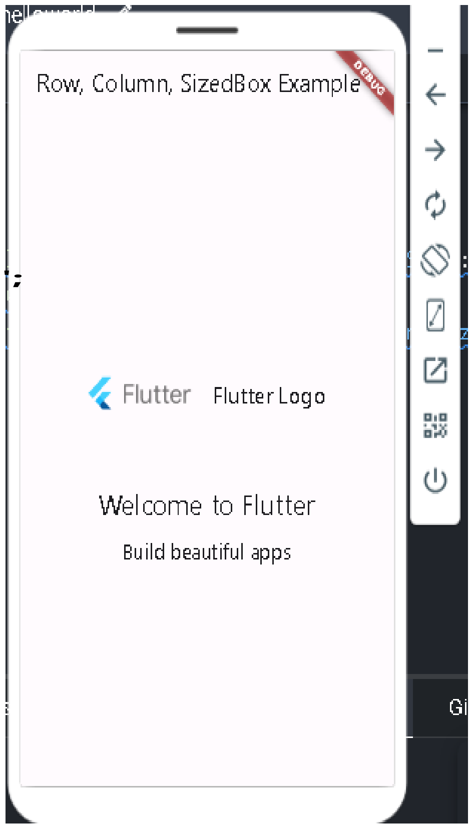
Description: The `SizedBox` widget is used to create a fixed size box within the layout. It can set either width, height, or both.

Use Case: Helpful for adding spacing between widgets, creating fixed dimensions, or acting as a placeholder.

Complete Example

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  final String flutterLogoUrl =
      'https:  upload.wikimedia.org/wikipedia/commons/1/17/Google-flutter-logo.png';
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Row, Column, SizedBox Example'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Row(

              mainAxisAlignment: MainAxisAlignment.center,
```



<u>Output Screenshot</u>

```
                children: [
                  Image.network(flutterLogoUrl, width: 100, height: 100),
                  SizedBox(width: 20),
                  Text('Flutter Logo', style: TextStyle(fontSize: 20)),
                ],
              ),
              SizedBox(height: 30),
              Column(
                children: [
                  Text('Welcome to Flutter', style: TextStyle(fontSize: 24)),
                  SizedBox(height: 10),
                  Text('Build beautiful apps', style: TextStyle(fontSize:
18)),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Explanation:

- **Row Widget:**
  - Arranges its children (`Image.network` and `Text` widgets) horizontally.
  - `MainAxisAlignment.center` aligns children at the center horizontally.
  - A `SizedBox` with `width: 20` adds 20 pixels of space between the `Image.network` and `Text`.
- **Column Widget:**
  - Stacks its children (`Text` widgets) vertically below each other.
  - `MainAxisAlignment.center` aligns children at the center vertically.
  - `SizedBox` with `height: 30` creates 30 pixels of vertical space between the `Row` and the first `Text` widget.
- **SizedBox Widget:**
  - Used to create fixed dimensions between widgets (`width` for `Row`, `height` for `Column`).
  - Helps in adding specific spacing or ensuring consistent layout within the UI.

Chapter-7
Button Widget

Button Widget:

The button widgets in Flutter are interactive elements used to trigger actions when pressed. They are essential for user interaction in apps and come in various types.

Common Properties:

1. **child**: The widget or text displayed as the button's content. It can be a `Text` widget for text-based buttons or an `Icon` widget for icon buttons.
2. **onPressed**: A callback function that gets triggered when the button is pressed. It defines the action to perform.

Types of Button Widget

1. **Elevated Button**
2. **TextButton**
3. **OutlineButton**
4. **IconButton**
5. **FloatingActionButton**

1. **ElevatedButton:** A material design button with elevation when pressed.

```
ElevatedButton(
  onPressed: () {
       Action to perform when button is pressed
  },
  child: Text('Elevated Button'),
)
```

2. **TextButton:** A text-based button with no elevation.

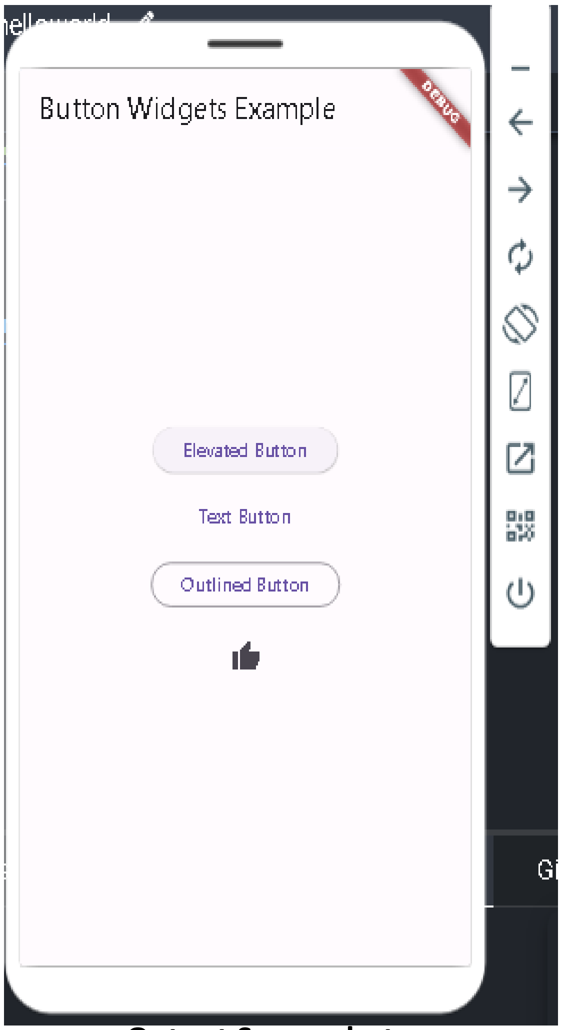
```
TextButton(
  onPressed: () {
       Action to perform when button is pressed
  },
  child: Text('Text Button'),
)
```

3. **OutlinedButton:** A button with an outlined border.

```
OutlinedButton(
  onPressed: () {
       Action to perform when button is pressed
  },
  child: Text('Outlined Button'),
)
```

4. IconButton: A button with an icon.

```
IconButton(
  icon: Icon(Icons.star),
  onPressed: () {
       Action to perform when button is pressed
  },
)
```



Output Screenshot

Complete Example

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Button Widgets Example'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              ElevatedButton(
                onPressed: () {
                  print('Elevated Button Pressed');
                },
                child: Text('Elevated Button'),
              ),
              SizedBox(height: 16),
              TextButton(
                onPressed: () {
                  print('Text Button Pressed');
                },
                child: Text('Text Button'),
              ),
              SizedBox(height: 16),
              OutlinedButton(
                onPressed: () {
                  print('Outlined Button Pressed');
                },
                child: Text('Outlined Button'),
              ),
              SizedBox(height: 16),
              IconButton(
                icon: Icon(Icons.thumb_up),
                onPressed: () {
                  print('Icon Button Pressed');
                },
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Explanation:

- **ElevatedButton:** When pressed, prints "Elevated Button Pressed" to the console.
- **TextButton:** When pressed, prints "Text Button Pressed" to the console.
- **OutlinedButton:** When pressed, prints "Outlined Button Pressed" to the console.
- **IconButton:** When pressed, prints "Icon Button Pressed" to the console.

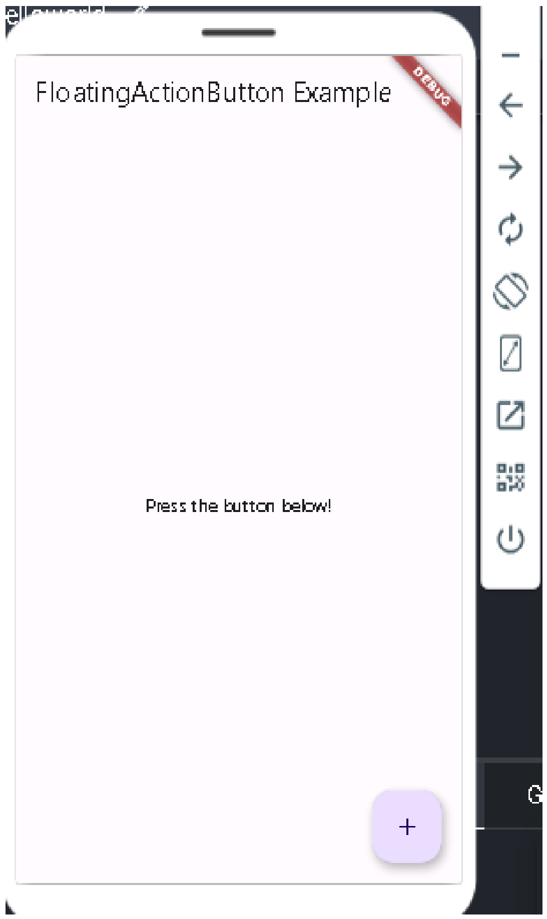
FloatingActionButton

The **FloatingActionButton** widget in Flutter is a button typically placed in a scaffold's **Scaffold.floatingActionButton** property, used for primary actions in the application. It floats above the main content and is usually circular in shape.

Properties of FloatingActionButton:

- **onPressed:** Callback function invoked when the button is pressed.
- **child:** Widget displayed inside the button (typically an Icon).
- **backgroundColor:** Background color of the button.
- **foregroundColor:** Color of the button's child widget.
- **elevation:** Elevation (shadow) of the button.
- **tooltip:** Optional tooltip text displayed when the button is long-pressed.

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  void _showMessage() {
    print('Floating Action Button Pressed');
  }
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('FloatingActionButton Example'),
        ),
        body: Center(
          child: Text('Press the button below!'),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: _showMessage,
          tooltip: 'Increment',
          child: Icon(Icons.add),
        ),
      ),
    );
  }
}
```


FloatingActionButton Example

Press the button below!

Explanation:

Output Screenshot

- **onPressed:** When the button is pressed, the **_showMessage** function is called, which prints a message to the console.
- **tooltip:** Displays "Increment" when the button is long-pressed, providing a brief description of the button's action.
- **child:** The **Icon(Icons.add)** widget serves as the button's visual representation, displaying a plus sign inside a circular button.
- **floatingActionButton:** Positioned at the bottom right corner of the screen by default, the **FloatingActionButton** floats above other content in the **Scaffold.**

Chapter-8
Expanded, Align, Container, Padding Widgets

Expanded Widget

Description: The Expanded widget expands a child widget to fill the available space along the main axis (horizontal in a Row, vertical in a Column).

Properties:

- **flex**: An integer that determines the ratio of space that the child occupies relative to other expanded children. Defaults to 1 if not specified.
- **child**: The widget that is expanded to fill the available space.

Use Case: Used within Row or Column to distribute available space among multiple children or to stretch a single child.

Example: Using Expanded in a Row

```
Row(
  children: [
    Expanded(
      flex: 2,
      child: Container(
        color: Colors.blue,
        height: 50,
      ),
    ),
    Expanded(
      child: Container(
        color: Colors.green,
        height: 50,
      ),
    ),
  ],
)
```

Align Widget

Description: The Align widget aligns its child within itself based on the alignment property.

Properties:

- **alignment**: Specifies how the child should be positioned within the parent. Uses Alignment class constants like Alignment.topLeft, Alignment.center, etc.

- **child**: The widget to be aligned within the Align widget's bounds.

Use Case: Useful for aligning a single child widget within a parent widget, adjusting its position relative to the parent's bounds.

Example: Aligning Text Center in a Container

```
Container(
  height: 100,
  width: 100,
  color: Colors.grey,
  child: Align(
    alignment: Alignment.center,
    child: Text('Centered Text'),
  ),
)
```

Container Widget

Description: The Container widget is a versatile widget that allows customization of its child widget's appearance, such as alignment, padding, margins, borders, and more.

Properties:

- **alignment**: Aligns the child within the container. Uses Alignment class constants.
- **padding**: Creates space around the child widget inside the container.
- **margin**: Creates space around the outside of the container.
- **color**: Sets the background color of the container.
- **decoration**: Applies decoration like border, background image, etc., to the container.

Use Case: Used as a basic layout element to contain and style other widgets within it.

Example: Container with Padding and Border

```
Container(
  padding: EdgeInsets.all(20),
  decoration: BoxDecoration(
    border: Border.all(color: Colors.black),
    borderRadius: BorderRadius.circular(10),
  ),
  child: Text('Container with Padding and Border'),
)
```

Padding Widget

Description: The Padding widget adds padding around its child widget.

Properties:

- **padding**: Defines the padding space. It uses EdgeInsets class to specify padding on all sides (all), individually (only), or symmetrically (symmetric).

Use Case: Used to add space around a child widget to provide visual separation or prevent content from touching the edges of its parent.

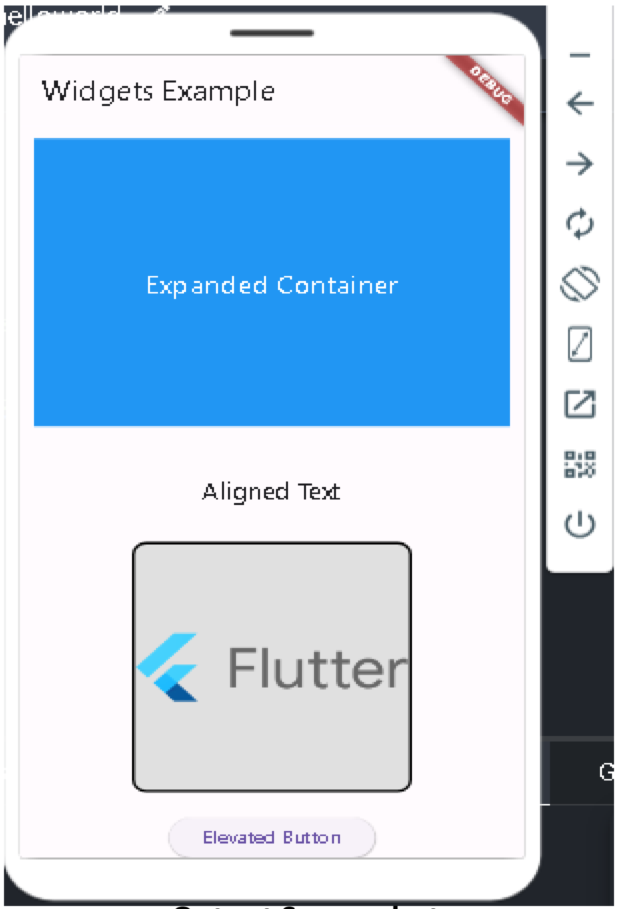
Example: Padding Around Text

```
Padding(
  padding: EdgeInsets.all(16),
  child: Text('Text with Padding'),
)
```

Complete Example

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  final String imageUrl =

'https:  upload.wikimedia.org/wikipedia/commons/1/17/Google-flutter-logo.png';
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Widgets Example'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Expanded(
                flex: 2,
                child: Container(
                  color: Colors.blue,
                  alignment: Alignment.center,
                  padding: EdgeInsets.all(16),
                  margin: EdgeInsets.all(10),
                  child: Text(
                    'Expanded Container',
                    style: TextStyle(fontSize: 20, color: Colors.white),
                  ),
                ),
              ),
              SizedBox(height: 20),
              Align(
                alignment: Alignment.center,
                child: Padding(
                  padding: EdgeInsets.all(8),
                  child: Text(
                    'Aligned Text',
                    style: TextStyle(fontSize: 18),
                  ),
                ),
              ),
              SizedBox(height: 20),
              Container(
                width: 200,
                height: 200,
                decoration: BoxDecoration(
                  color: Colors.grey[300],
                  border: Border.all(color: Colors.black, width: 2),
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Image.network(
                  imageUrl,
                  fit: BoxFit.contain,
                ),
              ),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () {
                     Button action
                  print('Button Pressed');
                },
                child: Text('Elevated Button'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```



Output Screenshot

Explanation:

Expanded Widget:

1. **Purpose**: Ensures the Container fills the available vertical space in the Column.

   - **Explanation**: The blue Container uses Expanded to take up as much space as possible vertically (flex: 2), making it larger relative to other widgets.
2. **Align Widget**:
   - **Purpose**: Centers the "Aligned Text" within its parent.
   - **Explanation**: The Align widget centers the Text widget horizontally and vertically within its parent Column.
3. **Container Widget**:
   - **Purpose**: Contains the Image widget with custom styling (border, padding, margin).
   - **Explanation**: The Container provides a gray background with a black border, ensuring the Image inside it (Image.network) is visually distinct and neatly enclosed.
4. **Padding Widget**:
   - **Purpose**: Adds space around the "Aligned Text".
   - **Explanation**: The Padding widget places padding around the Text widget to give it some breathing room within the Align widget.
5. **Text Widget**:
   - **Purpose**: Displays textual content.
   - **Explanation**: Simple text widgets used to demonstrate text display within different layout widgets (Container, Align).
6. **Button Widget**:
   - **Purpose**: Triggers an action when pressed.
   - **Explanation**: The ElevatedButton triggers a print statement when pressed, showcasing how buttons can be used for user interaction.
7. **Image Widget**:
   - **Purpose**: Displays an image fetched from a network URL.
   - **Explanation**: The Image.network widget loads and displays an image fetched from the specified imageUrl, demonstrating network image loading capability.

Two Marks Questions:

1. What are Widgets in Flutter?
2. Briefly describe the difference between Stateful and Stateless widgets.
3. What are gestures in Flutter?
4. What is the role of the build method in a Flutter widget?
5. Give two examples each of Platform Specific Widgets for Android and iOS.
6. What is the advantage of using Platform Independent Widgets?
7. What are Single Child Widgets in Flutter, and provide an example.
8. What is the purpose of a Row widget in Flutter layouts?
9. What property in a MaterialApp widget sets the title for the app?
10. What does the Center widget do in Flutter layouts?
11. What is the primary function of the Text widget in Flutter?
12. Briefly describe how the textAlign property affects the text within the Text widget.
13. What property in the Text widget controls how to handle overflowing text?
14. What style property in Flutter sets the font size of the Text widget?
15. What is the primary function of the Image widget in Flutter?
16. What is the purpose of the Icon widget in Flutter?
17. What property in the Image widget controls how an image resizes to fit its container?
18. In which direction does a Row widget arrange its child widgets by default?
19. Briefly describe the purpose of the SizedBox widget in Flutter.
20. What property in a Row or Column widget controls the alignment of child widgets along the main axis?
21. Briefly describe two common properties of most button widgets in Flutter.
22. What type of button in Flutter has an elevated appearance when pressed?
23. What does the Expanded widget do in Flutter layouts?
24. Briefly describe how the Align widget is used to position its child widget.
25. What property in a Container widget controls the background color?
26. What purpose does padding serve in the Padding widget?
27. Where in the widget tree is the EdgeInsets class typically used?

Five Marks Questions:

1. Describe the different layers involved in rendering a UI in Flutter. Briefly explain the function of each layer.
2. Discuss the advantages and disadvantages of using Stateful vs. Stateless widgets?

3. Explain the concept of Platform Specific Widgets in Flutter. Discuss the advantages and disadvantages of using them in app development.
4. Describe different types of Layout Widgets in Flutter, categorized as Single Child Widgets and Multiple Child Widgets. Provide examples of each type.
5. Briefly explain the role of Container, Padding, and Center widgets in Flutter?
6. Describe the different properties of the MaterialApp widget and how they configure a Flutter application.
7. How does the Scaffold widget contribute to the layout structure of a Material Design app in Flutter? Give an example of the properties it provides.
8. Explain the different properties available for the Text widget in Flutter and how they can be used to control the layout and appearance of the text.
9. Describe the concept of the style property in the Text widget and how it is used with the TextStyle object for styling text in Flutter.
10. Explain the different properties available for the Image widget in Flutter and how they are used to control the display of images.
11. Describe the steps involved in adding an image from your project assets to be used in a Flutter application.
12. Explain the difference between the Row and Column widgets in Flutter and provide examples of their use cases.
13. Describe the MainAxisAlignment property and how it affects the alignment of child widgets within a Row or Column widget.
14. Explain the different types of button widgets available in Flutter and provide an example of each type.
15. Describe the purpose of the onPressed callback function in a button widget and how it is used.
16. Discuss the functionalities and properties specifically associated with the FloatingActionButton widget.
17. Explain the functionalities of the Expanded widget and provide an example.
18. Describe the concept of alignment in Flutter and how the Align widget is used to achieve different alignments for child widgets.
19. How can you achieve various visual effects for a container using properties of the Container widget in Flutter? Provide examples.
20. Discuss the use cases of the Padding widget and how it affects the layout of child widgets within its bounds.