

## 1) \_Flutter Introduction & Basics

This section will help you to setup environment for flutter development and learn the basics of flutter. Here you will learn the following topics:

- [Introduction to Flutter,](#)
- [How to install Flutter,](#)
- [Useful Flutter Commands,](#)
- [Basic Flutter Program,](#)
- [Widgets in Flutter,](#)
- [Text in Flutter,](#)
- [Container in Flutter,](#)
- [Row in Flutter,](#)
- [Column in Flutter,](#)
- [Row and Column in Flutter,](#)
- [Stack in Flutter,](#) and
- [Create Counter App.](#)

### Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

## INTRODUCTION TO FLUTTER

### What is Flutter?

Flutter is a free tool developed by Google for creating mobile, web, and desktop apps with single code. It uses [Dart programming language](#) to create apps. Flutter quickly gained popularity because of easy and fast development process.

### Why You Should Learn Flutter?



Flutter offers a many advantages for developers and businesses like:

- **Single Codebase:** You can use the same code to make apps for android, iOS, web, and desktop.
- **Beautiful UI:** Flutter has special things called [widgets](#), that help you to create beautiful user interfaces.
- **Fast Development:** Flutter has a hot reload feature that allows you to see changes in your app in real time. This makes the development process faster.
- **Efficient Testing:** With Flutter, you can easily test your app on different devices and platforms at the same time.
- **Ready Made Packages:** You can get thousands of ready made [packages](#) to enhance your app's functionality.

## Some Popular Apps Built With Flutter

Here are some popular apps made with Flutter:

- [Google Earth](#),
- [Youtube Create](#),
- [Google Pay](#),
- [Alibaba](#),
- [BMW](#),

 <b>Date</b>	 <b>Event</b>
2015	Google started Flutter as an <b>open-source</b> project.
May 12, 2017	First alpha release of Flutter.
Dec 4, 2018	Flutter 1.0 (first stable version) launches.
Mar 3, 2021	Flutter 2.0 releases with web & desktop support.
May 5, 2022	Flutter 3.0 releases with advanced features.
Feb 15, 2024	Flutter 3.19 releases with more advanced features.

- [Ebay](#),
- [Google Ads](#), etc.

**Note:** As of May 2023, there are more than **1 millions apps built with flutter**. This shows the popularity of flutter in such a short time.

## Flutter History

Here is a timeline of Flutter's history:

## Key Points

- Flutter allows you to build, test, and publish mobile, web, desktop apps from a single codebase.
- It uses Dart programming language, which helps you to build apps fast for many platforms.
- It has lots of widgets to build beautiful user interfaces.

- There are many ready made packages available for Flutter.
- It is backed by Google and trusted by well-known brands worldwide.

## INSTALL FLUTTER

### Flutter Installation

There are multiple ways to install a flutter on your system. You can install Flutter on **Windows, Mac, and Linux** or run it from the browser. In this tutorial, you will learn easiest way to install a flutter on your system. If you got stuck somewhere, you can watch the video below to be more clear.

### Requirements

- **Flutter SDK**,
- **VS code or other editors** like IntelliJ [We will use VS Code here].
- **Git** [For Version Control].
- **Anrdoid Studio**.

### Flutter Windows Installation

Follow the below instructions to install a flutter on the windows operating system.

#### Steps:

- Download **Flutter** sdk.
- Extract or copy Flutter sdk to your C drive.
- Setup path **C:\flutter\bin** to your environment variable. Watch the video below to be more clear.
- Open the command prompt and type **flutter --version** to check it.

```
flutter --version
```

- Install **VS Code** from [here](#) and add Dart and Flutter Extension.
- Install **Android Studio** and add **Android sdk**.
- Install **Git** from [here](#) for version control and to keep track of your code changes.
- Open the command prompt and type **flutter doctor** to check and diagnose the setup of your flutter development environment.

```
flutter doctor
```

### Flutter Windows Installation Video

Watch this video to easily install a flutter on your windows operating system.

## USEFUL FLUTTER COMMANDS

### Useful Flutter Commands

Here you will find a list of useful Flutter commands that will make your app development process easier and faster.

#### 1. Setting Up

Before diving into development, it's essential to ensure that Flutter is set up correctly on your machine:

```
flutter doctor
```

This command checks your environment and displays a report of the status of your Flutter installation. It will also notify you of any dependencies you still need to install.

#### 2. Creating a New Project

To start a new Flutter project you need to run `flutter create` command. This command will create a new Flutter project in the specified directory.

```
flutter create my_app
```

This command sets up a new Flutter project called `my_app`.

#### 3. Running the App

After creating a project, navigate to the project directory and use this command to run the Flutter app in the default device (emulator/simulator).

```
flutter run
```

#### 4. Checking Installed Devices

To check the list of all connected devices, use the following command:

```
flutter devices
```

#### 5. Building Apk For Android

To build an APK file for Android, use:

```
flutter build apk
```

## 6. Building IPA For iOS

To build an IPA file for iOS, use:

```
flutter build ios
```

## 7. Get All Packages

To get all the packages in your project, use:

```
flutter pub get
```

## 8. To Create Android App Bundle

To create an app bundle for Android to publish app, use:

```
flutter build appbundle
```

## Conclusion

These are some of the basic commands to get you started with Flutter development. As you go advance, you will find more commands that Flutter provides to aid in the development process. Happy learning!

## BASIC FLUTTER PROGRAM

### Basic Flutter Program

In this section, you will learn how to create and run simple flutter program.

### Steps To Create Flutter Project

Let's create a flutter project called **first\_app**. Follow the below steps to create a flutter project.

- Open command prompt/terminal.
- Type **flutter create first\_app** and press enter.
- Type **cd first\_app** and press enter.
- Type **code** . to open project with visual studio code.

### Example 1: Hello World Program

This is a simple Flutter application that displays **Hello World** on screen. First, open the **main.dart** file from the **lib** folder and replace the code with the below code.

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My First App'),
        ),
        body: const Center(
          child: Text('Hello World!'),
        ),
      ),
    ),
  );
}
```

## Basic Flutter Program Explained

- The **import 'package:flutter/material.dart';** means we're using the **Material design package** from Flutter.
- The **void main() {...}** function is where every **Flutter application starts**.
- The **runApp(...);** function is how we **initialize and launch** the app's UI.
- The **MaterialApp** widget is our foundation for creating Material Design apps in Flutter.
- The **Scaffold** widget provides a basic visual structure, like a canvas, to design our app upon.
- The **AppBar** widget is like the top bar of an app, where we can place a title or action buttons.
- The **Text** widget is simply for displaying text on the screen.
- The **Center** widget centers the child widget.

## Run Flutter Project

To run flutter project, go to run menu and click on **Start Debugging** or **Run Without Debugging**. You can also run the project with this command.

```
flutter run
```

## Challenge

- Create a flutter project called **second\_app**. Display your name on screen.

## WIDGETS IN FLUTTER

### What is Flutter Widgets?

In Flutter, everything on the screen is a widget. Flutter widgets are the UI elements that you see on the screen e.g. button, text, image, list, etc. Widgets can be either stateful or stateless.

## Types of Widgets in Flutter

Flutter widgets are broadly categorized into:

- **Stateless Widget**
- **Stateful Widget**

### Stateless Widget

Stateless widgets are immutable. Once you define their properties, they remain constant and don't change. They are the go-to choice for displaying static content like text, images, or icons.

**Note:** You can use **stl** shortcut to create a stateless widget in VS Code.

#### Example of a Stateless Widget:

```
class MyStatelessWidget extends StatelessWidget {
  final String text;

  MyStatelessWidget({required this.text});

  @override
  Widget build(BuildContext context) {
    return Text(text);
  }
}
```

### Stateful Widget

Stateful widgets are dynamic and can change during their lifecycle, usually in response to user interactions or real-time data updates. They are essential for content that evolves over time, such as a shopping cart's contents.

**Note:** You can use **stf** shortcut to create a stateful widget in VS Code.

#### Example of a Stateful Widget:

```
class MyStatefulWidget extends StatefulWidget {
  @override
  _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}
```

```

class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  int counter = 0;

  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        setState(() {
          counter++;
        });
      },
      child: Text('You have pressed the button $counter times.'),
    );
  }
}

```

## Stateless Vs Stateful Widgets

	Stateless Widgets	Stateful Widgets
Properties	Stateless widgets remain constant and don't change	Stateful widgets can change their appearance based on events or data.
Use Case	For static content (e.g., buttons, icons)	For dynamic content (e.g., user interactions)
State	No internal state	Maintains internal state that can be updated
Rebuild	Entire widget tree rebuilt on state changes	Rebuilds only the widget affected by state change

## Most Used Flutter Widgets

- **Text:** It is used to display text on the screen.
- **Container:** It is used to display a rectangular box on the screen.
- **Row:** It is used to display widgets in a horizontal manner.
- **Column:** It is used to display widgets in a vertical manner.
- **Stack:** It is used to display widgets on top of each other.



**Note:** You can use **Container** widget to add padding, margin, border, background color, etc. to other widgets.

## TEXT IN FLUTTER

### Text In Flutter

**Text** widget is used to display a string of text. In this section, you will learn how to use text widget, style text, align text, and handle overflow.

#### Example 1: Display Text In Flutter

In this example, you will learn how to display your name using **Text** widget.

```
Text('Raj Sharma')
```

#### Example 2: Center Text In Flutter

In this example, you will learn how to center text in flutter. In center widget, you can pass any widget as a child. Here we are passing Text widget as a child.

```
Center(  
  child: Text('Raj Sharma'),  
)
```

**Note:** Center widget is used to center the child widget. Here child widget is Text widget.

#### Example 3: Styling Text In Flutter

In this example, you will learn how to style text in flutter. You will learn to change font size, font weight, and font color.

```
Text(  
  'Raj Sharma',  
  style: TextStyle(  
    fontSize: 24,  
    fontWeight: FontWeight.bold,  
    color: Colors.blue,  
  ),  
)
```

#### Example 4: Align Text In Flutter

In this example, you will learn how to align text in flutter. You will learn to align text to left, right, and center.

```
Center(  
  child: Text(  
    'Hello I am Mark and I am a Flutter Developer at Technology Channel.',  
    textAlign: TextAlign.center,  
  ),  
)
```

**Tip:** Try changing the value of **textAlign** property to **TextAlign.left**, **TextAlign.right**, **TextAlign.justify**.

### Example 5: Underline Text In Flutter

In this example, you will learn how to underline text in flutter. You will learn to underline text using **decoration** property.

```
Text(  
  'Raj Sharma',  
  style: TextStyle(  
    decoration: TextDecoration.underline,  
  ),  
)
```

### Example 6: Text With Background Color In Flutter

In this example, you will learn how to add background color to text in flutter. You will learn to add background color using **backgroundColor** property.

```
Text(  
  'Raj Sharma',  
  style: TextStyle(  
    backgroundColor: Colors.blue,  
    fontSize: 20,  
  ),  
)
```

### Example 7: Handle Text Overflow In Flutter

In this example, you will learn how to handle overflow in flutter. You will learn to handle overflow using **overflow** property.

```
Text(  
  'A very long text that might not fit the screen. '*10,  
  // try commenting the below line and see the difference  
  overflow: TextOverflow.ellipsis,  
)
```

## CONTAINER IN FLUTTER

### Container In Flutter

**Container** is a box like widget that can be shaped, colored, and sized according to your needs. In this section, you will learn how to use container widget, style it, add border, padding, margin, and background color.

### Key Features of a Container

- **Size:** You can specify its height and width.
- **Color and Decoration:** You can paint it with a color or add more complex decorations like borders, rounded corners, or shadows.
- **Alignment:** It lets you align its child (what you put inside the container) to center, left, right, etc.
- **Padding and Margin:** Think of padding as the space inside the box between its edges and the content. Margin is the space outside the box, between the box and other elements.

### Example 1: Display Container In Flutter

In this example, you will learn how to display container in flutter. In the child there is Text widget.

```
Container(  
  child: Text('Raj Sharma'),  
)
```

### Example 2: Style Container In Flutter

In this example, you will learn to style container by adding padding and color to it.

```
Container(  
  child: Text('Raj Sharma'),  
  color: Colors.blue,  
  padding: EdgeInsets.all(20),  
)
```

**Note:** You can use **Container** widget to add padding, margin, border, background color, etc. to other widgets.

### Example 3: Add Border To Container

In this example, you will learn how to add border to container. You will learn to add border using **Container** widget.

```
Container(  
  child: Text('Raj Sharma'),  
  padding: EdgeInsets.all(20),  
  decoration: BoxDecoration(  
    border: Border.all(  
      color: Colors.black,  
      width: 2,  
    ),  
  ),  
)
```

#### Example 4: Add Margin & Padding To Container

In this example, you will learn how to add margin and padding to container.

```
Container(  
  child: Text('Raj Sharma'),  
  color: Colors.blue,  
  padding: EdgeInsets.all(20),  
  margin: EdgeInsets.all(20),  
)
```

#### Example 5: Add Background Color To Container In Flutter

In this example, you will learn how to add background color to container.

```
Container(  
  child: Text('Raj Sharma'),  
  color: Colors.blue,  
  padding: EdgeInsets.all(20),  
)
```

#### Example 6: Add Shadow To Container In Flutter

In this example, you will learn to add shadow to container.

```
Container(  
  child: Text('Raj Sharma'),  
  padding: EdgeInsets.all(20),  
  decoration: BoxDecoration(  
    boxShadow: [  
      BoxShadow(  
        color: Colors.grey,  
        blurRadius: 10,  
        offset: Offset(4, 4),  
      ),  
    ],  
  ),  
)
```

```
],  
)  
)
```

### Example 7: Add Rounded Corners To Container

In this example, you will learn how to add rounded corners to container in flutter.

```
Container(  
  child: Text('Raj Sharma'),  
  padding: EdgeInsets.all(20),  
  decoration: BoxDecoration(  
    color: Colors.blue,  
    borderRadius: BorderRadius.circular(10),  
  ),  
)
```

### Example 8: Add Gradient To Container

In this example, you will learn to add gradient to container.

```
Container(  
  child: Text('Raj Sharma'),  
  padding: EdgeInsets.all(20),  
  decoration: BoxDecoration(  
    gradient: LinearGradient(  
      colors: [  
        Colors.blue,  
        Colors.green,  
      ],  
    ),  
  ),  
)
```

### Example 9: Add Image To Container In Flutter

In this example, you will learn to add image from internet using container in flutter.

```
Container(  
  child: Image.network(  
    'https://avatars.githubusercontent.com/u/33576285?v=4',  
  ),  
)
```

### Example 10: Create Square Blue Box

In this example, you will learn to create a square blue box with the text “Hello World” in the center.

```
Container(  
  height: 100.0, // Height of the container  
  width: 100.0, // Width of the container  
  color: Colors.blue, // Background color  
  alignment: Alignment.center, // Align the child to the center  
  child: Text('Hello World'), // Child widget  
  padding: EdgeInsets.all(10.0), // Padding inside the container  
  margin: EdgeInsets.all(20.0), // Margin outside the container  
)
```

## ROW IN FLUTTER

### Row Widget In Flutter

In Flutter, **Row** widget displays its children horizontally. It is very useful when placing UI side by side. This is one of the most used widget in flutter.

#### Example 1: Row In Flutter

In this example, the Row contains three children widgets which are **Icon**, **Text**, and **Icon**.

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(  
    MaterialApp(  
      home: Scaffold(  
        appBar: AppBar(  
          title: const Text('My First App'),  
        ),  
        body: const Row(  
          children: [  
            Icon(Icons.star, size: 50),  
            Text('I am learning flutter'),  
            Icon(Icons.star, size: 50),  
          ],  
        ),  
      ),  
    ),  
  );  
}
```

## Example 2: Create Rating Bar Using Flutter

In this example, you will learn to build star rating bar using row widget.

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row In Flutter'),
        ),
        body: const Row(
          children: [
            Icon(Icons.star, color: Colors.yellow),
            Icon(Icons.star, color: Colors.yellow),
            Icon(Icons.star, color: Colors.yellow),
            Icon(Icons.star, color: Colors.yellow),
            Icon(Icons.star_border, color: Colors.yellow),
          ],
        ),
      ),
    ),
  );
}
```

Property	Description
<b>mainAxisAlignment</b>	Determines how the children are aligned horizontally.
<b>crossAxisAlignment</b>	Determines how the children are aligned vertically.
<b>children</b>	The widgets that are displayed inside the row.

### Row Properties

These are the most common useful properties of the Row widget which helps you control the layout of its children:

MainAxisAlignment Values	Description
<b>MainAxisAlignment.start</b>	Children are aligned at the start of the horizontal axis.
<b>MainAxisAlignment.end</b>	Children are aligned at the end of the horizontal axis.
<b>MainAxisAlignment.center</b>	Children are centered along the horizontal axis.
<b>MainAxisAlignment.spaceBetween</b>	Children have equal spacing between them.
<b>MainAxisAlignment.spaceAround</b>	Children have equal spacing between them, and also the space at the start and the end is divided evenly.
<b>MainAxisAlignment.spaceEvenly</b>	Children have equal spacing before, between, and after them.

## Main Axis In Row

In Row, main axis determines how the children are aligned horizontally. The `mainAxisAlignment` property accepts the following values:

### Example 3: Main Axis Alignment

In this example, you see 5 stars which are in center horizontally. Run this code online and try changing **center** to **start**, **end**, **spaceBetween**, **spaceAround** or **spaceEvenly**.

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

void main() {
```



```

runApp(
  MaterialApp(
    home: Scaffold(
      appBar: AppBar(
        title: const Text('Row In Flutter'),
      ),
      body: const Row(
        // Try replacing "center" with "start", "end", "spaceBetween",
        "spaceAround" or "spaceEvenly"
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(Icons.star, color: Colors.yellow),
          Icon(Icons.star, color: Colors.yellow),
          Icon(Icons.star, color: Colors.yellow),
          Icon(Icons.star, color: Colors.yellow),
          Icon(Icons.star_border, color: Colors.yellow),
        ],
      ),
    ),
  ),
);
}

```

**Note:** By default **mainAxisAlignment** value is **mainAxisAlignment.start**

## Cross Axis In Row

In Row, cross axis determines how the children are aligned vertically. The **crossAxisAlignment** property accepts the following values:

CrossAxisAlignment Values	Description
<b>CrossAxisAlignment.start</b>	Children are aligned at the top of the <b>Row</b> .
<b>CrossAxisAlignment.end</b>	Children are aligned at the bottom of the <b>Row</b> .
<b>CrossAxisAlignment.center</b>	Children are vertically centered in the <b>Row</b> .
<b>CrossAxisAlignment.stretch</b>	Children are forced to fill the available space vertically.
<b>CrossAxisAlignment.baseline</b>	Children are aligned by their baseline (useful for text).

#### Example 4: Cross Axis Alignment

In this example, you see 5 stars which are in center vertically. Run this code online and try changing **center** to **start**, **end**, **stretch**, **baseline**.

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          height: double.infinity,
          child: const Row(
            // Try replacing "center" with "start", "end", "stretch" or
            "baseline"
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star_border, color: Colors.yellow),
            ],
          ),
        ),
      ),
    ),
  );
}
```

```

    ),
  ),
),
);
}

```

**Note:** By default **crossAxisAlignment** value is **crossAxisAlignment.center**

### Example 5: Using Main Axis & Cross Axis Alignment

In this example, you see 5 stars which are in center horizontally and vertically. Run this code online and try changing its main axis and cross axis alignment.

```

import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          height: double.infinity,
          child: const Row(
            // Try replacing "center" with "start", "end", "spaceAround"
            // or "spaceEvenly"
            mainAxisAlignment: MainAxisAlignment.center,
            // Try replacing "center" with "start", "end", "stretch" or
            "baseline"
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star, color: Colors.yellow),
              Icon(Icons.star_border, color: Colors.yellow),
            ],
          ),
        ),
      ),
    ),
  );
}

```

## Overflow Issue

When you put too many widgets inside a Row and they can't fit within the screen, you'll get an overflow error. Here are simple ways to handle this overflow issue in Flutter:

- Wrap in **SingleChildScrollView** widget
- Use **Expanded** or **Flexible** widget

### Example 6: Wrap In SingleChildScrollView

When you have many children widgets, and they can't all fit on the screen, you might want to scroll to see the ones that are out of view. For more see the example below:

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          height: double.infinity,
          child: SingleChildScrollView(
            scrollDirection: Axis.horizontal,
            child: Row(
              // Try replacing "center" with "start", "end",
              "spaceAround", "spaceEvenly" or "spaceBetween"
              mainAxisAlignment: MainAxisAlignment.center,
              // Try replacing "center" with "start", "end", "stretch" or
              "baseline"
              crossAxisAlignment: CrossAxisAlignment.center,
              children: List.generate(
                50, (index) => const Icon(Icons.star, color:
Colors.yellow)),
            ),
          ),
        ),
      ),
    ),
  );
}
```

### Example 7: Using Expanded

In a Row, use Expanded to make a widget take up all the available remaining space within its parent widget. For more, see the example below:

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          height: double.infinity,
          child: Row(
            // Try replacing "center" with "start", "end", "spaceAround",
            // "spaceEvenly" or "spaceBetween"
            mainAxisAlignment: MainAxisAlignment.center,
            // Try replacing "center" with "start", "end", "stretch" or
            // "baseline"
            crossAxisAlignment: CrossAxisAlignment.center,
            children: List.generate(
              50,
              (index) => const Expanded(
                child: Icon(Icons.star, color: Colors.yellow)),
            ),
          ),
        ),
      ),
    );
}
```

### Example 8: Using Flexible

Flexible allows a widget within a Row to fit its content but won't exceed its maximum size, even if there's extra space available. For more, see the example below:

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Row In Flutter'),
        ),
      ),
    ),
  );
}
```

```

body: Container(
  color: Colors.green,
  height: double.infinity,
  child: Row(
    // Try replacing "center" with "start", "end", "spaceAround",
    "spaceEvenly" or "spaceBetween"
    mainAxisAlignment: MainAxisAlignment.center,
    // Try replacing "center" with "start", "end", "stretch" or
    "baseline"
    crossAxisAlignment: CrossAxisAlignment.center,
    children: List.generate(
      50,
      (index) => const Flexible(
        fit: FlexFit.loose,
        child: Icon(Icons.star, color: Colors.yellow))),
    ),
  ),
),
);
}

```

**Note:** **Expanded** forces its child to fill available space, while **Flexible** allows its child to occupy space without necessarily filling it.

## COLUMN IN FLUTTER

### Column Widget In Flutter

In Flutter, **Column** widget displays its children vertically. It is very useful when placing UI top to bottom. This is one of the most used widgets in Flutter.

### Example 1: Column In Flutter

In this example, the Column contains three children widgets which are **Icon**, **Text**, and **Icon**.

```

import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My First App'),
        ),

```

```

        body: const Column(
          children: [
            Icon(Icons.star, size: 50),
            Text('I am learning flutter'),
            Icon(Icons.star, size: 50),
          ],
        ),
      ),
    ),
  );
}

```

## Example 2: Create Container Boxes

In this example, you will learn to build 4 different containers inside the Column.

```

import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Column In Flutter'),
        ),
        body: Column(
          children: [
            Container(height: 100, width: 100, color: Colors.blue),
            const SizedBox(height: 5),
            Container(height: 100, width: 100, color: Colors.blue),
            const SizedBox(height: 5),
            Container(height: 100, width: 100, color: Colors.blue),
            const SizedBox(height: 5),
            Container(height: 100, width: 100, color: Colors.blue),
          ],
        ),
      ),
    ),
  );
}

```

## Column Properties

These are the most common useful properties of the Column widget which help you control the layout of its children:

MainAxisAlignment Values	Description
<b>MainAxisAlignment.start</b>	Children are aligned at the start of the vertical axis.
<b>MainAxisAlignment.end</b>	Children are aligned at the end of the vertical axis.
<b>MainAxisAlignment.center</b>	Children are centered along the vertical axis.
<b>MainAxisAlignment.spaceBetween</b>	Children have equal spacing between them.
<b>MainAxisAlignment.spaceAround</b>	Children have equal spacing between them, and also the space at the start and the end is divided evenly.
<b>MainAxisAlignment.spaceEvenly</b>	Children have equal spacing before, between, and after them.

## Main Axis In Column

In Column, the main axis determines how the children are aligned vertically. The **mainAxisAlignment** property accepts the following values:

### Example 3: Main Axis Alignment

In this example, you see 4 containers which are centered vertically. Run this code online and try changing **center** to **start**, **end**, **spaceBetween**, **spaceAround** or **spaceEvenly**.

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

void main() {
  runApp(
    MaterialApp(
```



```

home: Scaffold(
  appBar: AppBar(
    title: const Text('Column In Flutter'),
  ),
  body: Column(
    // Try replacing "center" with "start", "end", "spaceBetween",
    "spaceAround" or "spaceEvenly"
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Container(height: 100, width: 100, color: Colors.blue),
      const SizedBox(height: 5),
      Container(height: 100, width: 100, color: Colors.blue),
      const SizedBox(height: 5),
      Container(height: 100, width: 100, color: Colors.blue),
      const SizedBox(height: 5),
      Container(height: 100, width: 100, color: Colors.blue),
    ],
  ),
),
);
}

```

**Note:** By default **mainAxisAlignment** value is **mainAxisAlignment.start**

## Cross Axis In Column

In Column, the cross axis determines how the children are aligned horizontally. The **crossAxisAlignment** property accepts the following values:

CrossAxisAlignment Values	Description
<b>CrossAxisAlignment.start</b>	Children are aligned at the left of the <b>Column</b> .
<b>CrossAxisAlignment.end</b>	Children are aligned at the right of the <b>Column</b> .
<b>CrossAxisAlignment.center</b>	Children are vertically centered in the <b>Column</b> .
<b>CrossAxisAlignment.stretch</b>	Children are forced to fill the available space horizontally.
<b>CrossAxisAlignment.baseline</b>	Children are aligned by their baseline (useful for text).

#### Example 4: Cross Axis Alignment

In this example, you see 4 boxes which are centered horizontally. Run this code online and try changing center to start, end, stretch, or baseline.

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Column In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          width: double.infinity,
          child: Column(
            // Try replacing "center" with "start", "end", "stretch", or
            "baseline"
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Container(height: 100, width: 100, color: Colors.blue),
              const SizedBox(height: 5),
              Container(height: 100, width: 100, color: Colors.blue),
              const SizedBox(height: 5),
              Container(height: 100, width: 100, color: Colors.blue),
            ],
          ),
        ),
      ),
    ),
  );
}
```

```

        const SizedBox(height: 5),
        Container(height: 100, width: 100, color: Colors.blue),
      ],
    ),
  ),
),
);
}

```

**Note:** By default **crossAxisAlignment** value is **crossAxisAlignment.center**

### Example 5: Using Main Axis & Cross Axis Alignment

In this example, you see 4 containers which are aligned vertically in the center. Run this code online and try changing its main axis and cross axis alignment.

```

import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Column In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          width: double.infinity,
          child: SingleChildScrollView(
            scrollDirection: Axis.vertical,
            child: Column(
              // Try replacing "center" with "start", "end",
              "spaceAround", "spaceEvenly" or "spaceBetween"
              mainAxisAlignment: MainAxisAlignment.center,
              // Try replacing "center" with "start", "end", "stretch" or
              "baseline"
              crossAxisAlignment: CrossAxisAlignment.center,
              children: List.generate(
                50, (index) => Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Container(height: 100, width: 100, color:
Colors.white),
                )),
            ),
          ),
        ),
      ),
    ),
  ),
);
}

```

```
    ),  
  ),  
);  
}
```

## Overflow Issue

When you put too many widgets inside a Column and they can't fit within the screen, you'll get an overflow error. Here are simple ways to handle this overflow issue in Flutter:

- Wrap in **SingleChildScrollView** widget
- Use **Expanded** or **Flexible** widget

### Example 6: Wrap In SingleChildScrollView

When you have many children widgets, and they can't all fit on the screen, you might want to scroll to see the ones that are out of view. For more see the example below:

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(  
    MaterialApp(  
      home: Scaffold(  
        appBar: AppBar(  
          title: const Text('Column In Flutter'),  
        ),  
        body: Container(  
          color: Colors.green,  
          width: double.infinity,  
          child: SingleChildScrollView(  
            scrollDirection: Axis.vertical,  
            child: Column(  
              // Try replacing "center" with "start", "end",  
"spaceAround", "spaceEvenly" or "spaceBetween"  
              mainAxisAlignment: MainAxisAlignment.center,  
              // Try replacing "center" with "start", "end", "stretch" or  
"baseline"  
              crossAxisAlignment: CrossAxisAlignment.center,  
              children: List.generate(  
                50, (index) => Padding(  
                  padding: const EdgeInsets.all(8.0),  
                  child: Container(height: 100, width: 100, color:  
Colors.white),  
                ),  
            ),  
          ),  
        ),  
      ),  
    ),  
  ),  
}
```

```

    ),
  ),
),
);
}

```

## Example 7: Using Expanded

In a Column, use Expanded to make a widget take up all the available remaining space within its parent widget. For more, see the example below:

```

import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Column In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          width: double.infinity,
          child: Column(
            // Try replacing "center" with "start", "end", "spaceAround",
            "spaceEvenly" or "spaceBetween"
            mainAxisAlignment: MainAxisAlignment.center,
            // Try replacing "center" with "start", "end", "stretch" or
            "baseline"
            crossAxisAlignment: CrossAxisAlignment.center,
            children: List.generate(
              50,
              (index) => Expanded(
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Container(height: 100, width: 100, color:
Colors.white),
                )),
            ),
          ),
        ),
      ),
    );
}

```

## Example 8: Using Flexible

Flexible allows a widget within a Column to fit its content but won't exceed its maximum size, even if there's extra space available. For more, see the example below:

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Column In Flutter'),
        ),
        body: Container(
          color: Colors.green,
          width: double.infinity,
          child: Column(
            // Try replacing "center" with "start", "end", "spaceAround",
            // "spaceEvenly" or "spaceBetween"
            mainAxisAlignment: MainAxisAlignment.center,
            // Try replacing "center" with "start", "end", "stretch" or
            // "baseline"
            crossAxisAlignment: CrossAxisAlignment.center,
            children: List.generate(
              50,
              (index) => Flexible(
                fit: FlexFit.loose,
                child: Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: Container(height: 100, width: 100, color:
Colors.white),
                )),
            ),
          ),
        ),
      ),
    );
}
```

**Note:** **Expanded** forces its child to fill available space, while **Flexible** allows its child to occupy space without necessarily filling it.

## ROW AND COLUMN IN FLUTTER

### Row and Column Flutter

**Row** and **Column** helps you to align its children horizontally and vertically. These widgets are important when designing the user interface for an application.

## Row

Arranges children horizontally.

## Column

Arranges children vertically.

### Example 1: Row & Column

In this example, you will learn to use Row and Column widgets together with a CircleAvatar to create a user profile layout.

```
Row(  
  crossAxisAlignment: CrossAxisAlignment.start,  
  children: <Widget>[  
    CircleAvatar(  
      radius: 35,  
      backgroundImage: NetworkImage(  
        'https://avatars.githubusercontent.com/u/33576285?v=4',  
      ),  
    ),  
    Padding(padding: EdgeInsets.all(2.0)),  
    Column(  
      children: <Widget>[  
        Text(  
          'John Doe',  
          style: TextStyle(  
            fontSize: 20,  
            fontWeight: FontWeight.bold,  
          ),  
        ),  
        Text(  
          'Flutter Developer',  
          style: TextStyle(  
            fontSize: 15,  
          ),  
        ),  
      ],  
    ),  
  ],  
)
```

### Example 2: Login Form

In this example, you will learn how to build a simple login form using Column and Row, along with the TextField for input fields.

```
Column(  
  mainAxisAlignment: MainAxisAlignment.center,
```

```

children: <Widget>[
  const Row(
    children: <Widget>[
      Text('Username: '),
      Expanded(child: TextField()),
    ],
  ),
  const Row(
    children: <Widget>[
      Text('Password: '),
      Expanded(
        child: TextField(
          obscureText: true,
        ),
      ),
    ],
  ),
  Row(
    children: <Widget>[
      Expanded(
        child: MaterialButton(
          color: Colors.blue,
          textColor: Colors.white,
          onPressed: () {},
          child: const Text('Login'),
        ),
      ),
    ],
  ),
],
)

```

### Example 3: Star Pattern

In this example, you will learn how to create a star pattern using Center, Column, and Row widgets, along with the Icon widget.

```

Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      // First row (one star in the middle)
      Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [Icon(Icons.star, color: Colors.blue)],
      ),
      // Second row (one star in the middle)
    ],
  ),
)

```



```

Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [Icon(Icons.star, color: Colors.blue)],
),
// Third row (five stars)
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Icon(Icons.star, color: Colors.blue),
    Icon(Icons.star, color: Colors.blue),
    Icon(Icons.star, color: Colors.blue),
    Icon(Icons.star, color: Colors.blue),
    Icon(Icons.star, color: Colors.blue),
  ],
),
// Fourth row (one star in the middle)
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [Icon(Icons.star, color: Colors.blue)],
),
// Fifth row (one star in the middle)
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [Icon(Icons.star, color: Colors.blue)],
),
],
),
),
),

```

## STACK IN FLUTTER

### Stack In Flutter

Imagine you're making a sandwich. You put one ingredient on top of another, right? In Flutter, the Stack works similarly. It lets you put one thing (like a button or an image) on top of another. It's super useful when you want things to overlap in your app's design.

### When to Use Stack

- **Buttons Over Pictures:** Like a play button on a video thumbnail.
- **Text on Images:** Like captions on photos.
- **Cool Designs:** For when you want to mix things up from the usual top-down look.

**Note:** The first one is at the bottom, and each new one goes on top. In a Stack, your first widget is at the bottom, and others are layered over it.

## Example 1: Basic Stack Implementation

Here's how you can create a basic Stack with overlaid elements:

```
Stack(  
  alignment: Alignment.center,  
  children: [  
    Container(  
      width: 100,  
      height: 100,  
      color: Colors.blue,  
    ),  
    const Text('I'm on top!'),  
  ],  
)
```

## Example 2: Profile Picture with Online Status Indicator

In this example, you will learn to create a circular profile picture with a small green dot at the bottom to indicate the user is online.

```
Stack(  
  alignment: Alignment.bottomRight,  
  children: [  
    const CircleAvatar(  
      radius: 40,  
      backgroundImage: NetworkImage(  
        'https://avatars.githubusercontent.com/u/33576285?v=4'  
      ),  
    ),  
    DecoratedBox(  
      decoration: BoxDecoration(  
        color: Colors.green,  
        shape: BoxShape.circle,  
        border: Border.all(color: Colors.white, width: 3),  
      ),  
      child: const SizedBox(width: 20, height: 20),  
    ),  
  ],  
)
```

## Positioned Widget

The Positioned widget is a special type of widget used inside a Stack widget to control the position of a child widget. It has the following useful properties:

Property	Description
<b>top</b>	Distance from the top edge of the stack.
<b>bottom</b>	Distance from the bottom edge of the stack.
<b>left</b>	Distance from the left edge of the stack.
<b>right</b>	Distance from the right edge of the stack.

You can see the demo of the Positioned widget in example 3 and 4.

### Example 3: Product Image with Discount Badge

In this example, you will learn to create a product image with a discount badge at the top right corner.

```
Stack(
  children: [
    Image.network('https://images.pexels.com/photos/19060954/pexels-photo-19060954/free-photo-of-iphone-15-pro-max.jpeg'),
    Positioned(
      top: 10,
      right: 10,
      child: Container(
        padding: const EdgeInsets.all(8),
        color: Colors.red,
        child: const Text(
          '20% OFF',
          style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
        ),
      ),
    ),
  ],
)
```

### Example 4: Overlay Text on Image

In this example, you will learn to overlay text on an image.

```
Stack(
  children: [
    Image.network(
      'https://images.pexels.com/photos/19060954/pexels-photo-19060954/free-photo-of-iphone-15-pro-max.jpeg'
    ),
  ],
)
```

```

    Positioned(
      bottom: 10,
      left: 10,
      child: DecoratedBox(
        decoration: BoxDecoration(
          color: Colors.black,
          borderRadius: BorderRadius.circular(4),
        ),
        child: const Padding(
          padding: EdgeInsets.all(8),
          child: Text(
            'iPhone 15 Pro Max',
            style: TextStyle(color: Colors.white, fontWeight:
FontWeight.bold),
          ),
        ),
      ),
    ),
  ],
)

```

## CREATE COUNTER APP

### Create Counter App In Flutter

In this section, you will learn how to create a simple counter app in flutter. This app will help you to understand the basic concepts of flutter. So let's start.

#### Step 1: Create Flutter Project

To create a flutter project, open the command prompt and run the following command:

```
flutter create counter_app
```

#### Step 2: Define the Main Entry Point

Open main.dart file and define the main entry point of the app.

```

import 'package:flutter/material.dart';

void main() {
  runApp(const CounterApp());
}

```

#### Step 3: Create the CounterApp Widget

Now, you need to create the CounterApp widget.

```
class CounterApp extends StatelessWidget {
  const CounterApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Counter App',
      home: CounterScreen(),
    );
  }
}
```

#### Step 4: Create the CounterScreen Widget

Now, create the CounterScreen widget.

```
class CounterScreen extends StatefulWidget {
  @override
  _CounterScreenState createState() => _CounterScreenState();
}

class _CounterScreenState extends State<CounterScreen> {
  // Define the counter variable
  int _counter = 0;

  // Define the increment and decrement counter methods
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  void _decrementCounter() {
    setState(() {
      _counter--;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Counter App')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
```

```

        children: [
          Text(
            '$_counter',
            style: const TextStyle(fontSize: 72),
          ),
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              FloatingActionButton(
                onPressed: _incrementCounter,
                tooltip: 'Add',
                child: const Icon(Icons.add),
              ),
              FloatingActionButton(
                onPressed: _decrementCounter,
                tooltip: 'Subtract',
                child: const Icon(Icons.remove),
              ),
            ],
          ),
        ],
      ),
    ),
  );
}

```

## Step 5: Run the App

Now, you can run the app using the following command:

```
flutter run
```

## QUESTIONS FOR PRACTICE 1

### Basic Flutter Practice Questions

1. Create a new Flutter application that displays "Welcome to Flutter" centered on the screen, with a font size of 24, in blue color, and with a yellow background.
2. Describe the difference between a StatelessWidget and a StatefulWidget. Give an example of a situation where you would use each.
3. Write a Flutter layout using a Row to create a horizontal bar of three evenly spaced buttons.
4. Use the Stack widget to overlay text on an image.
5. Create a Flutter application that contains an image of a puppy and a button that increases the size of the puppy (double the size each time the button is pressed).
6. Create a Profile Screen UI Design in Flutter using Row and Column Widgets.

## 2) \_Flutter List and Grid

This section will help you to learn about ListView and GridView in Flutter with the help of real-world examples. Here you will learn the following topics:

- [ListView in Flutter](#),
- [GridView in Flutter](#),
- [ListView.builder in Flutter](#), and
- [GridView.builder in Flutter](#).

### Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

#### LIST VIEW IN FLUTTER

### ListView In Flutter

**ListView** is a type of widget in Flutter used to display a list of items in a scrollable column or row. It's similar to how you scroll through your contacts on a phone or through posts on a social media app. ListView is flexible and can be used for simple lists as well as more complex ones with different types of items.

### Example 1: To Do List App

In this example below, you will learn how to create a simple to-do list app using ListView. Here's how you can make a list with titles and subtitles for each task.

```
ListView(  
  children: [  
    ListTile(  
      title: Text('Go to Gym'),  
      subtitle: Text('Go to Gym at 6:00 AM'),  
    ),  
    ListTile(  
      title: Text('Go to College'),  
      subtitle: Text('Go to College at 8:00 AM'),  
    ),  
    ListTile(  
      title: Text('Go to Office'),  
      subtitle: Text('Go to Office at 11:00 AM'),  
    ),  
    // Add more ListTiles as needed  
  ],  
)
```

```
)
```

**Note:** ListTiles are a convenient way to create lists of items. They provide a leading icon, a title, and a subtitle. You can also add trailing icons if needed.

## Example 2: Contact List App

In this example below, you will learn how to create a simple contact list using ListView with icons, names, numbers, and a call icon.

```
ListView(  
  children: [  
    ListTile(  
      leading: Icon(Icons.person),  
      title: Text('John Doe'),  
      subtitle: Text('555-555-5555'),  
      trailing: Icon(Icons.call),  
    ),  
    ListTile(  
      leading: Icon(Icons.person),  
      title: Text('Jane Doe'),  
      subtitle: Text('555-555-5555'),  
      trailing: Icon(Icons.call),  
    ),  
    ListTile(  
      leading: Icon(Icons.person),  
      title: Text('John Smith'),  
      subtitle: Text('555-555-5555'),  
      trailing: Icon(Icons.call),  
    ),  
  ],  
)
```

## Example 3: Image Gallery App

In this example below, you will learn how to create a photo gallery app using ListView. You can change the scroll direction to horizontal to create a horizontal photo gallery.

```
ListView(  
  // try changing to `scrollDirection: Axis.horizontal` to see horizontal  
  list  
  scrollDirection: Axis.horizontal,  
  children: [  
    Image.network('https://picsum.photos/250?image=9'),  
    Image.network('https://picsum.photos/250?image=10'),  
    Image.network('https://picsum.photos/250?image=11'),  
    Image.network('https://picsum.photos/250?image=12'),  
  ],  
)
```



```

        Image.network('https://picsum.photos/250?image=13'),
        Image.network('https://picsum.photos/250?image=14'),
        Image.network('https://picsum.photos/250?image=15'),
    ],
)

```

### Example 4: Reverse List

In this example below, you will learn how to reverse a list using ListView. You can use the **reverse** property to reverse the order of the list.

```

ListView(
  reverse: true,
  children: [
    ListTile(
      title: Text('Go to Gym'),
      subtitle: Text('Go to Gym at 6:00 AM'),
    ),
    ListTile(
      title: Text('Go to College'),
      subtitle: Text('Go to College at 8:00 AM'),
    ),
    ListTile(
      title: Text('Go to Office'),
      subtitle: Text('Go to Office at 11:00 AM'),
    ),
    // Add more ListTiles as needed
  ],
)

```

### Challenge

Create a restaurant app that displays a list of dishes. Each dish should have a name, description, and price.

## GRIDVIEW IN FLUTTER

### GridView In Flutter

In Flutter, **GridView** is a versatile widget that allows the creation of grid layouts. It's an essential widget when you need to display items in a two-dimensional list. GridView is ideal for situations where you want to present data in a visually appealing and organized manner, such as in photo galleries, product listings, or dashboard menus.

### Example 1: Photo Gallery App

In this example, you will build a simple photo gallery app using GridView, demonstrating how to display a collection of images in a grid format.

```
GridView.count(  
  crossAxisCount: 3,  
  children:[  
    Image.network('https://picsum.photos/200?image=25'),  
    Image.network('https://picsum.photos/200?image=26'),  
    Image.network('https://picsum.photos/200?image=27'),  
    Image.network('https://picsum.photos/200?image=28'),  
    Image.network('https://picsum.photos/200?image=29'),  
    Image.network('https://picsum.photos/200?image=30'),  
    // Add more images as needed  
  ],  
)
```

## Example 2: Ecommerce Product Listing Page

In this example below, you will learn how to create an ecommerce product listing page using GridView, displaying products in a grid with their images and details.

```
GridView.count(  
  crossAxisCount: 2,  
  children:[  
    Card(  
      child: Column(  
        crossAxisAlignment: CrossAxisAlignment.stretch,  
        children:[  
          Image.network('https://picsum.photos/200?image=25', height: 150,  
width: 150),  
          const Text('Product 1'),  
          const Text('Price: \">$100'),  
        ],  
      ),  
    ),  
    Card(  
      child: Column(  
        crossAxisAlignment: CrossAxisAlignment.stretch,  
        children:[  
          Image.network('https://picsum.photos/200?image=25', height: 150,  
width: 150),  
          const Text('Product 2'),  
          const Text('Price: \ \"$150'),  
        ],  
      ),  
    ),  
    // Add more product cards as needed  
  ],  
)
```

```
],  
)
```

### Example 3: Simple Home Dashboard App With Menu Categories

In this example below, you will learn how to create a simple home dashboard app with menu categories using GridView.

```
GridView.count(  
  crossAxisCount: 2,  
  children: [  
    MaterialButton(  
      onPressed: () {/* Handle click */},  
      child: const Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: [  
          Icon(Icons.category),  
          Text('Category 1'),  
        ],  
      ),  
    ),  
    MaterialButton(  
      onPressed: () {/* Handle click */},  
      child: const Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: [  
          Icon(Icons.category),  
          Text('Category 2'),  
        ],  
      ),  
    ),  
    // Add more categories as needed  
  ],  
)
```

### Challenge

Create a simple photo gallery app using GridView. Display images in a grid format with 3 columns and 5 rows.

[LISTVIEW.BUILDER IN FLUTTER](#)

### Introduction

**ListView.builder** is a highly efficient way to create lists that display a large number of items. It creates items as they're scrolled onto the screen, which is ideal for lists with a large number of items.

**Note:** Before learning **ListView.builder**, make sure you understand the basics of **ListView**.

### Example 1: Basic ListView.builder

In this example below, you will learn how to create a simple list of items using ListView.builder.

```
ListView.builder(  
  itemCount: 20,  
  itemBuilder: (BuildContext context, int index) {  
    return ListTile(  
      title: Text('Item $index'),  
    );  
  },  
)
```

### Example 2: To-Do List App

In this example below, you will learn how to create a simple to-do list app using ListView.builder.

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  final List<String> tasks = [  
    'Go to Gym',  
    'Go to College',  
    'Go to Office',  
    // Add more tasks  
  ];  
  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: Scaffold(  
        appBar: AppBar(title: const Text('To-Do List')),  
        body: ListView.builder(  
          itemCount: tasks.length,
```

```

        itemBuilder: (BuildContext context, int index) {
            return ListTile(
                title: Text(tasks[index]),
            );
        },
    ),
);
}
}

```

### Example 3: To-Do List App Using Using a Data Model

In this example below, you will learn how to create a to-do list app using Data Model and ListView.builder.

#### Step 1: Define the Data Model

First, we define a simple data model for a task.

```

class Task {
    final String title;
    final String subtitle;

    Task({
        required this.title,
        required this.subtitle,
    });
}

```

**Note:** Data Model is a simple class that defines the structure of your data. This allows for a more structured and detailed representation of your data.

#### Step 2: Create Sample Data

Next, we create a list of sample tasks.

```

List<Task> tasks = [
    Task(
        title: 'Go to Gym',
        subtitle: 'Go to Gym at 6:00 AM',
    ),
    Task(
        title: 'Go to College',
        subtitle: 'Go to College at 8:00 AM',
    ),
]

```

```

Task(
  title: 'Go to Office',
  subtitle: 'Go to Office at 11:00 AM',
),
// Add more sample tasks
];

```

### Step 3: Use ListView.builder

Now, we use **ListView.builder** to build the list.

```

ListView.builder(
  itemCount: tasks.length,
  itemBuilder: (BuildContext context, int index) {
    return ListTile(
      title: Text(tasks[index].title),
      subtitle: Text(tasks[index].subtitle),
    );
  },
)

```

### Example 4: Social Media Feed App

In this example below, you will learn how to create a social media feed app using **ListView.builder**. Click on run online button to see the output.

#### Step 1: Define the Data Model

First, we define a simple data model for a post.

```

class Post {
  final String username;
  final String imageUrl;
  final String timestamp;
  final String contentText;
  final String imageUrl;

  Post({
    required this.username,
    required this.imageUrl,
    required this.timestamp,
    required this.contentText,
    required this.imageUrl,
  });
}

```

## Step 2: Create Sample Data

Next, we create a list of sample posts.

```
List<Post> posts = [
  Post(
    username: 'John Doe',
    imageUrl: 'https://picsum.photos/250?image=237',
    timestamp: '2h',
    contentText: 'Enjoying the beautiful sunset at the beach!',
    contentImageUrl: 'https://picsum.photos/250?image=51',
  ),
  Post(
    username: 'Mark Doe',
    imageUrl: 'https://picsum.photos/250?image=238',
    timestamp: '1d',
    contentText: 'Just got back from a fun vacation in the mountains.',
    contentImageUrl: 'https://picsum.photos/250?image=52',
  ),
  // Add more sample posts
];
```

## Step 3: Build the ListView

Now, we use **ListView.builder** to build the list.

```
ListView.builder(
  itemCount: posts.length,
  itemBuilder: (BuildContext context, int index) {
    return Card(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          ListTile(
            leading: CircleAvatar(backgroundImage:
NetworkImage(posts[index].imageUrl)),
            title: Text(posts[index].username),
            subtitle: Text(posts[index].timestamp),
          ),
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: Text(posts[index].contentText),
          ),
          Image.network(posts[index].contentImageUrl),
        ],
      ),
    );
  });
```

```
},  
)
```

## Challenge

Create app using `ListView.builder` to display a scrollable list containing the names of your 10 friends, with each name as a separate item.

## GRIDVIEW.BUILDER IN FLUTTER

### Introduction

**GridView.builder** is an efficient way to create grid layouts that display a large number of items. It dynamically creates grid items as they become visible on the screen, making it ideal for grid layouts with numerous items.

**Note:** Before learning **GridView.builder**, make sure you understand the basics of **GridView**.

### Example 1: Basic GridView.builder

In this example below, you will learn how to create a basic grid layout using `GridView.builder`.

```
GridView.builder(  
  gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(  
    crossAxisCount: 2), // Number of columns  
  itemCount: 20,  
  itemBuilder: (BuildContext context, int index) {  
    return GridTile(  
      child: Center(  
        child: Text('Item $index'),  
      ),  
    );  
  },  
)
```

### Example 2: Photo Gallery App

In this example below, you will learn how to create a simple photo gallery app using `GridView.builder`.

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



```

}

class MyApp extends StatelessWidget {
  final List<String> imageUrls = [
    'https://picsum.photos/250?image=237',
    'https://picsum.photos/250?image=238',
    'https://picsum.photos/250?image=239',
    'https://picsum.photos/250?image=240',
    'https://picsum.photos/250?image=241',
    'https://picsum.photos/250?image=242',
    'https://picsum.photos/250?image=243',
    'https://picsum.photos/250?image=244',
    // Add more image URLs
  ];

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Photo Gallery')),
        body: GridView.builder(
          gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
            crossAxisCount: 3), // Number of columns
          itemCount: imageUrls.length,
          itemBuilder: (BuildContext context, int index) {
            return Image.network(imageUrls[index]);
          },
        ),
      ),
    );
  }
}

```

### Example 3: Create Tic-Tac-Toe Board

In this example below, you will learn how to create a Tic-Tac-Toe board using GridView.builder.

```

GridView.builder(
  gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 3), // Number of columns
  itemCount: 9,
  itemBuilder: (BuildContext context, int index) {
    return Container(
      decoration: BoxDecoration(
        border: Border.all(color: Colors.black),
      ),
    ),
  ),
)

```

```

        child: Center(
          child: index % 2 == 0
            ? const Text('X')
            : const Text('O'),
        ),
      ),
    ),
  },
),
)

```

## Example 4: Product Catalog App Using Data Model

In this example below, you will learn how to create a product catalog app using a Data Model and GridView.builder.

### Step 1: Define the Data Model

First, we define a simple data model for a product.

```

class Product {
  final String title;
  final String subtitle;
  final String imageUrl;

  Product({
    required this.title,
    required this.subtitle,
    required this.imageUrl,
  });
}

```

### Step 2: Create a List of Products

Create a list of sample products.

```

List<Product> products = [
  Product(
    title: 'Product 1',
    subtitle: 'Subtitle 1',
    imageUrl: 'https://picsum.photos/250?image=237',
  ),
  Product(
    title: 'Product 2',
    subtitle: 'Subtitle 2',
    imageUrl: 'https://picsum.photos/250?image=238',
  ),
  Product(

```

```

        title: 'Product 3',
        subtitle: 'Subtitle 3',
        imageUrl: 'https://picsum.photos/250?image=239',
    ),
    // Add more sample products
];

```

### Step 3: Build the GridView

Use GridView.builder to create the product grid.

```

GridView.builder(
  gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
    crossAxisCount: 2), // Number of columns
  itemCount: products.length,
  itemBuilder: (BuildContext context, int index) {
    return Card(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Image.network(products[index].imageUrl),
          Text(products[index].title),
          Text(products[index].subtitle),
        ],
      ),
    );
  },
);

```

### Challenge

Create chess board using GridView.builder. Make sure to use the following colors for the tiles:

```

- White: #F0D9B5
- Black: #B58863

```

## CREATE TIC TAC TOE GAME

### Tic Tac Toe Game Using Flutter

In this guide, we'll create a simple Tic Tac Toe game using Flutter. We'll use a `GridView.builder` to create the game grid and manage the game state.

#### 1. Setup and Project Creation

First, ensure Flutter is installed on your machine. Create a new Flutter project with the following command:

```
flutter create tic_tac_toe_game
```

## 2. Writing the Code

Open the main.dart file. This is where your Flutter application starts. Replace the existing code with:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Tic Tac Toe',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: TicTacToePage(),
    );
  }
}

class TicTacToePage extends StatefulWidget {
  @override
  _TicTacToePageState createState() => _TicTacToePageState();
}

class _TicTacToePageState extends State<TicTacToePage> {
  late List<String> board;
  late String currentPlayer;
  late String winner;
  late bool isDraw;

  @override
  void initState() {
    super.initState();
    _initializeGame();
  }

  void _initializeGame() {
    board = List.generate(9, (_) => '');
    currentPlayer = 'X';
  }
}
```

```

        winner = '';
        isDraw = false;
    }

    void _handleTap(int index) {
        if (board[index] != '' || winner != '') return; // Prevent overwriting
        a cell and ignore taps after game over

        setState(() {
            board[index] = currentPlayer;
            if (_checkWinner(currentPlayer)) {
                winner = currentPlayer;
            } else if (_checkDraw()) {
                isDraw = true;
            } else {
                currentPlayer = currentPlayer == 'X' ? 'O' : 'X';
            }
        });
    }

    bool _checkWinner(String player) {
        // Check rows, columns and diagonals for a win
        for (int i = 0; i < 3; i++) {
            if (board[i * 3] == player && board[i * 3 + 1] == player && board[i
* 3 + 2] == player) {
                return true;
            }
            if (board[i] == player && board[i + 3] == player && board[i + 6] ==
player) {
                return true;
            }
        }
        if (board[0] == player && board[4] == player && board[8] == player) {
            return true;
        }
        if (board[2] == player && board[4] == player && board[6] == player) {
            return true;
        }
        return false;
    }

    bool _checkDraw() {
        return board.every((cell) => cell != '');
    }

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

```

```

appBar: AppBar(
  title: const Text('Tic Tac Toe'),
),
body: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    winner != ''
      ? Text('Winner: $winner', style: const TextStyle(fontSize:
30))
      : isDraw
        ? const Text('Draw', style: TextStyle(fontSize: 30))
        : Text('Current Player: $currentPlayer', style: const
TextStyle(fontSize: 30)),
    Expanded(
      child: GridView.builder(
        gridDelegate: const
SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 3,
        ),
        itemCount: 9,
        itemBuilder: (context, index) {
          return GridTile(
            child: Container(
              margin: const EdgeInsets.all(10.0),
              decoration: BoxDecoration(
                border: Border.all(color: Colors.black),
              ),
              child: MaterialButton(
                onPressed: () => _handleTap(index),
                child: Text(board[index], style: const
TextStyle(fontSize: 40)),
              ),
            ),
          );
        },
      ),
    MaterialButton(
      child: const Text('Restart Game'),
      onPressed: () {
        setState(() {
          _initializeGame();
        });
      },
    ),
  ],
),
);

```

```
}  
}
```

## Run the App

To run the application, use the **flutter run** command, or press **F5** in Visual Studio Code to start debugging.

## QUESTIONS FOR PRACTICE 2

### List and Grid Flutter Practice Questions

1. Create a Flutter application that displays a list of 10 images of puppies in a ListView.
2. Create a Todo List application that displays a list of todos in a ListView.
3. Create a Flutter application that shows a Chessboard using GridView.
4. Create a simple ListView in Flutter that displays a list of ten strings. Each item in the list should be displayed as a text widget.
5. Display a list of numbers from 1 to 50. Each number should be shown in a separate list item.
6. Create a ListView.builder that displays a list of names. If a name starts with the letter 'A', it should be displayed with a green color; otherwise, it should be displayed in red.
7. Create a horizontal ListView that displays a list of images. Ensure that the ListView scrolls horizontally.
8. Implement a ListView inside another ListView. The outer ListView should have three items, and each item should contain an inner ListView with five text items.
9. Implement a GridView with a two-column layout. Populate it with 20 square containers, each with a unique color or decoration.
10. Use GridView.builder to create a grid of items where each item has a different aspect ratio. For example, alternate between items with a 1:1 and 2:1 aspect ratio.
11. Create a GridView where the number of columns changes based on the device orientation (portrait or landscape).
12. Implement an infinite scrolling ListView using ListView.builder, where more items are loaded when the user scrolls to the end of the list.
13. Create a GridView.builder that displays a grid of items. When an item is tapped, it should display an alert dialog with the item's index or other details.

## 3) \_Working With Assets In Flutter

This section will help you to learn how to use assets such as images, fonts, JSON file, icons and other files in your flutter application. Here you will learn the following topics:

- [Assets in Flutter,](#)
- [Add Images in Flutter,](#)
- [Add Local Images in Flutter,](#)
- [Add Network Images in Flutter,](#)
- [Add Cached Images in Flutter,](#)
- [Add Local Fonts in Flutter,](#)
- [Add Google Fonts in Flutter,](#)

- [Add Local JSON in Flutter](#),
- [Add Online JSON in Flutter](#), and
- [Create Quotes App in Flutter](#).

## Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

## ASSETS IN FLUTTER

### Introduction to Assets

In the simplest terms, assets are files that are pack together with your Flutter application. These files can include a variety of types:

Type	Description
Images	Common formats like JPEG, PNG, GIF, and more.
Fonts	Custom fonts to give your app a unique typography.
Audio	Sound effects and music files.
Videos	Small video files used in the app.
Data	JSON, XML, or other formatted data files.

**Note:** In upcoming sections, you will learn how to use images, fonts, and other files in flutter.

### Why Are Assets Important?

Assets are an important part of any application because of the following reasons:

- **Enhanced User Experience:** Visually rich elements like custom images and fonts help in making your app stand out.
- **Offline Availability:** Since assets are bundled with the app, they're available even when the user is offline.
- **Performance:** Bundled assets can be quicker to load compared to fetching resources over a network.

### Best Practices for Managing Assets

Here are some best practices for managing assets in your app:



- **Optimize Asset Size:** Large assets can increase the size of your app. Optimize images and other files to balance quality and size.
- **Organize Wisely:** Maintain a clear structure in your assets folder. Group similar types of assets together.
- **Use Descriptive Names:** Name your assets descriptively to make them easier to identify and manage.

## Best Way to Organize Assets

The best way to organize assets is to group them by type. For example, you can create separate folders for images, fonts, audio, and other files. This will make it easier to manage your assets as your app grows.

```
assets/  
  fonts/  
    Roboto-Regular.ttf  
    Roboto-Bold.ttf  
  images/  
    logo.png  
    banner.png  
  audio/  
    sound.mp3  
  data/  
    data.json
```

## USE IMAGES IN FLUTTER

### Add Images in Flutter

Images are a most common type of asset in mobile applications. Here you will learn how to include images in your flutter application. You can use images in your application in following ways:

- **Local Images**
- **Network Images**
- **Cached Images**

**Note:** You can use different image format like: PNG, JPG, JPEG, GIF, webP, SVG, etc.

### Local Images

Local images are images that are stored locally in your app. They don't need the internet to work and load faster. [Click here](#) to learn it practically.

### Network Images

Network images are images that are fetched and displayed from the internet in real-time. [Click here](#) to learn it practically.

## Cached Images

Cached images are images that are stored locally after the first download. [Click here](#) to learn it practically.

## USE LOCAL IMAGES IN FLUTTER

## Local Images

Local images are pictures stored in your app. They don't need the internet to work and load faster. These are ideal for images that don't change, like logos and icons.

## How to Use Local Images

### Step 1: Create an Assets Folder

To use local image, first download [it from here](#), then create an 'assets' folder in your project and add the downloaded image to this folder.

### Step 2: Add Images to the pubspec.yaml File

Now, you need to add images to the pubspec.yaml file. Go to your pubspec.yaml file and add the following code in it:

```
flutter:  
  assets:  
    - assets/learn_flutter.jpg
```

Save your pubspec.yaml file.

### Step 3: Use the Images

To display an image in Flutter, you need to use the Image widget. You can use the Image widget in two ways:

```
Image.asset(  
  'assets/learn_flutter.jpg',  
  width: 150,  
  height: 150,  
)  
Image(  
  image: AssetImage('assets/learn_flutter.jpg'),
```

```
width: 150,  
height: 150,  
)
```

## Example 1: Profile App

In this example, we will create a profile app and add a local image to it.

```
import 'package:flutter/material.dart';  
  
void main() {  
  runApp(const ProfileApp());  
}  
  
class ProfileApp extends StatelessWidget {  
  const ProfileApp({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      title: 'Profile App',  
      home: ProfileScreen(),  
    );  
  }  
}  
  
class ProfileScreen extends StatelessWidget {  
  const ProfileScreen({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Profile App'),  
      ),  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: [  
            const CircleAvatar(  
              backgroundImage: AssetImage('assets/learn_flutter.jpg'),  
              radius: 50,  
            ),  
            const SizedBox(height: 10),  
            const Text(  
              'Flutter Tutorial',  
              style: TextStyle(  
                color: Colors.purple,  
                fontSize: 16,  
                fontWeight: FontWeight.bold,  
              ),  
            ),  
          ],  
        ),  
      ),  
    );  
  }  
}
```

```

        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
    ),
    const SizedBox(height: 10),
    const Text(
      'https://flutter-tutorial.net',
      style: TextStyle(
        fontSize: 16,
        color: Colors.blue,
      ),
    ),
  ],
),
),
);
}
}

```

## USE INTERNET IMAGES IN FLUTTER

### Network Images

Network images are images that are fetched and displayed from the internet in real-time. They are important for applications that require constantly updated images, such as social media feeds, news apps, or e-commerce platforms.

### How to Use Network Images

To display an image in Flutter, you need to use the Image widget. You can use the Image widget in two ways:

```

Image.network(
  'https://picsum.photos/250?image=9',
  width: 150,
  height: 150,
)
Image(
  image: NetworkImage('https://picsum.photos/250?image=9'),
  width: 150,
  height: 150,
)

```

### Example 1: Profile App

In this example, we will create a profile app and add a network image to it.

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

void main() {
  runApp(const ProfileApp());
}

class ProfileApp extends StatelessWidget {
  const ProfileApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: 'Flutter Tutorial',
      home: ProfileScreen(),
    );
  }
}

class ProfileScreen extends StatelessWidget {
  const ProfileScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Learn App Development'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.network("https://flutter-tutorial.net/images/learn_flutter.jpg"),
            const SizedBox(height: 10),
            const Text(
              'Flutter Tutorial',
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
              ),
            ),
            const SizedBox(height: 10),
            const Text(
              'https://flutter-tutorial.net',
              style: TextStyle(
                fontSize: 16,
                color: Colors.blue,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```

```
    ),  
  ],  
),  
),  
);  
}  
}
```

## USE CACHED IMAGES IN FLUTTER

### Introduction

Cached images are images that are stored locally after the first download. Here are some of the benefits of using cached images in your Flutter application:

- **Faster Loading Times:**
- **Less Network Usage:**
- **Improved User Experience:**

### How to Use Cached Images in Flutter

To use the `cached_network_image` package, you need to add **cached\_network\_image** package in your flutter project. Open your terminal and go to your project directory and execute the following command:

```
flutter pub add cached_network_image
```

### Import the Package

Now, you need to import the `cached_network_image` package in your flutter application. To import the `cached_network_image` package, open your `main.dart` file and add the following import statement:

```
import 'package:cached_network_image/cached_network_image.dart';
```

### Use the Package

Now, you can use the `cached_network_image` package in your flutter application. To use the `cached_network_image` package, open your `main.dart` file and add the following code in it:

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

```

    runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Use Cached Images in Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Use Cached Images in Flutter'),
        ),
        body: Center(
          child: CachedNetworkImage(
            imageUrl: 'https://picsum.photos/250?image=9',
            placeholder: (context, url) => CircularProgressIndicator(),
            errorWidget: (context, url, error) => Icon(Icons.error),
          ),
        ),
      ),
    );
  }
}

```

In this code:

- **imageUrl** is the URL of the image you want to cache.
- **placeholder** is a widget displayed while the image is loading.
- **errorWidget** is displayed in case of any loading error.

## USE LOCAL FONTS IN FLUTTER

### Introduction

In this section, you will learn to use local fonts in your flutter application with the help of real-world example. Here are the steps to use local fonts in your app:

#### Step 1: Add the Font Files

To use local fonts in your app, the first step is to add the font files in your flutter project. You can download sample font files from [here](#) or you can use your own font files.

**Note:** Before downloading the font files, make sure you have proper license to use it.

#### Step 2: Add Font Files to Your Project

In project folder, create a new folder called **assets** and inside **assets** folder, create another folder called **fonts**. Now, place all the **font files** inside the **fonts** folder. Here is preview of the folder structure:

```
fonts_in_flutter/
├── assets/
│   └── fonts/
│       ├── Roboto-Black.ttf
│       ├── Roboto-BlackItalic.ttf
│       ├── Roboto-Bold.ttf
│       ├── Roboto-BoldItalic.ttf
│       ├── Roboto-Italic.ttf
│       ├── Roboto-Light.ttf
│       ├── Roboto-LightItalic.ttf
│       ├── Roboto-Medium.ttf
│       ├── Roboto-MediumItalic.ttf
│       ├── Roboto-Regular.ttf
│       ├── Roboto-Thin.ttf
│       └── Roboto-ThinItalic.ttf
└── lib/
    └── main.dart
```

### Step 3: Add Font Files to pubspec.yaml File

Now, you need to add font files to the pubspec.yaml file. Go to your pubspec.yaml file and add the following code in it:

```
flutter:
  fonts:
    - family: Roboto
      fonts:
        - asset: assets/fonts/Roboto-Black.ttf
        - asset: assets/fonts/Roboto-BlackItalic.ttf
          style: italic
        - asset: assets/fonts/Roboto-Bold.ttf
        - asset: assets/fonts/Roboto-BoldItalic.ttf
          style: italic
        - asset: assets/fonts/Roboto-Italic.ttf
          style: italic
        - asset: assets/fonts/Roboto-Light.ttf
        - asset: assets/fonts/Roboto-LightItalic.ttf
          style: italic
        - asset: assets/fonts/Roboto-Medium.ttf
        - asset: assets/fonts/Roboto-MediumItalic.ttf
          style: italic
```



```
- asset: assets/fonts/Roboto-Regular.ttf
- asset: assets/fonts/Roboto-Thin.ttf
- asset: assets/fonts/Roboto-ThinItalic.ttf
  style: italic
```

## Step 4: Use the Fonts

To use the fonts, you need to use the **TextStyle** widget. Here is how you can use the fonts:

```
Text(
  'I am using local font',
  style: TextStyle(
    fontFamily: 'Roboto',
    fontSize: 20,
    fontWeight: FontWeight.w900,
  ),
)
```

## Supporting Source Code

If you face any issue while using the fonts, then you can check [this link](#) for the complete source code.

## Challenge

Create profile app with your photo, name, address, and phone number. Use local fonts to style the text.

[USE GOOGLE FONTS IN FLUTTER](#)

## Using Google Fonts in Flutter App

In this section, you will learn how to use Google fonts in your flutter application with the help of examples.

### Use Google Fonts in Flutter

To use Google fonts in your flutter application, follow the below steps:

1. Add [google\\_fonts](#) package in your flutter project.
2. Use google fonts in your flutter application.

### Add google\_fonts Package

To use Google fonts in your application, you need to add the [google\\_fonts](#) plugin. Open terminal at your project folder and execute the following command:

```
flutter pub add google_fonts
```

## Run Pub Get

After saving the pubspec.yaml file, run pub get in your terminal to fetch the package:

```
flutter pub get
```

## Import the google\_fonts Package

Now, you need to import the google\_fonts package in your flutter application. To import the google\_fonts package, open your main.dart file and add the following import statement:

```
import 'package:google_fonts/google_fonts.dart';
```

## Use the Google Fonts

Now, you can use the Google fonts in your flutter application. To use the Google fonts, open your main.dart file and add the following code in it:

```
Text(  
  'Hello I am John Doe',  
  style: GoogleFonts.roboto(  
    fontSize: 20,  
    fontWeight: FontWeight.w500,  
    color: Colors.blue,  
  ),  
)
```

## Specifying Font Weights and Styles

If you want to use specific font weights or styles that are not the default, you can specify them when you use the font.

```
style: GoogleFonts.roboto(  
  fontWeight: FontWeight.w700, // Bold  
  fontStyle: FontStyle.italic, // Italic  
)
```

## Full Example

To use the Google fonts in your flutter application, open your main.dart file and add the following code in it:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Use Google Fonts in Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Google Fonts in Flutter'),
        ),
        body: Center(
          child: Text(
            'Hello I am John Doe',
            style: GoogleFonts.roboto(
              fontSize: 20,
              fontWeight: FontWeight.w500,
              color: Colors.blue,
            ),
          ),
        ),
      ),
    );
  }
}
```

## USE LOCAL JSON IN FLUTTER

### Introduction

In this section, you will learn how to use local JSON in your flutter application with the help of real-world example. Before we start, make sure you have basic knowledge of JSON. If you don't know start from [here](#).

### Example 1: Profile App

In this example, you will learn to create a profile app using local JSON. Here are the steps to create a app:

## Step 1: Add the JSON File

To use local JSON in your app, the first step is to add the JSON file in your project folder/**assets/data**. You can download sample JSON file from [here](#) or you can use your own JSON file.

## Step 2: Add JSON File to Your Project

In project folder, create a new folder called **assets** and inside **assets** folder, create another folder called **data**. Now, place the **JSON file** inside the **data** folder. Here is preview of the folder structure:

```
json_in_flutter/  
├── assets/  
│   └── data/  
│       └── info.json  
└── lib/  
    └── main.dart
```

## Step 3: Add JSON File to pubspec.yaml File

Now, you need to add JSON file to the pubspec.yaml file. Go to your pubspec.yaml file and add the following code in it:

```
flutter:  
  assets:  
    - assets/data/info.json
```

## Step 4: Create Model Class

Now, you need to create a model class for the JSON data. Create a new file called **person.dart** inside **lib** folder and add the following code in it:

```
class Person {  
  final String name;  
  final String address;  
  final int age;  
  final String image;  
  final String description;  
  
  // Constructor  
  Person(  
    {required this.name,  
    required this.address,
```

```

        required this.age,
        required this.image,
        required this.description});

// Convert JSON to Person Object
factory Person.fromJson(Map<String, dynamic> json) {
  return Person(
    name: json['name'],
    address: json['address'],
    age: json['age'],
    image: json['image'],
    description: json['description'],
  );
}
}

```

## Step 5: Load and Decode the JSON File

Now create a new file called **localservice.dart** inside **lib** folder and add the following code in it:

```

import 'dart:convert';
import 'package:flutter/services.dart';
import 'person.dart';

class LocalService {
  // Load and decode the JSON File
  Future<String> _loadPersonAsset() async {
    return await rootBundle.loadString('assets/data/info.json');
  }

  // Load and decode the JSON File
  Future<Person> loadPerson() async {
    String jsonString = await _loadPersonAsset();
    // json.decode() is used to convert JSON String to JSON Map
    final jsonResponse = json.decode(jsonString);
    return Person.fromJson(jsonResponse);
  }
}

```

## Step 6: Display the JSON Data

Now, you can use the **LocalService** class to display the JSON data. Here is how you can use it on your **main.dart** file:

```

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

```

```

import 'person.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    const appTitle = 'JSON in Flutter';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ),
        body: FutureBuilder(
          future: LocalService().loadPerson(),
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              Person person = snapshot.data as Person;
              return Center(
                child: Padding(
                  padding: const EdgeInsets.all(16.0),
                  child: Card(
                    elevation: 4.0,
                    child: Padding(
                      padding: const EdgeInsets.all(16.0),
                      child: Column(
                        mainAxisAlignment: MainAxisAlignment.min,
                        children: [
                          CircleAvatar(
                            backgroundImage: NetworkImage(person.image),
                            radius: 50.0,
                          ),
                          const SizedBox(height: 10),
                          Text(
                            person.name,
                            style: const TextStyle(
                              fontSize: 24,
                              fontWeight: FontWeight.bold,
                            ),
                          ),
                          const SizedBox(height: 5),
                          Text(

```

```

        person.address,
        style: TextStyle(
          fontSize: 16,
          color: Colors.grey[600],
        ),
      ),
      const SizedBox(height: 5),
      Text(
        'Age: ${person.age}',
        style: TextStyle(
          fontSize: 16,
          color: Colors.grey[800],
        ),
      ),
      const SizedBox(height: 10),
      Text(
        person.description,
        textAlign: TextAlign.center,
        style: const TextStyle(
          fontSize: 16,
          fontStyle: FontStyle.italic,
        ),
      ),
    ],
  ),
),
),
),
),
),
),
),
);
} else {
  return const Center(child: CircularProgressIndicator());
}
},
),
),
);
}
}

```

## Supporting Source Code

If you face any issue while using the fonts, then you can check [this link](#) for the complete source code.

## Challenge

Create quote app using local JSON. Use [this JSON file](#) to display the quotes.

## USE ONLINE JSON IN FLUTTER

### Introduction

In this section, you will learn how to use online JSON in your flutter application with the help of real-world example. Before we start, make sure you have basic knowledge of JSON. If you don't know start from [here](#).

### Example 1: Profile App

In this example, you will learn to create a profile app using online JSON. Here are the steps to create a app:

#### Step 1: Install http Package

To use online JSON in your app, the first step is to add the http package in your project. Open terminal at your project folder and execute the following command:

```
flutter pub add http
```

#### Step 2: Import the http Package

Now, you need to import the http package in your flutter application. To import the http package, open your main.dart file and add the following import statement:

```
import 'package:http/http.dart' as http;
```

#### Step 3: Create Model Class

Now, you need to create a model class for the JSON data. Create a new file called **person.dart** inside **lib** folder and add the following code in it:

```
class Person {
  final String name;
  final String address;
  final int age;
  final String image;
  final String description;

  // Constructor
  Person(
    {required this.name,
    required this.address,
    required this.age,
    required this.image,
    required this.description});
```



```
// Convert JSON to Person Object
factory Person.fromJson(Map<String, dynamic> json) {
  return Person(
    name: json['name'],
    address: json['address'],
    age: json['age'],
    image: json['image'],
    description: json['description'],
  );
}
```

## Step 4: Load and Decode the JSON File

Now create a new file called **onlineservice.dart** inside **lib** folder and add the following code in it:

```
import 'dart:convert';
import 'person.dart';

class OnlineService {
  // Load and decode the JSON File
  Future<Person> loadPerson() async {
    final response = await
http.get(Uri.parse('https://jsonguide.technologychannel.org/info.json'));
    if (response.statusCode == 200) {
      // json.decode() is used to convert JSON String to JSON Map
      final jsonResponse = json.decode(response.body);
      return Person.fromJson(jsonResponse);
    } else {
      throw Exception('Failed to load data from server');
    }
  }
}
```

## Step 5: Display the JSON Data

Now, you can use the **OnlineService** class to display the JSON data. Here is how you can use it on your **main.dart** file:

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

void main() {
  runApp(const MyApp());
}
```

```

}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    const appTitle = 'JSON in Flutter';

    return MaterialApp(
      title: appTitle,
      home: Scaffold(
        appBar: AppBar(
          title: const Text(appTitle),
        ),
        body: FutureBuilder(
          future: OnlineService().loadPerson(),
          builder: (context, snapshot) {
            if (snapshot.hasData) {
              Person person = snapshot.data as Person;
              return Center(
                child: Padding(
                  padding: const EdgeInsets.all(16.0),
                  child: Card(
                    elevation: 4.0,
                    child: Padding(
                      padding: const EdgeInsets.all(16.0),
                      child: Column(
                        mainAxisAlignment: MainAxisAlignment.min,
                        children: [
                          CircleAvatar(
                            backgroundImage: NetworkImage(person.image),
                            radius: 50.0,
                          ),
                          const SizedBox(height: 10),
                          Text(
                            person.name,
                            style: const TextStyle(
                              fontSize: 24,
                              fontWeight: FontWeight.bold,
                            ),
                          ),
                        ],
                      ),
                    ),
                  ),
                ),
                child: const SizedBox(height: 5),
                Text(
                  person.address,
                  style: TextStyle(
                    fontSize: 16,
                    color: Colors.grey[600],

```

```

    ),
  ),
  const SizedBox(height: 5),
  Text(
    'Age: ${person.age}',
    style: TextStyle(
      fontSize: 16,
      color: Colors.grey[800],
    ),
  ),
  const SizedBox(height: 10),
  Text(
    person.description,
    textAlign: TextAlign.center,
    style: const TextStyle(
      fontSize: 16,
      fontStyle: FontStyle.italic,
    ),
  ),
),
],
),
),
),
),
),
),
);
} else {
  return const Center(child: CircularProgressIndicator());
}
},
),
),
);
}
}

```

## Challenge

Create quote app using online JSON. Use [this JSON file](#) to display the quotes.

[CREATE QUOTE APP IN FLUTTER](#)

## Introduction

In this section, you will learn to create a quote app using online JSON in flutter. Before we start, make sure you have basic knowledge of JSON. If you don't know start from [here](#).

## Quote JSON Link

You can use JSON file from [here](#). It contains 50+ quotes. Here is preview for first 3 quotes.

```
[
  {
    "text": "The only people who never fail are those who never try.",
    "from": "Ilka Chase"
  },
  {
    "text": "Failure is just another way to learn how to do something right.",
    "from": "Marian Wright Edelman"
  },
  {
    "text": "I failed my way to success.",
    "from": "Thomas Edison"
  },
]
```

## How to Create Quote App

Now you will learn to create a quote app using online JSON. Here are the steps to create a app:

### Step 1: Install http Package

To use online JSON in your app, the first step is to add the http package in your project. Open terminal at your project folder and execute the following command:

```
flutter pub add http
```

### Step 2: Import the http Package

Now, you need to import the http package in your flutter application. To import the http package, open your main.dart file and add the following import statement:

```
import 'package:http/http.dart' as http;
```

### Step 3: Create Model Class

Now, you need to create a model class for the JSON data. Create a new file called **quote.dart** inside **lib** folder and add the following code in it:

```
class Quote {
```

```

final String text;
final String from;

// Constructor
Quote({required this.text, required this.from});

// Convert JSON to Quote Object
factory Quote.fromJson(Map<String, dynamic> json) {
  return Quote(
    text: json['text'],
    from: json['from'],
  );
}
}

```

#### Step 4: Load and Decode the JSON File

Now create a new file called **onlineservice.dart** inside **lib** folder and add the following code in it:

```

import 'dart:convert';
import 'quote.dart';

class OnlineService {
  // Load and Decode JSON
  Future<List<Quote>> getQuotes() async {
    // Load JSON
    final response = await
http.get(Uri.parse('https://jsonguide.technologychannel.org/quotes.json'))
;

    // Decode JSON
    final json = jsonDecode(response.body).cast<Map<String, dynamic>>();

    // Convert JSON to Quote
    return json.map<Quote>((json) => Quote.fromJson(json)).toList();
  }
}

```

#### Step 5: Create Quote Widget

Now, you need to create a widget to display the quote. Create a new file called **quote\_widget.dart** inside **lib** folder and add the following code in it:

```

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

```

```

class QuoteWidget extends StatelessWidget {
  final Quote quote;

  const QuoteWidget({Key? key, required this.quote}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 4,
      margin: const EdgeInsets.all(8),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(10),
      ),
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              quote.text,
              style: const TextStyle(fontSize: 20, fontWeight:
FontWeight.bold),
            ),
            const SizedBox(height: 10),
            Text(
              '- ${quote.from}',
              style: const TextStyle(fontSize: 16, fontStyle:
FontStyle.italic),
              textAlign: TextAlign.right,
            ),
          ],
        ),
      ),
    );
  }
}

```

## Step 6: Display the JSON Data

Now, you can use the **OnlineService** class to display the JSON data. Here is how you can use it on your **main.dart** file:

```

import 'package:flutter/material.dart';
import 'onlineservice.dart';
import 'quote.dart';
import 'quote_widget.dart';

```

```

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: FutureBuilder<List<Quote>>(
          future: OnlineService().getQuotes(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const Center(child: CircularProgressIndicator());
            }
            if (snapshot.hasError) {
              return Center(child: Text('Error: ${snapshot.error}'));
            }
            if (snapshot.hasData) {
              final quotes = snapshot.data!;
              return ListView.builder(
                itemCount: quotes.length,
                itemBuilder: (context, index) {
                  return QuoteWidget(quote: quotes[index]);
                },
              );
            }
            return const Center(child: Text('No data available'));
          },
        ),
      ),
    );
  }
}

```

## Challenge

Now create a similar app using local JSON. You can use JSON file from [here](#).

USE LOCAL AUDIO IN FLUTTER

## Introduction

In this section, you will learn to use local audio in your flutter application with the help of real-world example. You will create a simple audio player app. Here are the steps to create a app:

## Step 1: Add Audio File To Your Project

You can download sample audio file from [here](#) or you can use your own audio file. Place the audio file inside **assets/audio** with the name **sound.mp3**.

## Step 2: Install audioplayers Package

To use local audio in your app, the first step is to add the audioplayers package in your project. Open terminal at your project folder and execute the following command:

```
flutter pub add audioplayers
```

## Step 3: Update Pubspect.yaml Assets

Now, you need to update the pubspec.yaml file. Go to your pubspec.yaml file and add the following code in it:

```
flutter:  
  assets:  
    - assets/audio/sound.mp3
```

## Step 4: Import the audioplayers Package

Now, you need to import the audioplayers package in your flutter application. To import the audioplayers package, open your main.dart file and add the following import statement:

```
import 'package:audioplayers/audioplayers.dart';
```

## Step 5: Code for Audio Player

Now, you need to create a audio player widget. Open main.dart file and add the following code in it:

```
import 'package:flutter/material.dart';  
import 'package:audioplayers/audioplayers.dart';  
  
void main() {  
  runApp(MyApp());  
}  
  
// Stateful widget to handle audio playback.
```



```

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState(); // State creation.
}

class _MyAppState extends State<MyApp> {
  final AudioPlayer audioPlayer = AudioPlayer(); // Audio player instance.

  @override
  void dispose() {
    audioPlayer.dispose(); // Release audio player resources.
    super.dispose();
  }

  // Plays an audio file.
  void playAudio() async {
    await audioPlayer.play(AssetSource('audio/sound.mp3'));
  }

  // Stops the audio playback.
  void stopAudio() async {
    await audioPlayer.stop();
  }

  @override
  Widget build(BuildContext context) {
    // Builds the app's UI.
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: const Text('Audio Player Example')),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              ElevatedButton(
                onPressed: playAudio, child: const Text('Play Audio')),
              ElevatedButton(
                onPressed: stopAudio, child: const Text('Stop Audio')),
            ],
          ),
        ),
      ),
    );
  }
}

```

## Project Source Code

If you face any problem while creating this app, you can download the source code of this project from [here](#).

## Challenge

Create a simple audio player app which can play and stop the audio file from the following URL:

```
https://raw.githubusercontent.com/Flutter-Tutorial-Website/SimpleFlutterAudioPlayer/master/assets/audio/sound.mp3
```

## Solution

You can simply create **playAudioOnline()** method and call it on button press. Here is the code for **playAudioOnline()** method:

```
void playAudioOnline() async {  
  await  
  audioPlayer.play(UrlSource('https://raw.githubusercontent.com/Flutter-Tutorial-Website/SimpleFlutterAudioPlayer/master/assets/audio/sound.mp3'));  
}
```

## QUESTIONS FOR PRACTICE 3

### Flutter Assets and Data Handling Practice Questions

1. Create a Flutter app that loads and displays an image from the assets folder. Ensure the image scales correctly in both portrait and landscape modes.
2. Create a Flutter application that fetches and displays an image from a URL. Implement error handling for scenarios where the image cannot be loaded.
3. Create a Flutter app that caches images from the internet and loads them from the cache on subsequent launches for faster performance.
4. Create a Flutter application that uses a custom font from the local assets and apply it to the entire app text.
5. Create a Flutter app that dynamically fetches and applies a Google Font to a selected text widget.
6. Write a Flutter application that reads a local JSON file from the assets folder and displays its contents in a user-friendly format.
7. Create a Flutter app that fetches JSON data from an online source and displays it in a ListView.
8. Create a Flutter app that displays a list of quotes. The quotes should be stored locally and should include functionality to add, delete, and view quotes.
9. In a Flutter app, create a feature to switch between two sets of assets (like images and fonts) based on user selection or a toggle switch.
10. Build a Flutter application that combines local and network images in a grid layout, demonstrating effective asset management in Flutter.

## 4) \_Useful Widgets In Flutter

This section will help you to learn most useful widgets in flutter. Here you will learn the following topics:

- [Drawer in Flutter,](#)
- [Snackbar in Flutter,](#)
- [Bottom Navigation Bar in Flutter,](#)
- [Alert Dialog in Flutter,](#)
- [Table in Flutter,](#)
- [Forms in Flutter,](#) and
- [Buttons in Flutter.](#)

### Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

#### DRAWER IN FLUTTER

### Introduction

**Drawer** is a sidebar that appears from the side of the screen, usually filled with menu options or navigation links, allowing you to navigate between different parts of the application.

### Create Drawer in Flutter

To create a drawer in flutter, you need to use **Scaffold** widget. Scaffold widget provides a **drawer** property to add a drawer in your app. Here is an example of drawer in flutter:

```
Scaffold(  
  drawer: Drawer(  
  ),  
  appBar: AppBar(  
    title: Text('Click Drawer Icon'),  
  ),  
)
```

### Drawer From Right

By default, the drawer appears from the left side of the screen. If you want to make it appear from the right side, you can use **endDrawer** property of Scaffold widget. Here is an example of drawer from right:

```
Scaffold(  
  endDrawer: Drawer(  
    child: Text('Click Drawer Icon'),  
  ),  
)
```

```

),
appBar: AppBar(
  title: Text('Click End Drawer Icon =>>>'),
),
)

```

## Example 1: Simple Drawer

In this example below, you will learn to create drawer with options **Home**, **About**, and **Settings** with icons.

```

Drawer(
  child: ListView(
    children: const [
      DrawerHeader(
        child: Text('Drawer Header'),
      ),
      ListTile(
        leading: Icon(Icons.home),
        title: Text('Home'),
      ),
      ListTile(
        leading: Icon(Icons.info),
        title: Text('About'),
      ),
      ListTile(
        leading: Icon(Icons.settings),
        title: Text('Settings'),
      ),
    ],
  ),
)

```

## Example 2: Ecommerce App Drawer

In this example below, you will learn how to create a beautiful drawer for an ecommerce app with options **Home**, **Categories**, **Orders**, **Wishlist** and **Settings**.

```

Drawer(
  child: ListView(
    padding: EdgeInsets.zero,
    children: <Widget>[
      DrawerHeader(
        child: Text('Menu'),
        decoration: BoxDecoration(
          color: Colors.blue,
        ),
      ),
    ],
  ),
)

```

```

    ),
    ListTile(title: Text('Home'), onTap: () {}),
    ListTile(title: Text('Categories'), onTap: () {}),
    ListTile(title: Text('Orders'), onTap: () {}),
    ListTile(title: Text('Wishlist'), onTap: () {}),
    ListTile(title: Text('Settings'), onTap: () {}),
  ],
),
)

```

### Example 3: Account Management in Social Media Apps

In this example below, you will learn how to create a beautiful drawer for a social media app with options **Profile**, **Friends**, **Messages** and **Settings**.

```

Drawer(
  child: ListView(
    padding: EdgeInsets.zero,
    children: <Widget>[
      UserAccountsDrawerHeader(
        accountName: Text('User Name'),
        accountEmail: Text('user@example.com'),
        currentAccountPicture: CircleAvatar(
          backgroundColor: Colors.orange,
          child: Text('U'),
        ),
      ),
      ListTile(title: Text('Profile'), onTap: () {}),
      ListTile(title: Text('Friends'), onTap: () {}),
      ListTile(title: Text('Messages'), onTap: () {}),
      ListTile(title: Text('Settings'), onTap: () {}),
    ],
  ),
)

```

### Example 4: Beautiful Drawer Header

In this example below, you will build a beautiful drawer header with background image and user profile image.

```

Drawer(
  child: ListView(
    children: <Widget>[
      DrawerHeader(
        decoration: BoxDecoration(
          image: DecorationImage(

```

```

        image: NetworkImage('https://flutter-tutorial.net/images/sample_background_image.jpg'),
        fit: BoxFit.cover,
      ),
    ),
    child: Column(
      mainAxisAlignment: MainAxisAlignment.end,
      children: <Widget>[
        CircleAvatar(
          radius: 40,
          backgroundImage:
NetworkImage('https://avatars.githubusercontent.com/u/33576285?v=4'),
        ),
        SizedBox(height: 10),
        Text('John Doe', style: TextStyle(color: Colors.white)),
        SizedBox(height: 10),
      ],
    ),
  ),
  ListTile(
    leading: Icon(Icons.home),
    title: Text('Home'),
    onTap: () {
      // Handle the tap if needed
    },
  ),
  ListTile(
    leading: Icon(Icons.info),
    title: Text('About'),
    onTap: () {
      // Handle the tap if needed
    },
  ),
],
),
)

```

## Challenge

Create a beautiful drawer for accounting app with options **Dashboard**, **Invoices**, **Payments**, **Expenses** and **Settings**.

[SNACKBAR IN FLUTTER](#)

## Introduction

**Snackbar** is a lightweight message bar displayed at the bottom of the screen. It is used to show short notifications like **saved** or **deleted** and lets users perform actions such as **undo** or **retry**.

### Example 1: Simple Snackbar

In this example below, you will create a simple Snackbar that displays a message **This is a Snackbar!** when a button is pressed.

```
ScaffoldMessenger.of(context).showSnackBar(  
  SnackBar(  
    content: Text('This is a Snackbar!'),  
  ),  
);
```

### Example 2: Snackbar with Action

In this example below, you will create a Snackbar with an action that allows the user to undo an operation.

```
ScaffoldMessenger.of(context).showSnackBar(  
  SnackBar(  
    content: Text('Message deleted'),  
    action: SnackBarAction(  
      label: 'UNDO',  
      onPressed: () {  
        // Perform some action  
      },  
    ),  
  ),  
);
```

### Example 3: Customized Snackbar

In this example below, you will create a Snackbar with a custom background color.

```
ScaffoldMessenger.of(context).showSnackBar(  
  SnackBar(  
    content: Text('This is a custom Snackbar!'),  
    backgroundColor: Colors.blue,  
  ),  
);
```

### Example 4: Snackbar with Duration

In this example below, you will create a Snackbar that disappears after 3 seconds.

```
ScaffoldMessenger.of(context).showSnackBar(  
  SnackBar(  
    content: Text('This message will disappear after 3 seconds'),  
    duration: Duration(seconds: 3),  
  ),  
);
```

## Challenge

Create a SnackBar that displays a message **Account created successfully** when a user press **Register** button. The SnackBar should have a green background and disappear after 5 seconds.

```
MaterialButton(  
  onPressed: () {  
    // your code here  
  },  
  child: Text('Register'),  
)
```

## BOTTOM NAV BAR IN FLUTTER

### Introduction

**Bottom Navigation Bar** is a material widget at the bottom of the screen that displays multiple tabs for easy navigation between different views or functionalities in an app.

### Basic Bottom Navigation Bar

To use a Bottom Navigation Bar in Flutter, you can wrap it inside a **Scaffold** widget. Here's a basic example:

```
Scaffold(  
  bottomNavigationBar: BottomNavigationBar(  
    items: const <BottomNavigationBarItem>[  
      BottomNavigationBarItem(  
        icon: Icon(Icons.home),  
        label: 'Home',  
      ),  
      BottomNavigationBarItem(  
        icon: Icon(Icons.business),  
        label: 'Business',  
      ),  
      BottomNavigationBarItem(  
        icon: Icon(Icons.school),
```



```

        label: 'School',
      ),
    ],
  ),
);

```

## Handling Navigation

Typically, you want to change the body of the **Scaffold** based on the tab selected. This requires managing the state of the selected index. You can see complete demo in **Example 1** soon.

```

int _selectedIndex = 0;

void _onItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });
}

...

BottomNavigationBar(
  items: <BottomNavigationBarItem>[...],
  currentIndex: _selectedIndex,
  onTap: _onItemTapped,
),

```

## Example 1: Ecommerce App Bottom Navigation Bar

In this example below, you will create a Bottom Navigation Bar with 4 tabs: **Home**, **Categories**, **Cart**, and **Profile**. Each tab will have a corresponding screen.

```

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key}) : super(key: key);

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _selectedIndex = 0;

  final List<Widget> _widgetOptions = const [
    Text('Home'),
    Text('Categories'),
    Text('Cart'),

```

```

        Text('Profile'),
    ];

    void _onItemTapped(int index) {
        setState(() {
            _selectedIndex = index;
        });
    }

    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
                title:
                    const Text('Ecommerce App', style: TextStyle(color:
Colors.white)),
                backgroundColor: Colors.indigo, // AppBar color
            ),
            body: Center(
                child: _widgetOptions.elementAt(_selectedIndex),
            ),
            bottomNavigationBar: BottomNavigationBar(
                type: BottomNavigationBarType.fixed,
                backgroundColor: Colors.indigo, // BottomNavigationBar background
color
                selectedItemColor: Colors.amber, // Selected item color
                unselectedItemColor: Colors.white, // Unselected item color
                items: const [
                    BottomNavigationBarItem(
                        icon: Icon(Icons.home),
                        label: 'Home',
                    ),
                    BottomNavigationBarItem(
                        icon: Icon(Icons.category),
                        label: 'Categories',
                    ),
                    BottomNavigationBarItem(
                        icon: Icon(Icons.shopping_cart),
                        label: 'Cart',
                    ),
                    BottomNavigationBarItem(
                        icon: Icon(Icons.person),
                        label: 'Profile',
                    ),
                ],
                currentIndex: _selectedIndex,
                onTap: _onItemTapped,
            ),

```

```
}  
  }  
);
```

## Challenge

Create a Bottom Navigation Bar with at least 4 items, each representing different functionalities of a gym app (e.g., **Home**, **Workouts**, **Profile**, **Settings**). Implement corresponding Text widgets for each tab.

## ALERT DIALOG IN FLUTTER

### Introduction

**Alert Dialog** is simple pop-up message that grab the user's attention for important information or decisions. They pop up over the app's content, showing titles, messages, and action buttons for quick user responses.

### Why Use Alert Dialogs?

- **User Attention:** Captures the user's focus effectively.
- **Immediate Action:** Facilitates quick decision-making or acknowledgment from the user.
- **Feedback Collection:** Can be used to gather input or feedback from the user.

### Creating an Alert Dialog

Creating an alert dialog in Flutter is straightforward. You typically wrap the dialog presentation in a function that can be called on a specific event, such as pressing a button.

```
void showMyAlertDialog(BuildContext context) {  
  showDialog(  
    context: context,  
    builder: (BuildContext context) {  
      return AlertDialog(  
        title: Text('Alert Dialog Title'),  
        content: Text('This is an alert dialog. Are you sure you want to  
proceed?'),  
        actions: <Widget>[  
          TextButton(  
            child: Text('Cancel'),  
            onPressed: () {  
              Navigator.of(context).pop();  
            },  
          ),  
          TextButton(  

```

```

        child: Text('OK'),
        onPressed: () {
          // Handle the action and close the dialog
          Navigator.of(context).pop();
        },
      ),
    ],
  );
},
);
}

```

To use the **showMyAlertDialog** function, you can call it from a button's **onPressed** event or any other event that triggers the dialog.

```

ElevatedButton(
  onPressed: () => showMyAlertDialog(context),
  child: Text('Show Alert Dialog'),
)

```

### Example 1: Show Alert Dialog To Exit From App

In this example, you will create an alert dialog that asks the user for confirmation before exiting the app.

```

showMyAlertDialog(BuildContext context) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Exit App'),
        content: Text('Are you sure you want to exit?'),
        actions: <Widget>[
          TextButton(
            child: Text('Cancel'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
          TextButton(
            child: Text('OK'),
            onPressed: () {
              // Handle the action and close the dialog
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}

```

```

    );
  },
);
}

```

## Example 2: Readymade Alert Dialog

This is ready-made alert dialog function that you can use in your app. This function takes the **context**, **title**, and **content** as parameters and displays the alert dialog.

```

showMyAlertDialog(BuildContext context, String title, String content) {
  showDialog(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text(title),
        content: Text(content),
        actions: <Widget>[
          TextButton(
            child: Text('Cancel'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
          TextButton(
            child: Text('OK'),
            onPressed: () {
              // Handle the action and close the dialog
              Navigator.of(context).pop();
            },
          ),
        ],
      );
    },
  );
}

```

## Challenge

Try creating an alert dialog displaying a message **Account created successfully** when a user presses the **Register** button.

```

MaterialButton(
  onPressed: () {
    // your code here
  },
  child: Text('Register'),
)

```

)

## TABLE IN FLUTTER

### Table in Flutter

Table widget is a great way to display data in a table layout within your mobile app. It's useful when you need to present data in a structured format, such as financial reports, timetables, or settings.

### Example 1: Student Record With Table

In this example below, you will learn to create a table to display student information such as name, subject, and grade.

```
Table(
  border: TableBorder.all(), // Adds a border to all cells
  columnWidths: <int, TableColumnWidth>{
    0: FixedColumnWidth(100.0), // fixed to 100.0 width
    1: FlexColumnWidth(), // automatically adapts to fill the table width
    2: FixedColumnWidth(100.0), // fixed to 100.0 width
  },
  children: [
    TableRow(children: [
      Text('Name'),
      Text('Subject'),
      Text('Grade'),
    ]),
    TableRow(children: [
      Text('Alice'),
      Text('Math'),
      Text('A'),
    ]),
    TableRow(children: [
      Text('Bob'),
      Text('Science'),
      Text('B+'),
    ]),
    // Add more students and grades here
  ],
)
```

### DataTable In Flutter

DataTable widget is a powerful widget that allows you to display tabular data with advanced features such as sorting, selection, and pagination. It is best for displaying complex tables with interactivity.

## Example 2: Student Record With DataTable

In this example below, you will learn to create a DataTable to display student information such as name, subject, and grade.

```
DataTable(  
  columns: const [  
    DataColumn(label: Text('Name')),  
    DataColumn(label: Text('Subject')),  
    DataColumn(label: Text('Grade')),  
  ],  
  rows: const [  
    DataRow(cells: [  
      DataCell(Text('Alice')),  
      DataCell(Text('Math')),  
      DataCell(Text('A')),  
    ]),  
    DataRow(cells: [  
      DataCell(Text('Bob')),  
      DataCell(Text('Science')),  
      DataCell(Text('B+')),  
    ]),  
    // Add more students and grades here  
  ],  
)
```

## Example 3: Contact List App

Now you will learn to create a contact list app using DataTable widget. You will define a model class called **contact.dart** to store contact information. Then you will build DataTable widget to display contact information.

### Step 1: Define Model Class

In this step, you will define a model class called **contact.dart**. This class will be used to store contact information. You can define this class in **lib/model/contact.dart** file. Add the following code in **contact.dart** file.

```
class Contact {  
  final String name;  
  final String email;  
  final String phone;
```

```
Contact({this.name, this.email, this.phone});  
}
```

## Step 2: Create Contact List

In this step, you will create a list of contacts. You can define this list in **lib/data/contact\_data.dart** file. Add the following code in **contact\_data.dart** file.

```
import 'package:your_app_name/model/contact.dart';  
  
List<Contact> contacts = [  
  Contact(name: 'Alice', email: 'alice@domain.com', phone: '1234567890'),  
  Contact(name: 'Bob', email: 'bob@domain.com', phone: '1234567890'),  
  Contact(name: 'Charlie', email: 'charl@domain.com', phone:  
    '1234567890'),  
  // Add more contacts here  
];
```

## Step 3: Display Contact List

In this step, you will display the contact list using DataTable widget. You can define this widget in **lib/screens/contact\_list.dart** file. Add the following code in **contact\_list.dart** file.

```
import 'package:flutter/material.dart';  
import 'package:your_app_name/data/contact_data.dart';  
import 'package:your_app_name/model/contact.dart';  
  
class ContactList extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Contact List'),  
      ),  
      body: DataTable(  
        columns: const [  
          DataColumn(label: Text('Name')),  
          DataColumn(label: Text('Email')),  
          DataColumn(label: Text('Phone')),  
        ],  
        rows: contacts.map((contact) {  
          return DataRow(cells: [  
            DataCell(Text(contact.name)),  
            DataCell(Text(contact.email)),  
            DataCell(Text(contact.phone)),  
          ]);  
        }).toList(),  
      ),  
    );  
  }  
}
```



```

        ]);
    }).toList(),
  ),
);
}
}

```

## Challenge

Create a DataTable that displays a list of products with columns **Name**, **Price**, and **Stock**. The DataTable should have 5 rows of products with different names, prices, and stock values.

## QUESTIONS FOR PRACTICE 4

### Flutter Practice Questions

- Create a Flutter application with a form that includes text fields for name, email, and phone number. Use custom styling for the form fields.
- Create a screen with a variety of buttons (e.g., raised, flat, icon, and floating action buttons) demonstrating different styles and functionalities.
- Implement a table in Flutter displaying a list of products with columns for product name, price, and quantity. Include functionality to sort the table by each column.
- Design a form with validation that includes fields for user registration: username, password, confirm password, and email. Display appropriate error messages for invalid inputs. Construct a dynamic form in Flutter that adds a new text field every time the user presses an 'Add' button. Include a 'Submit' button to display all entered data in a dialog box.
- Create a responsive Flutter layout with a table on a large screen (e.g., tablet or desktop) and a list view on smaller screens (e.g., mobile), containing the same data.
- Create a multi-step form in Flutter where each step is a different page in a PageView. Include 'Next' and 'Previous' buttons to navigate between the steps.
- Implement a custom button in Flutter that changes its color and elevation when pressed, with a smooth animation transition.
- Create a Flutter application that uses a DataTable widget to display user data fetched from a mock JSON API. Include features for pagination and row selection.
- Create a sign-in form in Flutter with animated transitions between the 'Sign In' and 'Sign Up' forms, showcasing a smooth user experience.

## 5) \_Flutter Buttons

This section will help you to learn different types of buttons in flutter. Here you will learn the following topics:

- [Buttons in Flutter](#),

- [MaterialButton](#) in Flutter.
- [ElevatedButton](#) in Flutter,
- [TextButton](#) in Flutter,
- [OutlinedButton](#) in Flutter,
- [IconButton](#) in Flutter,
- [FloatingActionButton](#) in Flutter,
- [DropDownButton](#) in Flutter, and
- [PopupMenuButton](#) in Flutter.

## Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

### BUTTONS IN FLUTTER

## Flutter Buttons

**Buttons** are an essential part of any application. They allow you to interact with the app and perform various actions. In this section, you will learn about different types of buttons available in Flutter, along with examples for each.

## Button Types in Flutter

You can get different types of buttons in Flutter, such as:

Button	Description
<b>Material Button</b>	Mostly used button with material design.
<b>Elevated Button</b>	Shadowed button for important actions.
<b>Text Button</b>	Flat button for less important actions.
<b>Outlined Button</b>	Border button for secondary actions.
<b>Icon Button</b>	Button with an icon.
<b>Floating Action Button</b>	Circular main action button.
<b>DropDown Button</b>	Button to select from a list of options.
<b>Popup Menu Button</b>	Button that shows more options.

**Note:** You will learn about each button type with real world examples in upcoming sections.

## Button Useful Properties

Here are some common properties that you can use with buttons in Flutter:

Property	Description
<b>onPressed</b>	Things to do when the button is pressed.
<b>child</b>	Widget to display inside the button.

### Example 1: Simple MaterialButton In Flutter

If you want to create a simple button with text, use the MaterialButton widget. It is a versatile button that can display text, icons, and more.

```
MaterialButton(  
  onPressed: () {  
    // action to perform when button is pressed  
  },  
  color: Colors.blue,  
  child: const Text('Press Me', style: TextStyle(color: Colors.white)),  
)
```

### Popular Buttons in Flutter

In upcoming sections, you will learn about different types of buttons available in Flutter, along with examples for each. Click on the links below to get started with flutter buttons:

- [Material Button in Flutter](#)
- [Elevated Button in Flutter](#)
- [Text Button in Flutter](#)
- [Outlined Button in Flutter](#)
- [Icon Button in Flutter](#)
- [Floating Action Button in Flutter](#)
- [Dropdown Button in Flutter](#)
- [Popup Menu Button in Flutter](#)

## MATERIAL BUTTON IN FLUTTER

### Introduction

MaterialButton is one of the most commonly used buttons in Flutter. It is a versatile button that can display text, icons, and more. This guide will help you understand the MaterialButton with the help of real-world examples.

### Example 1: Simple MaterialButton

In this example below, you will learn to create a simple button with text using the `MaterialButton` widget.

```
MaterialButton(  
  onPressed: () {},  
  color: Colors.blue,  
  child: const Text('Press Me'),  
)
```

## Example 2: MaterialButton With Icon

In this example below, you will learn to create a button with an icon using the `MaterialButton` widget.

```
MaterialButton(  
  onPressed: () {  
    // Action to perform on button press  
  },  
  color: Colors.green,  
  textColor: Colors.white,  
  child: const Row(  
    mainAxisAlignment: MainAxisAlignment.min, // Align text and icon closely  
    together  
    children: <Widget>[  
      Icon(Icons.add),  
      SizedBox(width: 8), // Space between icon and text  
      Text('Add Item'),  
    ],  
  ),  
)
```

## Example 3: Rounded Corners MaterialButton

In this example below, you will learn to create a button with rounded corners using the `MaterialButton` widget.

```
MaterialButton(  
  onPressed: () {},  
  color: Colors.blue,  
  shape: RoundedRectangleBorder(  
    borderRadius: BorderRadius.circular(18.0),  
  ),  
  child: const Text('Press Me'),  
)
```

## Example 4: MaterialButton with Shadow and Elevation

In this example below, you will learn to create a button with shadow and elevation using the `MaterialButton` widget.

```
MaterialButton(  
  onPressed: () {},  
  color: Colors.blue,  
  elevation: 5,  
  child: const Text('Press Me'),  
)
```

### Example 5: Disabled MaterialButton

In this example below, you will learn to create a disabled button using the `MaterialButton` widget.

```
MaterialButton(  
  onPressed: null,  
  color: Color.fromARGB(255, 205, 208, 211),  
  child: Text('Disabled Button'),  
)
```

### Example 6: Gradient Background In MaterialButton

In this example below, you will learn to create a button with a gradient background using the `MaterialButton` widget.

```
MaterialButton(  
  onPressed: () {},  
  child: Ink(  
    decoration: BoxDecoration(  
      gradient: const LinearGradient(  
        colors: [Colors.blue, Colors.red]  
      ),  
      borderRadius: BorderRadius.circular(10.0),  
    ),  
    child: Container(  
      constraints: const BoxConstraints(minWidth: 88.0, minHeight: 36.0),  
      alignment: Alignment.center,  
      child: const Text(  
        'Gradient Button',  
        style: TextStyle(color: Colors.white, fontSize: 20),  
      ),  
    ),  
  ),  
)
```

**Note:** Ink widget is used for decorating a part of your UI with a background image, color, or custom drawings

## Challenge

Create a simple MaterialButton with text “**Click Me**” with a **red** color and **white** text color. When the button is pressed, display a **snackbar** with the message “**Button Pressed**”.

## ELEVATED BUTTON IN FLUTTER

### Introduction

**ElevatedButton** is one of the most commonly used button widgets in Flutter. It is a material design button that is elevated from the surface, making it prominent and visually appealing. Here you will learn how to use elevated buttons with real-world examples.

### Example 1: Simple ElevatedButton

In this example below, you will learn to create a simple button with text using the ElevatedButton widget.

```
ElevatedButton(  
  onPressed: () {},  
  child: Text('Press Me'),  
)
```

### Example 2: ElevatedButton With Icon

In this example below, you will learn to create a button with an icon using the ElevatedButton widget.

```
ElevatedButton.icon(  
  onPressed: () {},  
  icon: Icon(Icons.add),  
  label: Text('Add Item'),  
)
```

### Example 3: Custom Color and TextStyle

In this example below, you will learn to create a button with custom color and text style using the ElevatedButton widget.

```
ElevatedButton(  
  onPressed: () {},  
  style: ElevatedButton.styleFrom(  
    backgroundColor: Colors.red,  
    textStyle: TextStyle(  
      color: Colors.white,  
      fontSize: 16,  
    ),  
  ),  
  child: Text('Click Me'),  
)
```

```

onPressed: () {},
style: ButtonStyle(
  backgroundColor: MaterialStateProperty.all(Colors.green),
  textStyle: MaterialStateProperty.all(
    TextStyle(fontSize: 20),
  ),
),
child: Text('Custom Style'),
)

```

#### Example 4: Rounded Corners ElevatedButton

In this example below, you will learn to create a button with rounded corners using the ElevatedButton widget.

```

ElevatedButton(
  onPressed: () {},
  style: ButtonStyle(
    shape: MaterialStateProperty.all<RoundedRectangleBorder>(
      RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(18.0),
      ),
    ),
  ),
  child: Text('Rounded Corners'),
)

```

#### Example 5: Adjusting Shadow and Elevation

In this example below, you will learn to create a button with shadow and elevation using the ElevatedButton widget.

```

ElevatedButton(
  onPressed: () {},
  style: ButtonStyle(
    elevation: MaterialStateProperty.all(10),
  ),
  child: Text('Shadow and Elevation'),
)

```

#### Example 6: Disabled ElevatedButton

In this example below, you will learn to create a disabled button using the ElevatedButton widget.

```

ElevatedButton(

```

```
onPressed: null,  
child: Text('Disabled Button'),  
)
```

## Challenge

Construct an ElevatedButton that changes its text from “**Click Me**” to “**Clicked**” upon being pressed. Additionally, the button should display a **snackbar** with the message “**Button Pressed**” when clicked.

## FLOATING ACTION BUTTON IN FLUTTER

### Introduction

**Floating Action Button (FAB)** is a circular icon button that floats above the user interface. In this section, you will learn how to use the FAB widget in Flutter to create a variety of floating action buttons with different styles and functionalities. It is commonly used to trigger primary actions in an application, such as adding a new item, composing a new message, or initiating a new task.

### Example 1: Simple FAB

In this example below, you will learn to create a basic FAB with a default icon.

```
// put FAB inside Scaffold.  
floatingActionButton: FloatingActionButton(  
  onPressed: () {},  
  child: Icon(Icons.add),  
)
```

### Example 2: Custom Color FAB

In this example, you will learn to create a FAB with a custom background color.

```
FloatingActionButton(  
  onPressed: () {},  
  backgroundColor: Colors.green,  
  child: Icon(Icons.phone),  
)
```

### Example 3: Mini FAB

In this example below, you will learn to implement a smaller version of the FAB suitable for limited spaces.



```
FloatingActionButton(  
  mini: true,  
  onPressed: () {},  
  child: Icon(Icons.star),  
)
```

### Example 4: Extended FAB

In this example below, you will learn to create an extended FAB with a label and icon.

```
FloatingActionButton.extended(  
  onPressed: () {},  
  icon: Icon(Icons.add),  
  label: Text('Add Item'),  
)
```

### Challenge

Create floating action button which change its background color from green to red when pressed.

## ICON BUTTON IN FLUTTER

### Introduction

IconButton is a type of button in Flutter that displays an icon instead of text. In this section, you will learn how to use the IconButton widget effectively, with real-world examples.

### Example 1: Simple IconButton

In this example below, you will learn to create a simple IconButton with a default icon.

```
IconButton(  
  onPressed: () {},  
  icon: Icon(Icons.home),  
)
```

### Example 2: IconButton with Custom Color

In this example below, you will learn to create an IconButton with a custom color for the icon.

```
IconButton(  
  onPressed: () {},
```

```
icon: Icon(Icons.send),  
color: Colors.purple,  
)
```

### Example 3: IconButton with Size Customization

In this example below, you will learn to create an IconButton with a custom size for the icon.

```
IconButton(  
  onPressed: () {},  
  icon: Icon(Icons.alarm),  
  iconSize: 30,  
)
```

### Example 4: IconButton with Tooltip

In this example below, you will learn to create an IconButton with tooltip. Tooltip provide additional information on hover or long press.

```
IconButton(  
  onPressed: () {},  
  icon: Icon(Icons.info),  
  tooltip: 'More Info',  
)
```

### Example 5: Disabled IconButton

In this example below, you will learn to create a disabled button using the IconButton widget.

```
IconButton(  
  icon: Icon(Icons.block),  
  onPressed: null,  
)
```

### Challenge

Create an IconButton that changes the color of the icon when pressed.

[OUTLINED BUTTON IN FLUTTER](#)

### Introduction

OutlinedButton is a type of button in Flutter that displays a border around the button's child. This guide will help you understand the OutlinedButton widget with the help of real-world examples.

### Example 1: Simple OutlinedButton

In this example below, you will learn to create a simple button with text.

```
OutlinedButton(  
  onPressed: () {},  
  child: Text('Press Me'),  
)
```

### Example 2: OutlinedButton With Icon

In this example below, you will learn to create a button with an icon.

```
OutlinedButton.icon(  
  onPressed: () {},  
  icon: Icon(Icons.add),  
  label: Text('Add Item'),  
)
```

### Example 3: Custom Border and TextStyle

In this example below, you will learn to create a button with custom border and text styles.

```
OutlinedButton(  
  onPressed: () {},  
  style: OutlinedButton.styleFrom(  
    side: BorderSide(color: Colors.green, width: 2),  
    textStyle: TextStyle(color: Colors.green, fontSize: 20),  
  ),  
  child: Text('Custom Style'),  
)
```

### Example 4: Rounded Corners OutlinedButton

In this example below, you will learn to create a button with rounded corners.

```
OutlinedButton(  
  onPressed: () {},  
  style: OutlinedButton.styleFrom(  
    shape: RoundedRectangleBorder(  
      borderRadius: BorderRadius.circular(18.0),
```

```
    ),  
  ),  
  child: Text('Rounded Corners'),  
)
```

### Example 5: Disabled OutlinedButton

In this example below, you will learn to create a disabled button using the OutlinedButton widget.

```
OutlinedButton(  
  onPressed: null,  
  child: Text('Disabled Button'),  
)
```

### Challenge

Create an OutlinedButton that changes the border color from blue to green when pressed.

[TEXT BUTTON IN FLUTTER](#)

## Introduction

TextButton is a simple button that contains a text label and perform an action when pressed. This guide will help you understand the TextButton with the help of real-world examples.

### Example 1: Default TextButton

In this example below, you will learn to create a simple TextButton with text label.

```
TextButton(  
  onPressed: () {},  
  child: Text('Press Me'),  
)
```

### Example 2: TextButton with Icon

In this example below, you will learn to create a TextButton with an icon and text label.

```
TextButton.icon(  
  onPressed: () {},  
  icon: Icon(Icons.info),  
  label: Text('More Info'),  
)
```

```
)
```

### Example 3: Custom Color and TextStyle

In this example below, you will learn to create a TextButton with custom text color and style.

```
TextButton(  
  onPressed: () {},  
  style: TextButton.styleFrom(  
    foregroundColor: Colors.black,  
    backgroundColor: Colors.yellow,  
  ),  
  child: Text('Custom Style'),  
)
```

### Example 4: TextButton with Underline

In this example below, you will learn to create a TextButton with underlined text.

```
TextButton(  
  onPressed: () {},  
  style: TextButton.styleFrom(  
    textStyle: TextStyle(decoration: TextDecoration.underline),  
  ),  
  child: Text('Underlined Text'),  
)
```

### Example 5: Disabled TextButton

In this example below, you will learn to create a disabled TextButton.

```
TextButton(  
  onPressed: null,  
  child: Text('Disabled Button'),  
)
```

### Challenge

Design a TextButton that toggles its text color between two colors upon each press, illustrating a simple state change.

[DROPDOWN BUTTON IN FLUTTER](#)

### Introduction

**DropDownButton** is a type of button in Flutter that displays a list of items when pressed. It allows users to select a single item from the list. This guide will help you understand the DropDownButton widget with the help of real-world examples.

## Use Cases

- **Country Selection:** Use a dropdown button to allow users to select a country from a list of countries.
- **Changing App Settings:** Use a dropdown button to allow users to change app settings such as language, theme, or font size.

## Example 1: Basic Dropdown Button

In this example below, you will learn to create a basic dropdown button with a list of items. For full code, press **Run Online** button.

```
DropDownButton<String>(  
  value: dropdownValue,  
  onChanged: (String? newValue) {  
    setState(() {  
      dropdownValue = newValue!;  
    });  
  },  
  items: <String>['One', 'Two', 'Three', 'Four']  
    .map<DropDownMenuItem<String>>((String value) {  
      return DropDownMenuItem<String>(  
        value: value,  
        child: Text(value),  
      );  
    }).toList(),  
)
```

## Example 2: Country Selection App

In this example below, you will learn to create a country selection app using a dropdown button. For full code, press **Run Online** button.

```
DropDownButton<String>(  
  value: selectedCountry,  
  onChanged: (String? newValue) {  
    setState(() {  
      selectedCountry = newValue!;  
    });  
  },  
  items: <String>['India', 'USA', 'UK', 'Canada']  
    .map<DropDownMenuItem<String>>((String value) {  
      return DropDownMenuItem<String>(  
        value: value,  
        child: Text(value),  
      );  
    }).toList(),  
)
```

```

        value: value,
        child: Text(value),
      );
    }).toList(),
  )

```

## Challenge

Create a dropdown button to select days of the week (Monday, Tuesday, Wednesday, etc.) and display the selected day in the app.

## POPUP MENU BUTTON IN FLUTTER

### Introduction

The PopupMenuButton widget in Flutter is used to display a menu when pressed and allows you to select from a list of options. It's commonly used for overflow menus, showing choices related to an item.

### Example 1: Basic Popup Menu Button

In this example below, you will learn to create a basic popup menu button with a list of items. When you press the button, a menu will appear with the available options.

```

PopupMenuButton<int>(  
  onSelect: (int result) {  
    setState(() {  
      _selectedMenu = result;  
    });  
  },  
  itemBuilder: (BuildContext context) => <PopupMenuEntry<int>>[  
    const PopupMenuItem<int>(  
      value: 1,  
      child: Text('About Us'),  
    ),  
    const PopupMenuItem<int>(  
      value: 2,  
      child: Text('Contact Us'),  
    ),  
    const PopupMenuItem<int>(  
      value: 3,  
      child: Text('Privacy Policy'),  
    ),  
  ],  
)

```

## Example 2: Popup Menu Button with Icons

In this example below, you will learn to create a popup menu button with icons for the menu items. When you press the button, a menu will appear with the available options, each with an icon.

```
PopupMenuButton<int>(  
  onSelect: (int result) {  
    setState(() {  
      _selectedMenu = result;  
    });  
  },  
  itemBuilder: (BuildContext context) => <PopupMenuEntry<int>>[  
    const PopupMenuItem<int>(  
      value: 1,  
      child: ListTile(  
        leading: Icon(Icons.info),  
        title: Text('About Us'),  
      ),  
    ),  
    const PopupMenuItem<int>(  
      value: 2,  
      child: ListTile(  
        leading: Icon(Icons.phone),  
        title: Text('Contact Us'),  
      ),  
    ),  
    const PopupMenuItem<int>(  
      value: 3,  
      child: ListTile(  
        leading: Icon(Icons.privacy_tip),  
        title: Text('Privacy Policy'),  
      ),  
    ),  
  ],  
)
```

## Challenge

Create a popup menu button that shows options for changing the theme of the app (Light, Dark, System Default).

[QUESTIONS FOR PRACTICE 5](#)

## Button Widgets in Flutter: Practice Questions



1. Create a Flutter application that displays a `MaterialButton` on the screen. When pressed, the button should change its color from blue to green.
2. Implement an `ElevatedButton` with a shadow of 10. The button should display a `SnackBar` with a message “ElevatedButton Pressed” when clicked.
3. Design a `TextButton` that toggles its text between “Enabled” and “Disabled” upon each click. Use a variable to manage the button’s state.
4. Create a `DropDownButton` with items “Item 1”, “Item 2”, and “Item 3”. Display the selected item in a `SnackBar` when an item is selected.
5. Develop a `PopupMenuButton` that shows three menu items: “Option 1”, “Option 2”, and “Option 3”. Handle the selection and display the chosen option in a `Toast` or a `SnackBar`.

## 6) \_Flutter Buttons

This section will help you to learn different types of buttons in flutter. Here you will learn the following topics:

- [Form in Flutter](#),
- [TextFormField in Flutter](#),
- [Checkbox in Flutter](#),
- [Radio Button in Flutter](#),
- [Switch in Flutter](#),
- [Date and Time Picker in Flutter](#), and
- [Form Validation in Flutter](#).

### Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

### FORM IN FLUTTER

#### Forms in Flutter

**Form** widget is used to group multiple form fields, such as textbox, checkboxes, and buttons, to create a easy data entry experience.

#### Why Forms are Useful?

Form are important for building interactive and user-friendly applications. They are useful for the following reasons:

- **Data Collection:** It is used to collect the user input data such as name, email, phone number, etc.
- **Validation:** It is used to validate the user input data such as email, phone number, etc.
- **Submission:** It is used to submit the user input data to the server.

#### Example 1: Basic Form in Flutter

In this example below, you will learn to create a form with **First Name** and **Last Name** fields.

```
Form(
  child: Column(
    children: [
      // Form fields go here
      TextFormField(
        decoration: InputDecoration(
          labelText: 'First Name',
          hintText: 'Enter your first name',
        ),
      ),
      TextFormField(
        decoration: InputDecoration(
          labelText: 'Last Name',
          hintText: 'Enter your last name',
        ),
      ),
    ],
  ),
)
```

## Example 2: Form with Validation in Flutter

In this example below, you will learn to create a form with **First Name** and **Last Name** fields with validation. You will learn more about validation in the next section.

```
final _formKey = GlobalKey<FormState>();
Form(
  key: _formKey,
  child: Column(
    children: [
      // Form fields go here
      TextFormField(
        decoration: const InputDecoration(
          labelText: 'First Name',
          hintText: 'Enter your first name',
        ),
        validator: (value) {
          if (value == "") {
            return 'Please enter your first name';
          }
          return null;
        },
      ),
      TextFormField(
```



In this example below, you will learn to implement validation to ensure the input is not empty.

```
TextFormField(
  decoration: InputDecoration(
    labelText: 'Enter your email',
  ),
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your email';
    }
    return null;
  },
)
```

### Example 3: Password Field

In this example below, you will learn to create a password field that hides the input.

```
TextFormField(
  decoration: InputDecoration(
    labelText: 'Password',
  ),
  obscureText: true,
)
```

### Example 4: Styled TextFormField

In this example below, you will learn to customize the appearance of your **TextFormField**.

```
TextFormField(
  decoration: InputDecoration(
    border: OutlineInputBorder(
      borderRadius: BorderRadius.circular(10),
    ),
    labelText: 'Enter your address',
  ),
)
```

### Example 5: TextFormField with Prefix and Suffix Icons

In this example below, you will learn to create a **TextFormField** with prefix and suffix icons.

```
TextFormField(
```

```
decoration: InputDecoration(
  labelText: 'Phone Number',
  prefixIcon: Icon(Icons.phone),
  suffixIcon: Icon(Icons.check_circle),
),
)
```

### Example 6: TextFormField with Max Length

In this example below, you will learn to create a **TextFormField** with a maximum length of 10 characters.

```
TextFormField(
  decoration: InputDecoration(
    labelText: 'Enter your address',
  ),
  maxLength: 10,
)
```

### Example 7: TextFormField with Number Keyboard

In this example below, you will learn to create a **TextFormField** that opens the number keyboard on focus.

```
TextFormField(
  decoration: InputDecoration(
    labelText: 'Enter your age',
  ),
  keyboardType: TextInputType.number,
)
```

### Example 8: TextFormField with Input Formatter

In this example below, you will learn to create a **TextFormField** with an input formatter that formats the input as a phone number. To achieve this, you will need to add the following import statement to your file:

```
import 'package:flutter/services.dart';
TextFormField(
  decoration: InputDecoration(
    labelText: 'Phone Number',
  ),
  inputFormatters: [
    FilteringTextInputFormatter.digitsOnly,
    LengthLimitingTextInputFormatter(10),
  ],
)
```

)

Input Formatters are used to format the input as it is being entered. In above example, we use **FilteringTextInputFormatter** to allow only digits and **LengthLimitingTextInputFormatter** to limit the input to 10 characters.

## Challenge

Create a **TextFormField** for user feedback that includes validation to ensure the input is at least 15 characters long. Style the field with a rounded border and include a prefix icon that represents communication.

## CHECKBOX IN FLUTTER

### Introduction

**Checkbox** is a widget that allows users to select between two states: **checked** or **unchecked**. This is typically used in forms and settings to enable or disable options.

### Example 1: Basic Checkbox

In this example, you will learn to create a simple checkbox using the Checkbox widget.

```
Checkbox(  
  value: isChecked,  
  onChanged: (bool? value) {  
    setState(() {  
      isChecked = value!;  
    });  
  },  
)
```

### Example 2: Checkbox with Custom Colors

In this example, you will learn to create a checkbox with custom colors using the Checkbox widget.

```
Checkbox(  
  value: isChecked,  
  checkColor: Colors.white, // color of tick Mark  
  activeColor: Colors.green, // background color  
  onChanged: (bool? value) {  
    setState(() {  
      isChecked = value!;  
    });  
  })
```

```
},  
)
```

### Example 3: Checkbox List

In this example, you will learn to create a group of checkboxes using the CheckboxListTile widget.

```
var options = <String>['Option 1', 'Option 2', 'Option 3', 'Option 4'];  
var selectedOptions = <String>[];  
  
ListView(  
  children: options.map((String option) {  
    return CheckboxListTile(  
      title: Text(option),  
      value: selectedOptions.contains(option),  
      onChanged: (bool? value) {  
        setState(() {  
          if (value == true) {  
            selectedOptions.add(option);  
          } else {  
            selectedOptions.remove(option);  
          }  
        });  
      },  
    );  
  }).toList(),  
)
```

### Example 4: Accept Terms & Conditions Checkbox

In this example, you will learn to create a checkbox for accepting terms and conditions using the CheckboxListTile widget.

```
Form(  
  child: Column(  
    children: [  
      CheckboxListTile(  
        title: Text("Accept Terms & Conditions"),  
        value: isAccepted,  
        onChanged: (bool? value) {  
          setState(() {  
            isAccepted = value!;  
          });  
        },  
      ),  
      // Other form elements
```

```
    ],  
  ),  
)
```

### Example 5: Interest Selection Checkbox

In this example, you will learn to find the selected interests from a list of interests using the `CheckboxListTile` widget.

```
List<String> interests = [  
  'Reading',  
  'Music',  
  'Travel',  
  'Sports',  
  'Cooking',  
];  
  
List<String> selectedInterests = [];  
  
Column(  
  children: interests.map((String interest) {  
    return CheckboxListTile(  
      title: Text(interest),  
      value: selectedInterests.contains(interest),  
      onChanged: (bool? value) {  
        setState(() {  
          if (value == true) {  
            selectedInterests.add(interest);  
          } else {  
            selectedInterests.remove(interest);  
          }  
        });  
      },  
    );  
  }).toList(),  
)
```

### Challenge

Create a checkbox group for selecting multiple programming languages from a list of languages.

```
// available programming languages  
List<String> languages = [  
  'Dart',  
  'Python',  
  'Java',  
]
```



```
'JavaScript',  
'C++',  
'C#',  
'Ruby',  
'Go',  
'Swift',  
'Kotlin',  
];
```

## RADIOBUTTON IN FLUTTER

### Introduction

**RadioButtons** are a type of input widget to select one option from a group. Checkbox allows multiple selections but Radio Button allows only one selection at a time.

### Example 1: Basic RadioButton

In this example, learn how to create a group of radio buttons for selecting **Male**, **Female** and **Others**. If you have any problem try **Run Online** button to see the code in action.

#### *Step 1: Create Enum For Options*

We will use enhanced enum to create the options for the radio buttons.

```
enum Gender {  
  male("Male"),  
  female("Female"),  
  others("Others");  
  
  // Members  
  final String text;  
  const Gender(this.text);  
}
```

#### *Step 2: Create State Variable*

It's time to create a state variable to store the selected option.

```
Gender? _selectedOption = Gender.male;
```

#### *Step 3: Create Radio Buttons*

Finally, create the radio buttons using the **RadioListTile** widget.

```
Column(  

```

```
// Radio buttons
children: Gender.values
    .map((option) => RadioListTile<Gender>(
        title: Text(option.text),
        value: option,
        groupValue: _selectedOption,
        onChanged: (value) {
            setState(() {
                _selectedOption = value;
            });
        },
    ))
    .toList(),
)
```

## Example 2: Game Difficulty Selection

In this example, learn how to create a group of radio buttons for selecting the game difficulty level. If you have any problem try **Run Online** button to see the code in action.

### Step 1: Create Enum For Options

We will use enhanced enum to create the options for the radio buttons.

```
enum Difficulty {
    easy("Easy"),
    medium("Medium"),
    hard("Hard");

    // Members
    final String text;
    const Difficulty(this.text);
}
```

### Step 2: Create State Variable

It's time to create a state variable to store the selected option.

```
Difficulty? _selectedDifficulty = Difficulty.easy;
```

### Step 3: Create Radio Buttons

Finally, create the radio buttons using the **RadioListTile** widget.

```
Column(
    // Radio buttons
    children: Difficulty.values
        .map((option) => RadioListTile<Difficulty>(
```

```

        title: Text(option.text),
        value: option,
        groupValue: _selectedDifficulty,
        onChanged: (value) {
          setState(() {
            _selectedDifficulty = value;
          });
        },
      ),
    ).toList(),
  ),
)

```

## Challenge

Create a quiz app with radio buttons to select the correct answer for each question.

```

Question: What is the capital of France?
Options:
- Paris
- London
- Berlin

```

## SWITCH IN FLUTTER

### Introduction

The **Switch** widget in Flutter is a fundamental UI component used for toggling between two states, such as on/off or true/false. This article will guide you through various implementations of the Switch widget, from basic usage to more advanced scenarios including custom styling and integration with forms.

### Example 1: Basic Switch

Learn how to create a simple switch that toggles a boolean value.

```

Switch(
  value: isSwitched,
  onChanged: (value) {
    setState(() {
      isSwitched = value;
    });
  },
),

```

### Example 2: Custom Styled Switch

Customize the appearance of a switch with custom colors.

```
Switch(  
  value: isSwitched,  
  activeTrackColor: Colors.lightGreenAccent,  
  activeColor: Colors.green,  
  onChanged: (value) {  
    setState(() {  
      isSwitched = value;  
    });  
  },  
)
```

### Example 3: Switch List

Implement a list of switches, each controlling a different setting.

```
var settings = <String, bool>{  
  'Wi-Fi': true,  
  'Bluetooth': false,  
  'Airplane Mode': false,  
  'Mobile Data': true,  
};  
  
ListView(  
  children: settings.keys.map((String key) {  
    return SwitchListTile(  
      title: Text(key),  
      value: settings[key]!,  
      onChanged: (bool value) {  
        setState(() {  
          settings[key] = value;  
        });  
      },  
    );  
  }).toList(),  
)
```

### Example 4: Form Switch

Incorporate a switch into a form to accept terms & conditions or toggle preferences.

```
Form(  
  child: Column(  
    children: [  
      SwitchListTile(  
        title: Text("Accept Terms & Conditions"),  

```

```

        value: isAccepted,
        onChanged: (bool value) {
          setState(() {
            isAccepted = value;
          });
        },
      ),
      // Other form elements
    ],
  ),
)

```

### Example 5: Profile Visibility Switch

Use a switch to control the visibility of user profile information.

```

SwitchListTile(
  title: Text('Show Profile Publicly'),
  value: isProfilePublic,
  onChanged: (bool value) {
    setState(() {
      isProfilePublic = value;
    });
  },
)

```

### Challenge

Create a settings page using switches to control various app features such as notifications, dark mode, location tracking, and automatic updates.

```

// App settings options
Map<String, bool> appSettings = {
  'Notifications': true,
  'Dark Mode': false,
  'Location Tracking': true,
  'Automatic Updates': false,
};

```

## DATETIME PICKER IN FLUTTER

### Introduction

The **DateTime Picker** is a widget that allows users to select a date, time, or both from a dialog. This is very useful for apps that require you to input dates or times, like scheduling events or setting reminders.

### Example 1: BirthDate Picker

In this example, you will learn to create a date picker for selecting a birthdate.

```
DateTime? selectedDate;

Future<void> _selectDate(BuildContext context) async {
  final DateTime? picked = await showDatePicker(
    context: context,
    initialDate: DateTime.now(),
    firstDate: DateTime(1900),
    lastDate: DateTime.now(),
  );
  if (picked != null && picked != selectedDate)
    setState(() {
      selectedDate = picked;
    });
}
```

### Example 2: Attendance Time Picker

In this example, you will learn to create a time picker for selecting attendance time.

```
TimeOfDay? selectedTime;

Future<void> _selectTime(BuildContext context) async {
  final TimeOfDay? picked = await showTimePicker(
    context: context,
    initialTime: TimeOfDay.now(),
  );
  if (picked != null && picked != selectedTime)
    setState(() {
      selectedTime = picked;
    });
}
```

### Example 3: Combined DateTime Picker

In this example, you will learn to create a combined date and time picker.

```
DateTime? selectedDateTime;
```

```

Future<void> _selectDateTime(BuildContext context) async {
  final DateTime? picked = await showDatePicker(
    context: context,
    initialDate: DateTime.now(),
    firstDate: DateTime(1900),
    lastDate: DateTime.now(),
  );
  if (picked != null) {
    final TimeOfDay? pickedTime = await showTimePicker(
      context: context,
      initialTime: TimeOfDay.now(),
    );
    if (pickedTime != null) {
      setState(() {
        selectedDateTime = DateTime(
          picked.year,
          picked.month,
          picked.day,
          pickedTime.hour,
          pickedTime.minute,
        );
      });
    }
  }
}

```

## Challenge

Create application that find age from the selected birthdate and display it in the app.

## FORM VALIDATION IN FLUTTER

### Form Validation in Flutter

If you are building a form in your Flutter application, it is essential to validate the user input data to ensure data integrity. In this section, you will learn how to validate user input data using the **TextFormField** widget in Flutter.

### Why Form Validation is Important?

- **Data Integrity:** It ensures that the data entered by the user is accurate and consistent.
- **User Experience:** It provides a better user experience by guiding the user to enter valid data.
- **Error Prevention:** It helps in preventing errors and exceptions caused by invalid data.

### Example 1: Feedback Form with Validation

In this example, you will learn to create a feedback form with a validation rule ensuring a minimum character count. If you have any problem try **Run Online** button to see the code in action.

```
// Create a GlobalKey for the form
final _formKey = GlobalKey<FormState>();

// Create a TextEditingController for the feedback field
final _feedbackController = TextEditingController();

Form(
  key: _formKey,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    children: [
      TextFormField(
        maxlines: 5,
        controller: _feedbackController,
        decoration: const InputDecoration(
          hintText: 'Enter your feedback',
          border: OutlineInputBorder(),
        ),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter your feedback';
          } else if (value.length < 10) {
            return 'Feedback must be at least 10 characters';
          }
          return null;
        },
      ),
      const SizedBox(height: 20),
      ElevatedButton(
        onPressed: () {
          if (_formKey.currentState!.validate()) {
            ScaffoldMessenger.of(context).showSnackBar(
              const SnackBar(content: Text('Feedback submitted successfully!')),
            );
            // Additional submission logic here
          }
        },
        child: const Text('Submit'),
      ),
    ],
  ),
)
```



## Example 2: Sign-Up Form with Validation

In this example, you will learn to create a sign-up form with validation to ensure users provide a valid email address and a secure password. If you have any problem try **Run Online** button to see the code in action.

```
// Create a GlobalKey for the form
final _formKey = GlobalKey<FormState>();

// Create TextEditingController for email and password fields
final _emailController = TextEditingController();
final _passwordController = TextEditingController();

Form(
  key: _formKey,
  child: Column(
    mainAxisAlignment: MainAxisAlignment.min,
    children: [
      TextFormField(
        controller: _emailController,
        decoration: const InputDecoration(
          labelText: 'Email',
          hintText: 'Enter your email',
          border: OutlineInputBorder(),
        ),
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter your email';
          } else if (!value.contains('@')) {
            return 'Please enter a valid email';
          }
          return null;
        },
      ),
      const SizedBox(height: 10),
      TextFormField(
        controller: _passwordController,
        decoration: const InputDecoration(
          labelText: 'Password',
          hintText: 'Enter your password',
          border: OutlineInputBorder(),
        ),
        obscureText: true,
        validator: (value) {
          if (value == null || value.isEmpty) {
            return 'Please enter your password';
          } else if (value.length < 6) {
```

```

        return 'Password must be at least 6 characters';
      }
      return null;
    },
  ),
  const SizedBox(height: 20),
  ElevatedButton(
    onPressed: () {
      if (_formKey.currentState!.validate()) {
        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text('Sign-up successful!')),
        );
        // Additional submission logic here
      }
    },
    child: const Text('Sign Up'),
  ),
],
),
)

```

## Challenge

Create a contact form with fields name, phone number, and email and validate according to the following rules:

```

Name: Required
Phone Number: Required, must be 10 digits
Email: Required, must be a valid email address

```

## QUESTIONS FOR PRACTICE 6

### Form Validation Practice Questions

1. Create a form with **First Name** and **Last Name** fields. Validate the form to ensure that both fields are not empty.
2. Create a form with **Email** and **Password** fields. Validate the form to ensure that the email is not empty and the password is at least 6 characters long.
3. Create a form with **Phone Number** and **Address** fields. Validate the form to ensure that the phone number is not empty and the address is at least 10 characters long.
4. Create a form with **Username** and **Confirm Password** fields. Validate the form to ensure that both fields are not empty and the password matches the confirm password.

## 7) \_Flutter Navigation

In this section, you will learn how to navigate between screens in flutter.

This will include following topics:

- [Navigation in Flutter](#)
- [Move to Another Screen.](#)
- [Move Using Named Routes.](#)
- [Pass Data to Another Screen.](#)
- [Return Data from Another Screen.](#)
- [Drawer Navigation.](#)
- [Drawer Navigation.](#)
- [Navigation Methods, and](#)
- [TabBar Navigation.](#)

## Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

## NAVIGATION IN FLUTTER

### Navigation in Flutter

Navigation is process of moving between different screens (**also known as routes or pages**) within an app. It's like going from the **Home** screen of a app to the **Contact** screen.

### How to Navigate in Flutter?

There are different ways to navigate between screens in Flutter. They are:

- Using the **Navigator**
- Using **Named Routes**
- Using **Router**

### Using Navigator

Here's a simple way to navigate to a new screen using **Navigator**. You will learn more about navigator in the next section.

```
Navigator.of(context).push(  
  MaterialPageRoute(builder: (context) => SecondScreen()),  
);
```

### Using Named Routes

Named routes are routes that are named using a string. Here's how to use named routes to navigate to a new screen.

```
// First, ensure you have defined the named route in your MaterialApp
// widget.
MaterialApp(
  routes: {
    '/': (context) => FirstScreen(),
    '/second': (context) => SecondScreen(),
  },
);

// Then, use Navigator.pushNamed to navigate to the second screen.
Navigator.pushNamed(context, '/second');
```

## Using Router

For simple apps, you can use **Navigator** to move between screens. If your app needs advance navigation, you can use a Router package like (**go\_router**).

## Conclusion

In conclusion, if you are making simple apps, use **Navigator** to move between screens. If you are making complex apps, consider using a Router package like **go\_router**.

## MOVE TO ANOTHER SCREEN

### Moving to Another Screen in Flutter

Flutter provides several ways to navigate between screens, but the most common method is using **Navigator**. Here is sample code to move to another screen:

```
Navigator.of(context).push(
  MaterialPageRoute(builder: (context) => SecondScreen()),
);
```

### Example 1: Move To Second Screen

Flutter provides the **Navigator** class to manage routes. Here's a simple way to navigate to a new screen:

#### Step 1: Define a Screen

First, define a new screen as a StatelessWidget or StatefulWidget:

```
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

```

    appBar: AppBar(
      title: Text("Second Screen"),
    ),
    body: Center(
      child: Text("Welcome to the second screen!"),
    ),
  );
}

```

*Step 2: Moving to the New Screen*

Use the **Navigator.push** method to move to the new screen:

```

Navigator.of(context).push(
  MaterialPageRoute(builder: (context) => SecondScreen()),
);

```

## Navigating Back

To return to the previous screen, use the **Navigator.pop** method:

```

Navigator.pop(context);

```

## Challenge

Create two screens, **HomeScreen** and **AboutScreen**. When you click a button on the home screen, navigate to the about screen.

## MOVE USING NAMED ROUTES

### Move Using Named Routes in Flutter

Named routes provide a clear and easy way to manage navigation between screens. Instead of navigating directly to a widget, you use a string identifier (**the route's name**) to tell Flutter which screen to show.

```

Navigator.pushNamed(context, '/routeName')

```

### Example 1: Move To Another Screen Using Named Routes

In this example below, there is **HomeScreen** and **ProfileScreen**. When you click a button on the home screen, it navigates to the profile screen. First, let's create a MaterialApp with named routes.

### Step 1: Define Named Routes

Define the named routes in the **MaterialApp** widget:

```
MaterialApp(  
  initialRoute: '/',  
  routes: {  
    '/': (context) => HomeScreen(),  
    '/profile': (context) => ProfileScreen(),  
  },  
);
```

#### Home Screen (HomeScreen)

This is the main screen of the application.

```
class HomeScreen extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Home Screen"),  
      ),  
      body: Center(  
        child: ElevatedButton(  
          child: Text('Go to Profile Page'),  
          onPressed: () {  
            // Navigation Code Here  
          },  
        ),  
      ),  
    );  
  }  
}
```

#### Profile Screen (ProfileScreen)

A page displaying profile information.

```
class ProfileScreen extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Profile Page"),  
      ),  
      body: Center(  
        child: Text("Welcome to the Profile Page!"),  
      ),  
    );  
  }  
}
```

```
    );  
  }  
}
```

*Navigate To The New Screen*

When the button is pressed, **HomeScreen** should navigate to **ProfileScreen**. Use the **Navigator.pushNamed** method to move to the new screen:

```
Navigator.pushNamed(context, '/profile');
```

## PASS DATA TO OTHER SCREEN

### Introduction

In previous section, you learned how to move between screens. In this section, you will learn how to send data between screens. This is useful when you want to pass data from one screen to another, such as a user's name, email, or any other information.

### Example 1: Greet User

In this example, you will learn to greet the user by passing their name to the new screen. First, create a simple app with two screens: **HomeScreen** and **GreetScreen**.

#### *Step 1: Define Home Screen*

This is the main screen of the application. It has a text field to enter the user's name and a button to navigate to the greet screen.

```
class HomeScreen extends StatelessWidget {  
  final TextEditingController _controller = TextEditingController();  
  
  HomeScreen({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(title: Text('Greet App')),  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children:[  
            TextField(  
              controller: _controller,  
              decoration: InputDecoration(hintText: 'Enter your name'),  
            ),  
          ],  
        ),  
      ),  
    );  
  }  
}
```

```

        ElevatedButton(
          child: Text('Greet'),
          onPressed: () {
            Navigator.of(context).push(MaterialPageRoute(
              builder: (context) => GreetScreen(name:
_controller.text),
            ));
          },
        ),
      ],
    ),
  ),
);
}
}

```

*Step 2: Define Greet Screen*

This screen displays a greeting message to the user. It receives the user's name as a parameter.

```

class GreetScreen extends StatelessWidget {
  final String name;

  const GreetScreen({Key? key, required this.name}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Greet Screen")),
      body: Center(child: Text('Hello, $name!')),
    );
  }
}

```

## Example 2: Pass Product Object Details

In this example, you will learn to pass product object details to the new screen. First, create a simple app with two screens: **HomeScreen** and **ProductScreen**.

*Step 1: Create a Product Model*

First you need to create a simple data model for the product.

```

class Product {
  final String name;
  final double price;
}

```



```
Product({required this.name, required this.price});
}
```

#### *Step 2: Sample Data*

Create a list of products to display on the home screen.

```
final product = Product(name: 'Laptop', price: 1000);
```

#### *Step 3: Define Product Screen*

This screen displays the product details. It receives the product object as a parameter.

```
class ProductScreen extends StatelessWidget {
  final Product product;

  const ProductScreen({Key? key, required this.product}) : super(key:
key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text(product.name)),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Name: ${product.name}'),
            Text('Price: \${product.price}'),
          ],
        ),
      ),
    );
  }
}
```

#### *Step 4: Navigate to Product Screen*

Navigate to the product screen and pass the product object as a parameter.

```
ElevatedButton(
  child: Text('View Product'),
  onPressed: () {
    Navigator.of(context).push(MaterialPageRoute(
      builder: (context) => ProductScreen(product: product),
    ));
  },
),
```

### Example 3: Pass Data Using Named Routes

In this example below, you will learn how to pass arguments to the second screen using named routes.

#### Defining Routes

First, define the named routes in the **MaterialApp** widget:

```
MaterialApp(  
  title: 'Named Route Navigation',  
  initialRoute: '/',  
  routes: {  
    '/': (context) => HomeScreen(),  
    '/second': (context) => SecondScreen(),  
  },  
);
```

#### Passing Arguments

When navigating to the second screen, you can pass arguments using the **Navigator.pushNamed** method:

```
Navigator.pushNamed(  
  context,  
  '/second',  
  arguments: 'Hello from the first screen!',  
);
```

And receive the arguments in the **SecondScreen** widget:

```
class SecondScreen extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    final String args = ModalRoute.of(context).settings.arguments;  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Second Screen'),  
      ),  
      body: Center(  
        child: Text(args),  
      ),  
    );  
  }  
}
```

## Example 4: Pass Data Using Named Routes

In this example, you will learn to pass product object details to the new screen using named routes. First, create a simple app with two screens: **HomeScreen** and **ProductScreen**.

### *Step 1: Create a Product Model*

First you need to create a simple data model for the product.

```
class Product {  
  final String name;  
  final double price;  
  
  Product({required this.name, required this.price});  
}
```

### *Step 2: Sample Data*

Create a list of products to display on the home screen.

```
final product = Product(name: 'Laptop', price: 1000);
```

### *Step 3: Define Routes*

Define the routes in the **MaterialApp** widget.

```
MaterialApp(  
  title: 'Named Route Navigation',  
  initialRoute: '/',  
  routes: {  
    '/': (context) => HomeScreen(),  
    '/product': (context) => ProductScreen(),  
  },  
);
```

### *Step 4: Navigate to Product Screen*

Navigate to the product screen and pass the product object as a parameter.

```
ElevatedButton(  
  child: Text('View Product'),  
  onPressed: () {  
    Navigator.pushNamed(context, '/product', arguments: product);  
  },  
),
```

### *Step 5: Receive Data*

Receive the product object in the product screen.

```
class ProductScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final Product product = ModalRoute.of(context)!.settings.arguments as
Product;
    return Scaffold(
      appBar: AppBar(title: Text(product.name)),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Name: ${product.name}'),
            Text('Price: \${product.price}'),
          ],
        ),
      ),
    );
  }
}
```

## Challenge

Create a Flutter app with 2 screens called **HomeScreen** and **ShopScreen**. Use named routes to navigate between the screens. Also pass a list of products from the **HomeScreen** to the **ShopScreen**. **HomeScreen** should have a button to navigate to the **ShopScreen**. The **ShopScreen** should display a list of products.

## RETURN DATA FROM SCREEN

### Introduction

In the previous section, you learned how to send data to another screen. Now, you will learn how to return data from the new screen to the previous screen.

### Example 1: Pizza Selection

In this example, you will learn to select a pizza from a list of pizzas. When you select a pizza, it will return the selected pizza to the previous screen.

#### Step 1: Define Home Screen

This is the main screen of the application. It has a button to navigate to the pizza selection screen and a text widget to display the selected pizza.

```
class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}
```

```

class _HomeScreenState extends State<HomeScreen> {
  String? _selectedPizza;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Pizza Selection')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            ElevatedButton(
              child: Text('Select Pizza'),
              onPressed: () async {
                final selectedPizza = await Navigator.of(context).push(
                  MaterialPageRoute(builder: (context) =>
                    PizzaSelectionScreen()),
                );
                setState(() {
                  _selectedPizza = selectedPizza;
                });
              },
            ),
            SizedBox(height: 20),
            Text('Selected Pizza: $_selectedPizza'),
          ],
        ),
      ),
    );
  }
}

```

*Step 2: Define Pizza Selection Screen*

This screen displays a list of pizzas. When you select a pizza, it will return the selected pizza to the previous screen.

```

class PizzaSelectionScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Select Pizza')),
      body: ListView(
        children: [
          ListTile(
            title: Text('Margherita'),
            onTap: () {

```

```

        Navigator.of(context).pop('Margherita');
      },
    ),
    ListTile(
      title: Text('Pepperoni'),
      onTap: () {
        Navigator.of(context).pop('Pepperoni');
      },
    ),
    ListTile(
      title: Text('Vegetarian'),
      onTap: () {
        Navigator.of(context).pop('Vegetarian');
      },
    ),
  ],
),
);
}
}

```

## Challenge

Create a app with a HomeScreen and a ColorSelectionScreen. The HomeScreen should have a button to navigate to the ColorSelectionScreen where the user can select a color. Once a color is selected, the ColorSelectionScreen should return that color to the HomeScreen, which then changes its background color accordingly.

Available Colors:

- Red
- Blue
- Green

## DRAWER NAVIGATION IN FLUTTER

### Introduction

You already learned how to create drawer in flutter. In this section, you will learn how to implement drawer navigation in flutter.

### Example 1: Simple Drawer Navigation

In this example below, there are three screens: **HomePage**, **ContactPage**, and **AboutPage**. We'll add a drawer to navigate between these screens.

### Step 1: Create Contact Page & About Page

First define **ContactPage** and **AboutPage**.

```
// Contact Page
class ContactPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Contact Page')),
      body: Center(child: Text('Contact Page')),
    );
  }
}

// About Page
class AboutPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('About Page')),
      body: Center(child: Text('About Page')),
    );
  }
}
```

### Step 2: Define Home Page with Drawer

Now in home page, add a drawer to navigate between these screens.

```
Drawer(
  child: ListView(
    padding: EdgeInsets.zero,
    children: [
      DrawerHeader(
        child: Text('My App'),
        decoration: BoxDecoration(
          color: Colors.blue,
        ),
      ),
      ListTile(
        title: Text('Home'),
        onTap: () {
          Navigator.pop(context);
        },
      ),
      ListTile(
        title: Text('Contact'),
```

```

        onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) =>
ContactPage()));
        },
    ),
    ListTile(
        title: Text('About'),
        onTap: () {
            Navigator.push(context, MaterialPageRoute(builder: (context) =>
AboutPage()));
        },
    ),
],
),
),
),

```

## Challenge

Click on **Try Online** and create new screen called **SettingsPage**. Add a new item to the drawer to navigate to the **SettingsPage**.

## NAVIGATION METHODS IN FLUTTER

# Introduction

In this section, you will learn about useful navigation methods in Flutter. Here are some useful navigation methods:

## Navigator.push()

**Navigator.push()** is used to navigate to a new screen by pushing it onto the navigation stack.

### Example 1: Using Navigator.push()

In this example below, you will learn how to navigate to a new screen using **Navigator.push()**.

```
// Navigator.push example
Navigator.push(
  context,
  MaterialPageRoute(builder: (context) => AboutScreen()),
);
```

## Navigator.pushNamed()



**Navigator.pushNamed()** is used to navigate to a new screen using a named route. Named routes are routes that are named using a string.

### Example 2: Using Navigator.pushNamed()

In this example below, you will learn how to navigate to a new screen using **Navigator.pushNamed()**.

```
Navigator.pushNamed(context, '/about');
```

### Navigator.pop()

**Navigator.pop()** is used to return to the previous screen, effectively popping the current screen off the navigation stack. It's akin to the back button on a device.

### Example 3: Using Navigator.pop()

```
Navigator.pop(context);
```

### Navigator.pushReplacement()

This method replaces the current screen with a new one on the navigation stack, which is useful in scenarios like a login flow where you do not want the user to return to the login screen.

### Example 4: Using Navigator.pushReplacement()

```
Navigator.pushReplacement(  
  context,  
  MaterialPageRoute(builder: (context) => HomeScreen()),  
);
```

### Navigator.pushAndRemoveUntil()

**Navigator.pushAndRemoveUntil()** pushes a new screen and removes all the previous screens until a certain condition is met. This method is handy for logging out a user, where you want to remove all screens from the stack and navigate back to the login screen.

### Example 5: Using Navigator.pushAndRemoveUntil()

```
Navigator.pushAndRemoveUntil(  
  context,  
  MaterialPageRoute(builder: (context) => WelcomeScreen()),  
  (Route<dynamic> route) => false,
```

```
);
```

## TABBAR IN FLUTTER

### TabBar In Flutter

TabBar is a widget that displays a row of tabs at the top or bottom of the app, allowing users to switch between different views or pages by tapping on them.

### Example 1: Courier Tracking App

In this example, you will learn how to create a simple courier tracking app using TabBar. The app will have three tabs (**In Transit, Delivered, and Pending**) to display the status of the courier.

```
DefaultTabController(  
  length: 3,  
  child: Scaffold(  
    appBar: AppBar(  
      title: Text('Courier Tracking'),  
      bottom: TabBar(  
        tabs: [  
          Tab(icon: Icon(Icons.directions_car), text: 'In Transit'),  
          Tab(icon: Icon(Icons.check), text: 'Delivered'),  
          Tab(icon: Icon(Icons.access_time), text: 'Pending'),  
        ],  
      ),  
    ),  
    body: TabBarView(  
      children: [  
        Center(child: ListView(children: [Text('Parcel 21'), Text('Parcel  
22')]))),  
        Center(child: ListView(children: [Text('Parcel 23'), Text('Parcel  
24')]))),  
        Center(child: ListView(children: [Text('Parcel 26'), Text('Parcel  
27')]))),  
      ],  
    ),  
  ),  
)
```

**Note:** When you run online, this code is slightly modified to enhance the presentation of the parcels.

### Example 2: Todo App

In this example below, you will learn how to create a Todo App with a TabBar that has two tabs (**Active and Completed**) to display the tasks based on their status. Use the TabBarView to display the list of tasks for each tab.

```
DefaultTabController(  
  length: 2,  
  child: Scaffold(  
    appBar: AppBar(  
      title: Text('Todo App'),  
      bottom: TabBar(  
        tabs: [  
          Tab(text: 'Active'),  
          Tab(text: 'Completed'),  
        ],  
      ),  
    ),  
    body: TabBarView(  
      children: [  
        Center(child: ListView(children: [  
          ListTile(title: Text('Task 1'), leading: Icon(Icons.info)),  
          ListTile(title: Text('Task 2'), leading: Icon(Icons.info)),  
        ])),  
        Center(child: ListView(children: [  
          ListTile(title: Text('Task 3'), leading: Icon(Icons.info)),  
          ListTile(title: Text('Task 4'), leading: Icon(Icons.info)),  
        ])),  
      ],  
    ),  
  ),  
)
```

## Challenge

Look at the example 2 and add trailing icons to move tasks between tabs. For example, when you tap on the trailing icon of a task in the **Active** tab, it should move to the **Completed** tab. Similarly, when you tap on the trailing icon of a task in the **Completed** tab, it should move to the **Active** tab.

## QUESTIONS FOR PRACTICE 7

### Flutter Navigation Practice Questions

- Create a basic Flutter app with three screens: Home, About, and Contact. Implement navigation between these screens using MaterialPageRoute.
- Develop a Flutter application with a bottom navigation bar. Ensure that switching between the tabs does not rebuild the pages.

- Implement named route navigation in a Flutter app. Create a menu with options, and each option should navigate to a different screen using named routes.
- Build a multi-page form in Flutter where each step is on a different screen. Use named routes to navigate between the steps and pass data across these screens.
- Design a master-detail interface in Flutter where selecting an item from a list on the master page navigates to a detail page with more information about the item.
- Create a Flutter app with a drawer menu. Each item in the drawer should navigate to a different screen using named routes.
- Implement an authentication flow in Flutter. After login, redirect the user to the home screen, and ensure the back button does not navigate back to the login screen.
- Develop a Flutter app with a nested navigation scenario where a tab navigator is nested inside a bottom navigation bar, and demonstrate switching between nested routes.
- Create a Flutter app with a custom transition animation between screens. Use named routes and customize the `MaterialPageRoute` to achieve this.
- Build a Flutter application with an onboarding flow. Use named routes to navigate through the onboarding screens, and after the last screen, navigate to the main content.

## 8) \_Build Flutter Apps

This section will help you to build different apps in flutter. Here you will learn to build the following apps:

- [My Profile App](#)
- [Simple Interest Calculator App](#)
- [Todo App](#)
- [Book Reader App](#)
- [Tic Tac Toe Game](#)

### Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

#### PROFILE APP

### Profile App Using Flutter

This guide will walk you through creating a Profile App using Flutter. The app will display user information in a clean and simple layout.

#### 1. Setup and Project Creation

Ensure you have Flutter installed on your system. Create a new Flutter project with this command:

```
flutter create profile_app
```

## 2. Designing the User Interface

Replace the **lib/main.dart** file with the following code to set up the user interface:

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

void main() {
  runApp(const ProfileApp());
}

class ProfileApp extends StatelessWidget {
  const ProfileApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Profile'),
        ),
        body: const ProfilePage(),
      ),
    );
  }
}

class ProfilePage extends StatelessWidget {
  const ProfilePage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      padding: const EdgeInsets.all(20.0),
      child: Column(
        children: <Widget>[
          const CircleAvatar(
            radius: 80,
            backgroundImage: NetworkImage(
              'https://avatars.githubusercontent.com/u/33576285?v=4'),
          ),
          const SizedBox(height: 20),
          Text('Bishworaj Poudel',
```

```

        style: Theme.of(context)
            .textTheme
            .bodyLarge
            ?.copyWith(fontWeight: FontWeight.bold)),
const SizedBox(height: 10),
Text(
    'I love teaching students and helping them to achieve their
dreams.',
    textAlign: TextAlign.center,
    style: Theme.of(context).textTheme.bodyLarge),
const SizedBox(height: 20),
Card(
    elevation: 4.0,
    child: Column(
        children: <Widget>[
            const ListTile(
                leading: Icon(Icons.cake),
                title: Text('Birth Date'),
                subtitle: Text('1997-05-14'),
            ),
            ListTile(
                leading: const Icon(Icons.web),
                title: const Text('Website'),
                subtitle: const Text('brp.com.np'),
                onTap: () {}),
            const ListTile(
                leading: Icon(Icons.email),
                title: Text('Email'),
                subtitle: Text('bishworajpoudeofficial@gmail.com'),
            ),
            const ListTile(
                leading: Icon(Icons.location_on),
                title: Text('Address'),
                subtitle: Text('Pokhara, Nepal'),
            ),
        ],
    ),
),
const SizedBox(height: 20),
Wrap(
    spacing: 10,
    children: <Widget>[
        IconButton(
            icon: const Icon(Icons.facebook),
            onPressed: () {},
            color: Colors.blue,
            tooltip: 'Facebook',
        ),
    ],
),

```

```

        IconButton(
          icon: const Icon(Icons.link),
          onPressed: () {},
          color: Colors.blue,
          tooltip: 'LinkedIn',
        ),
        IconButton(
          icon: const Icon(Icons.code),
          onPressed: () {},
          color: Colors.black,
          tooltip: 'GitHub',
        ),
      ],
    ),
  ),
);
}
}

```

## Run the App

To run the app, execute `flutter run` in your terminal. Alternatively, in Visual Studio Code, you can start debugging by pressing `F5`.

This guide gives you a basic structure for a Profile App. You can customize and enhance this app by adding more user information, styling, and interactivity.

## Challenge

Create your profile app with your information and customize the UI to your liking. You can add more details, such as education, work experience, and hobbies.

## INTEREST CALCULATOR APP

### Simple Interest Calculator App Using Flutter

This guide will help you create a Simple Interest Calculator using Flutter. You'll learn to build a user interface for inputting values, calculate simple interest, and display the result.

## 1. Setup and Project Creation

Ensure Flutter is installed on your system. Begin by creating a new Flutter project:

```
flutter create simple_interest_calculator
```

## 2. Designing the User Interface

Navigate to `lib/main.dart` and replace its content with the following code to design the interface:

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

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

class SimpleInterestApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Simple Interest Calculator',
      home: InterestCalculatorScreen(),
    );
  }
}

class InterestCalculatorScreen extends StatefulWidget {
  @override
  _InterestCalculatorScreenState createState() =>
    _InterestCalculatorScreenState();
}

class _InterestCalculatorScreenState extends
State<InterestCalculatorScreen> {
  final TextEditingController principalController =
    TextEditingController();
  final TextEditingController rateController = TextEditingController();
  final TextEditingController timeController = TextEditingController();
  String result = '';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Simple Interest Calculator')),
      body: Container(
        padding: EdgeInsets.all(20),
        child: Column(
          children: [
            _buildTextField(principalController, 'Principal'),
            _buildTextField(rateController, 'Rate of Interest'),

```



```

        _buildTextField(timeController, 'Time in Years'),
        SizedBox(height: 20),
        MaterialButton(
          child: Text('Calculate'),
          onPressed: _calculateInterest,
        ),
        SizedBox(height: 20),
        Text(result, style: TextStyle(fontSize: 20)),
      ],
    ),
  ),
);
}

Widget _buildTextField(TextEditingController controller, String label) {
  return TextField(
    controller: controller,
    decoration: InputDecoration(labelText: label),
    keyboardType: TextInputType.number,
  );
}

void _calculateInterest() {
  double principal = double.tryParse(principalController.text) ?? 0.0;
  double rate = double.tryParse(rateController.text) ?? 0.0;
  double time = double.tryParse(timeController.text) ?? 0.0;

  double interest = principal * rate * time / 100;

  setState(() {
    result = 'Simple Interest: \${interest.toStringAsFixed(2)}';
  });
}
}

```

## Explanation

- **SimpleInterestApp**: The root widget that sets up the MaterialApp.
- **InterestCalculatorScreen**: A StatefulWidget that contains the UI for the calculator.
- **TextEditingController**: Controllers for handling text input.
- **\_calculateInterest**: Function to calculate the simple interest.

## User Interface Components

- Text fields for entering the principal amount, rate of interest, and time.
- A button to trigger the calculation.
- A text widget to display the result.

## Run the App

To run the application in Visual Studio Code, press `F5` or navigate to `Run > Start Debugging`. Alternatively, use the following command in your terminal:

```
flutter run
```

## Challenge

Create a new screen to display the result in a more visually appealing manner. You can use a **Card** widget to display the result.

[TODO APP](#)

## Todo App Using Flutter

This guide will walk you through the steps to create a simple Todo application using Flutter. You will learn how to create a basic UI, manage state, and implement basic functionalities.

### 1. Setup and Project Creation

First, make sure you have Flutter installed. Then, create a new Flutter project:

```
flutter create todo_app
```

### 2. Designing the User Interface

Now, it's time to design the user interface. Go to the `lib/main.dart` file and replace the code with the following:

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

void main() {
  runApp(TodoApp());
}

class TodoApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Simple Todo App',
      home: TodoListScreen(),
    );
  }
}
```

```

    }
}

class TodoListScreen extends StatefulWidget {
  @override
  _TodoListScreenState createState() => _TodoListScreenState();
}

class _TodoListScreenState extends State<TodoListScreen> {
  List<String> tasks = [];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Todo List')),
      body: ListView.builder(
        itemCount: tasks.length,
        itemBuilder: (context, index) => ListTile(title:
Text(tasks[index])),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () => _addTask(),
        child: Icon(Icons.add),
      ),
    );
  }

  void _addTask() {
    // Prompt user to enter a task
    TextEditingController controller = TextEditingController();
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        title: Text('New Task'),
        content: TextField(controller: controller),
        actions: [
          MaterialButton(
            child: Text('Add'),
            onPressed: () {
              setState(() {
                tasks.add(controller.text);
                controller.clear();
              });
              Navigator.of(context).pop();
            },
          ),
        ],
      ),
    );
  }
}

```

```
}  
  }  
);  
}
```

## Explanation

- **TodoApp**: The root widget that sets up the MaterialApp.
- **TodoListScreen**: A StatefulWidget that manages the list of tasks.
- **tasks**: A list of strings to hold tasks.
- **ListView.builder**: Dynamically creates a list of tasks.
- **FloatingActionButton**: A button to add new tasks.
- **\_addTask**: Function to show a dialog for adding a new task.

## Run the App

To run the app in vs code, press `F5` or go to `Run > Start Debugging`. You can also run the app using the following command:

```
flutter run
```

## Challenge

Create a delete functionality for the tasks. When a user taps on a task, it should be removed from the list.

## BOOK READER APP

### Online PDF Book Reader App Using Flutter

In this guide, you'll learn how to create an online PDF book reader app using Flutter. This app will allow users to view PDF books from a URL.

#### 1. Setup and Project Creation

To start, ensure Flutter is installed on your system. Create a new Flutter project with the following command:

```
flutter create pdf_reader_app
```

#### 2. Adding Dependencies

Open your `pubspec.yaml` file and add the following dependency to include the PDF rendering package:

```
dependencies:
```

```
flutter:  
  sdk: flutter  
flutter_pdfview: ^1.0.4+2
```

### 3. Designing the User Interface

Replace the code in `lib/main.dart` with the following to set up the user interface:

```
import 'package:flutter/material.dart';  
import 'package:flutter_pdfview/flutter_pdfview.dart';  
  
void main() => runApp(PDFReaderApp());  
  
class PDFReaderApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      title: 'PDF Reader App',  
      theme: ThemeData(  
        primarySwatch: Colors.blue,  
      ),  
      home: PDFViewPage(),  
    );  
  }  
}  
  
class PDFViewPage extends StatefulWidget {  
  @override  
  _PDFViewPageState createState() => _PDFViewPageState();  
}  
  
class _PDFViewPageState extends State<PDFViewPage> {  
  final String pdfUrl =  
    'http://www.africau.edu/images/default/sample.pdf';  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Online PDF Book Reader'),  
      ),  
      body: PDFView(  
        filePath: pdfUrl,  
        enableSwipe: true,  
        swipeHorizontal: true,  
        autoSpacing: false,  
        pageFling: true,  
      ),  
    );  
  }  
}
```

```
    ),  
  );  
}  
}
```

## Explanation

- **PDFReaderApp**: The root widget initializing the MaterialApp.
- **PDFViewPage**: A StatefulWidget that displays the PDF reader.
- **PDFView**: A widget from `flutter_pdfview` used to render the PDF.

## User Interface Components

- A `PDFView` widget is used to display the PDF file.
- The app bar to provide a title for the app.

## Run the App

To run the app, use the `flutter run` command. You can also run the app in Visual Studio Code by pressing `F5`.

## TIC TAC TOE GAME

### Tic Tac Toe Game Using Flutter

In this guide, we'll create a simple Tic Tac Toe game using Flutter. We'll use a `GridView.builder` to create the game grid and manage the game state.

#### 1. Setup and Project Creation

First, ensure Flutter is installed on your machine. Create a new Flutter project with the following command:

```
flutter create tic_tac_toe_game
```

#### 2. Designing the User Interface

Edit the `lib/main.dart` file with the following code to design the game's interface:

```
import 'package:flutter/material.dart';  
  
void main() => runApp(TicTacToeApp());  
  
class TicTacToeApp extends StatelessWidget {  
  @override
```

```

Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Tic Tac Toe',
    home: TicTacToeScreen(),
  );
}

class TicTacToeScreen extends StatefulWidget {
  @override
  _TicTacToeScreenState createState() => _TicTacToeScreenState();
}

class _TicTacToeScreenState extends State<TicTacToeScreen> {
  List<String> board = List.filled(9, '', growable: false);
  String currentPlayer = 'X';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Tic Tac Toe')),
      body: GridView.builder(
        gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount: 3),
        itemCount: board.length,
        itemBuilder: (context, index) {
          return GestureDetector(
            onTap: () => _markCell(index),
            child: Container(
              decoration: BoxDecoration(
                border: Border.all(color: Colors.black),
              ),
              child: Center(
                child: Text(
                  board[index],
                  style: TextStyle(fontSize: 40),
                ),
              ),
            ),
          );
        },
      ),
    );
  }

  void _markCell(int index) {
    if (board[index] == '') {
      setState(() {

```

```

        board[index] = currentPlayer;
        currentPlayer = currentPlayer == 'X' ? 'O' : 'X';
    });
}
}
}

```

## Explanation

- **TicTacToeApp**: Sets up the MaterialApp.
- **TicTacToeScreen**: A StatefulWidget that contains the UI and logic for the game.
- **board**: A list that represents the 3x3 grid of the game.
- **GridView.builder**: Used to create the game grid.
- **\_markCell**: A function to mark a cell with the current player's symbol.

## Game Logic

- The game alternates between two players, 'X' and 'O'.
- Players tap on a cell to mark it with their symbol.
- The game continues until all cells are marked.

## Run the App

To run the application, use the **flutter run** command, or press **F5** in Visual Studio Code to start debugging.

## QUESTIONS FOR PRACTICE 8

### Flutter Practice Questions

- Create a **Counter App** that allows users to increment and decrement a counter, introducing you to the fundamentals of state management in Flutter.
- Create a **Note-taking App** for writing and saving notes, teaching you about file handling and persistent storage in Flutter.
- Create a **Currency Converter App** that converts currencies, involving fetching current exchange rates from an API and applying them in your application.
- Create a **Stopwatch App** with functionalities like start, stop, and reset, which will help you learn about timers and updating the UI in real-time in Flutter.
- Create a **Basic Calculator App** that performs basic arithmetic operations. This project will help you understand the nuances of layout design and handling user input in Flutter.

## 9) \_Flutter Local Storage

This section will help you to learn about local storage in Flutter. Here you will learn the following topics:



- [Local Storage in Flutter](#)
- [Shared Preferences in Flutter](#)

## Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

## LOCAL STORAGE IN FLUTTER

### Introduction

Local storage is like the **memory of your app**. Think about it:

- 🗑️ You have a to-do app.
- ✕ Every time you close the app, your tasks are gone.
- 📁 Local storage is solution to this problem.

### Why Do We Need Local Storage?

Wondering why local storage is a big deal? Here's why:

- **Persistence:** Keep data even after the app is closed.
- **Offline Access:** Users can access their data without internet.
- **Performance:** Data on the device means faster access.
- **User Experience:** Apps that remember preferences = Happy users!

## 3. Flutter's Local Storage Options

There are different ways to store data locally in Flutter. Here are the most popular ones:

Option	Description
<b>SharedPreferences</b>	Great for small data like settings. Like a tiny storage box.
<b>Flutter Secure Storage</b>	For storing sensitive data like passwords. Encrypted & secure.
<b>Hive</b>	A blend of simplicity & performance. Think of it as Flutter's magic.
<b>SQLite</b>	A lightweight database. Ideal when data needs structure.

When you're starting out, it's best to start with **SharedPreferences**. It's simple and easy to use. In upcoming sections, you will learn about each of these options in detail.

## SHARED PREFERENCES IN FLUTTER

### Shared Preferences in Flutter

Shared preferences is a simple key-value pair storage feature in Flutter that's frequently used to permanently store small amounts of basic data. It works well for storing preferences, settings, and little quantities of user-specific information.

## Steps To Use Shared Preferences in Flutter

Follow the below steps to use shared preferences in flutter.

- Add **shared\_preferences** package in your pubspec.yaml file.

```
dependencies:  
  flutter:  
    sdk: flutter  
  shared_preferences: ^2.2.2
```

### Step 1: Import Package

To use shared preferences in flutter, you need to import the **shared\_preferences** package. You can import the **shared\_preferences** package in **lib/main.dart** file. Add the following code in **lib/main.dart** file.

```
import 'package:shared_preferences/shared_preferences.dart';
```

### Step 2: Create a SharedPreferences Helper Class

In this step, you will create a helper class called **shared\_preferences\_helper.dart**. This class will be used to store and retrieve data from shared preferences. You can create this class in **lib/helper/shared\_preferences\_helper.dart** file. Add the following code in **lib/helper/shared\_preferences\_helper.dart** file.

```
import 'package:shared_preferences/shared_preferences.dart';  
  
class SharedPreferencesHelper {  
  static SharedPreferences? _sharedPreferences;  
  
  static Future<void> init() async {  
    if (_sharedPreferences == null) {  
      _sharedPreferences = await SharedPreferences.getInstance();  
    }  
  }  
  
  // To set a string value  
  static Future<bool> setString(String key, String value) {  
    return _sharedPreferences!.setString(key, value);  
  }  
}
```

```
// To get a string value
static String? getString(String key) {
    return _sharedPreferences!.getString(key);
}

// Similarly, you can create methods for other data types.
}
```

### Step 3: Initialize SharedPreferences

Before using shared preferences, initialize it as a priority in the main() method. You can initialize shared preferences in **lib/main.dart** file. Add the following code in **lib/main.dart** file.

```
void main() async {
    WidgetsFlutterBinding.ensureInitialized();
    await SharedPreferencesHelper.init();
    runApp(const MyApp());
}
```

### Step 4: Set and Get Values

Now you can set and get values from shared preferences. You can set and get values from shared preferences as shown below.

```
// To set a string value
await SharedPreferencesHelper.setString('username', 'FlutterFan123');

// To get a string value
String? username = SharedPreferencesHelper.getString('username');
```

### Example 1: Light and Dark Theme App

In this example below, you will build a light and dark theme app by using shared preferences. You will use shared preferences to store the theme value and retrieve it when the app is opened.

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

void main() async {
    WidgetsFlutterBinding.ensureInitialized();
    await SharedPreferencesHelper.init();
    runApp(MyApp());
}
```

```

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  bool _isDarkTheme = false;

  // initialize the theme from shared preferences
  @override
  void initState() {
    super.initState();
    _isDarkTheme = SharedPreferencesHelper.getBool("isDarkTheme") ??
false;
  }

  void _toggleTheme() {
    setState(() {
      _isDarkTheme = !_isDarkTheme;
      SharedPreferencesHelper.setBool("isDarkTheme", _isDarkTheme);
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Theme Switcher',
      theme: _isDarkTheme ? ThemeData.dark() : ThemeData.light(),
      home: Scaffold(
        appBar: AppBar(title: Text('Theme Switcher with
SharedPreferences')),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text('Click the button below to toggle theme:'),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: _toggleTheme,
                child: Text('Toggle Theme'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

```

```

}

class SharedPreferencesHelper {
  static SharedPreferences? _sharedPreferences;

  static Future<void> init() async {
    if (_sharedPreferences == null) {
      _sharedPreferences = await SharedPreferences.getInstance();
    }
  }

  // To set a bool value
  static Future<bool> setBool(String key, bool value) {
    return _sharedPreferences!.setBool(key, value);
  }

  // To get a bool value
  static bool? getBool(String key) {
    return _sharedPreferences!.getBool(key);
  }
}

```

**Note:** Close and Reopen the app to see the changes.

## QUESTIONS FOR PRACTICE 9

### Flutter Local Storage Practice Questions

- Build a settings page in a Flutter app that saves user preferences using Shared Preferences. Include options like theme color and notification settings.
- Develop a todo list application where tasks are stored and retrieved from local storage, ensuring data persistence even after the app restarts.
- Design a user profile screen where users can enter and save their information (like name, email, and profile picture). Use local storage to save the data.
- Create an app for tracking game scores that stores the high scores locally using Shared Preferences, allowing users to view their best scores over time.
- Build an app that allows users to select their preferred language for the app interface, and save this preference using Shared Preferences.
- Develop a Flutter app with the functionality to switch between light and dark themes, saving the user's preference in Shared Preferences.
- Build a simple shopping app where the user's cart items are saved in local storage, ensuring the cart remains intact even if the app is closed and reopened.
- Design an app where users can create notes and categorize them. Use local storage to save notes and their respective categories.
- Implement an onboarding process for first-time users and use Shared Preferences to store a flag indicating that the user has completed the onboarding.
- Build a fitness app that tracks daily steps or exercise routines and saves the data locally, allowing users to view their progress over time.

## 10) \_Working with API in Flutter

This section will help you to learn about working with API in Flutter. Here you will learn the following topics:

- [Introduction to API and REST API](#),
- [Introduction to JSON](#),
- [Fetch Data from REST API](#),
- [Post Data to REST API](#),
- [Put Data to REST API](#),
- [Delete Data from REST API](#), and
- [Create Quiz App](#).

### Practice Questions

Complete this section & [practice this question](#) to improve and test your flutter skill.

#### API AND REST API

### What is an API?

An API, or Application Programming Interface, is a set of rules for building and interacting with software applications. It defines the methods and data formats for requesting and exchanging information between different software programs, enabling them to communicate and share functionalities efficiently.

### What is REST API?

A REST API, or Representational State Transfer API, is a type of API that adheres to the REST architectural style. It uses HTTP requests to access and use data, with the operations including GET, POST, PUT, and DELETE.

### Principles of REST

- **Client-server architecture:** Separation of client and server functionalities to improve portability and scalability of the application.
- **Stateless:** Each request from the client to the server must contain all the information the server needs to understand and complete the request.
- **Cacheable:** Responses must define themselves as cacheable or not to prevent clients from reusing stale or inappropriate data.
- **Uniform interface:** Simplifies and decouples the architecture, which enables each part to evolve independently.
- **Layered system:** The client cannot tell whether it is connected directly to the end server or to an intermediary.
- **Code on demand (optional):** Servers can temporarily extend or customize the functionality of a client by transferring executable code.

## How Do REST APIs Work?

When a client requires data or functionality, it sends a request to the server using a URL. The server processes the request, and then sends back a response. This response includes a status line indicating success or failure and the requested resource in the format specified by the request.

## Components of a RESTful API Request

- **Resource identifier (URL):** Uniquely identifies the resource being requested.
- **HTTP Method:** Defines the operation to be performed (GET, POST, PUT, DELETE, etc.).
- **Headers:** Provide metadata for the HTTP request, including content type, authentication, etc.
- **Body (optional):** Contains data sent with the request (for POST, PUT methods).

## Benefits of RESTful APIs

- **Scalability:** Efficient client-server interactions and stateless operations facilitate scalability.
- **Flexibility and Independence:** The separation of client and server, along with the stateless nature of requests, allows various parts of an application to evolve independently.
- **Widespread Use:** Given its efficiency, simplicity, and support across programming languages, REST APIs are widely used for web services, including social media platforms, mobile applications, and IoT devices.

## HTTP Status Codes

Here are some common HTTP status codes and their meanings:

- **200 OK:** The request was successful.
- **201 Created:** The request has been fulfilled and resulted in a new resource being created.
- **400 Bad Request:** The server cannot process the request due to a client error.
- **401 Unauthorized:** The request has not been applied because it lacks valid authentication credentials.
- **404 Not Found:** The server cannot find the requested resource.
- **500 Internal Server Error:** The server encountered an unexpected condition that prevented it from fulfilling the request.

## INTRODUCTION TO JSON

### What is JSON?

JSON (JavaScript Object Notation) is a lightweight data-interchange format. Mostly it is used to transfer data between computers.

## Characteristics of JSON

- **Lightweight:** JSON is designed to be easy to use and quick to interpret, making it ideal for data interchange.
- **Self-describing:** JSON structures are clear and understandable, making it easy to read and write.
- **Language Independent:** Although derived from JavaScript, JSON is language-independent, with many programming languages providing support for parsing and generating JSON data.

**Note:** JSON files have a **.json** extension.

## Use Cases of JSON

- **App and Web Development:** JSON is extensively used in app and web development for transferring data between a server and a client.
- **APIs and Web Services:** Many web services and APIs exchange data using JSON format due to its straightforward and efficient structure.

## JSON Data Types

JSON supports various data types, including:

- **String:** A sequence of characters wrapped in quotes.
- **Number:** Numeric data (integer or floating point).
- **Boolean:** Represents a truth value (**true** or **false**).
- **Null:** Represents a null value.
- **Array:** An ordered list of values, enclosed in square brackets `[]`.
- **Object:** A collection of key-value pairs, enclosed in curly braces `{}`.

## JSON Syntax Rules

- Data is in name/value pairs.
- Curly braces `{}` hold objects, while square brackets `[]` hold arrays.
- Data is separated by commas, and each name in an object is followed by a colon `:` which precedes the value.

Example of JSON Data:

```
{
  "name": "John Doe",
  "age": 30,
  "isEmployed": true,
  "address": {
    "street": "123 Main St",
    "city": "Anytown"
  },
  "phoneNumbers": ["123-456-7890", "456-789-0123"]
}
```



```
}
```

## JSON vs. XML

JSON and XML are both used for data interchange, but JSON is generally preferred due to its simpler syntax, faster parsing, and lighter data format.

## Parsing JSON with Dart

To parse JSON in Dart, you can use the `jsonDecode` function from the `dart:convert` package.

Example of Parsing JSON in Dart:

```
import 'dart:convert';

void main() {
  String jsonString = '''
  {
    "name": "John Doe",
    "age": 30,
    "isEmployed": true
  }
  ''';

  var decodedJson = jsonDecode(jsonString);

  print('Name: ${decodedJson['name']}');
  print('Age: ${decodedJson['age']}');
  print('Is Employed: ${decodedJson['isEmployed']}');
}
```

**Note:** Dart's `jsonDecode` function returns a `Map<String, dynamic>`, allowing for easy access to JSON properties.

## FETCH DATA FROM REST API

### Introduction

Fetching data from a REST API is a fundamental task in modern app development. It allows your app to interact with external data sources, providing dynamic content to users. This guide covers how to fetch data from a REST API in Flutter, including setup, making network requests, and handling responses.

### Setup

First, add the `http` package to your `pubspec.yaml` file to make HTTP requests:

```
dependencies:
  flutter:
    sdk: flutter
  http: ^1.2.1
```

## Example 1: Posts Listing App

In this example below, you will learn how to fetch data from a REST API and display it in your Flutter app.

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: PostList(),
    );
  }
}

class PostList extends StatefulWidget {
  @override
  _PostListState createState() => _PostListState();
}

class _PostListState extends State<PostList> {
  List<dynamic> _postData = [];

  Future<void> fetchData() async {
    final response = await
http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));

    if (response.statusCode == 200) {
      // If the server returns a 200 OK response, parse the JSON.
      setState(() {
        _postData = jsonDecode(response.body);
      });
    } else {
```

```

        // If the server did not return a 200 OK response,
        // throw an exception.
        throw Exception('Failed to load data');
    }
}

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: const Text('Fetch Data Example'),
        ),
        body: Center(
            child: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                    ElevatedButton(
                        onPressed: fetchData,
                        child: const Text('Fetch Data'),
                    ),
                    const SizedBox(height: 20),
                    Expanded(
                        child: ListView.builder(
                            itemCount: _postData.length,
                            itemBuilder: (context, index) {
                                return ListTile(
                                    title: Text(_postData[index]['title']),
                                );
                            },
                        ),
                    ),
                ],
            ),
        ),
    );
}
}

```

## Challenges

Create a motivational quotes app that fetches quotes from a REST API and displays them in a list. Here is URL to fetch quotes from:

<https://jsonguide.technologychannel.org/quotes.json>

## POST DATA TO REST API

### Introduction

Posting data to a REST API is a common requirement for mobile applications, enabling them to send information to a server for processing or storage. This guide demonstrates how to make POST requests in Flutter, using the `http` package.

### Setup

Ensure the `http` package is included in your `pubspec.yaml` for making HTTP requests:

```
dependencies:  
  flutter:  
    sdk: flutter  
  http: ^1.2.1
```

### Example: Creating a New Post

Here's how you can post data to create a new post in a JSON placeholder API:

```
import 'package:flutter/material.dart';  
import 'package:http/http.dart' as http;  
import 'dart:convert';  
  
void main() {  
  runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: CreatePostExample(),  
    );  
  }  
}  
  
class CreatePostExample extends StatefulWidget {  
  @override  
  _CreatePostExampleState createState() => _CreatePostExampleState();  
}  
  
class _CreatePostExampleState extends State<CreatePostExample> {  
  Future<void> createPost(String title, String body) async {  
    final response = await http.post(  
      Uri.parse('https://jsonplaceholder.typicode.com/posts'),  
      headers: {'Content-Type': 'application/json'},  
      body: jsonEncode({'title': title, 'body': body}),  
    );  
  }  
}
```

```

Uri.parse('https://jsonplaceholder.typicode.com/posts'),
headers: <String, String>{
    'Content-Type': 'application/json; charset=UTF-8',
},
body: jsonEncode(<String, String>{
    'title': title,
    'body': body,
    'userId': '1',
}),
);

if (response.statusCode == 201) {
    // If the server returns a 201 CREATED response, then the post was
    // successfully created.
    final responseBody = jsonDecode(response.body);
    showDialog(
        context: context,
        builder: (BuildContext context) {
            return AlertDialog(
                title: Text("Post Created"),
                content: Text("New post ID: ${responseBody['id']}"),
                actions: [
                    MaterialButton(
                        child: Text("OK"),
                        onPressed: () {
                            Navigator.of(context).pop();
                        },
                    ),
                ],
            );
        },
    );
} else {
    // If the server did not return a 201 CREATED response,
    // then throw an exception.
    throw Exception('Failed to create post');
}
}

@override
Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
            title: const Text('Post Data Example'),
        ),
        body: Center(
            child: ElevatedButton(

```

```

        onPressed: () => createPost('Flutter', 'Posting to API from
Flutter'),
        child: const Text('Create Post'),
      ),
    ),
  );
}
}

```

## PUT DATA TO REST API IN FLUTTER

### Introduction

Updating data on a server through a REST API is a common requirement for mobile applications. PUT requests are used for updating existing resources. This tutorial shows how to send PUT requests in Flutter using the `http` package.

### Setup

Add the `http` package to your `pubspec.yaml` file:

```

dependencies:
  flutter:
    sdk: flutter
  http: ^1.2.1

```

### Example: Updating a Post

This example demonstrates updating a post using the JSON placeholder API:

```

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: UpdatePostExample(),
    );
  }
}

```

```

}

class UpdatePostExample extends StatefulWidget {
  @override
  _UpdatePostExampleState createState() => _UpdatePostExampleState();
}

class _UpdatePostExampleState extends State<UpdatePostExample> {
  Future<void> updatePost(int postId, String title, String body) async {
    final response = await http.put(
      Uri.parse('https://jsonplaceholder.typicode.com/posts/$postId'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: jsonEncode(<String, String>{
        'title': title,
        'body': body,
      })),
    );

    if (response.statusCode == 200) {
      // If the server returns a 200 OK response, then the post was
      // successfully updated.
      final responseBody = jsonDecode(response.body);
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: Text("Post Updated"),
            content: Text("Post ID: ${responseBody['id']} updated
successfully."),
            actions: [
              MaterialButton(
                child: Text("OK"),
                onPressed: () {
                  Navigator.of(context).pop();
                },
              ),
            ],
          );
        },
      );
    } else {
      // If the server did not return a 200 OK response,
      // then throw an exception.
      throw Exception('Failed to update post');
    }
  }
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Put Data Example'),
    ),
    body: Center(
      child: ElevatedButton(
        onPressed: () => updatePost(1, 'Updated Title', 'Updated body
text'),
        child: const Text('Update Post'),
      ),
    ),
  );
}

```

## DELETE DATA FROM REST API

### Introduction

Deleting data from a REST API is a critical function in many applications, allowing users to remove information from external data sources. This tutorial explains how to implement DELETE requests in Flutter using the `http` package.

### Setup

Ensure you have the `http` package in your `pubspec.yaml` file to make HTTP requests:

```

dependencies:
  flutter:
    sdk: flutter
  http: ^1.2.1

```

### Example: Deleting a Post

This example demonstrates how to delete a post from a JSON placeholder API. We'll send a DELETE request and handle the response.

```

import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;

void main() {
  runApp(MyApp());
}

```



```

}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: DeletePostExample(),
    );
  }
}

class DeletePostExample extends StatefulWidget {
  @override
  _DeletePostExampleState createState() => _DeletePostExampleState();
}

class _DeletePostExampleState extends State<DeletePostExample> {
  Future<void> deletePost(int postId) async {
    final response = await
http.delete(Uri.parse('https://jsonplaceholder.typicode.com/posts/$postId'
));

    if (response.statusCode == 200) {
      // If the server returns a 200 OK response, post is successfully
deleted.
      showDialog(
        context: context,
        builder: (BuildContext context) {
          return AlertDialog(
            title: Text("Success"),
            content: Text("Post has been deleted successfully."),
            actions: [
              MaterialButton(
                child: Text("OK"),
                onPressed: () {
                  Navigator.of(context).pop();
                },
              ),
            ],
          );
        },
      );
    } else {
      // If the server did not return a 200 OK response,
      // throw an exception.
      throw Exception('Failed to delete post');
    }
  }
}

```

```

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Delete Data Example'),
    ),
    body: Center(
      child: ElevatedButton(
        onPressed: () => deletePost(1), // Assuming you want to delete
the post with ID 1
        child: const Text('Delete Post'),
      ),
    ),
  );
}

```

## FLUTTER QUIZ APP WITH REST API

### Introduction

In this section, you will learn to build a simple quiz app in Flutter using API.

### Setup

Ensure your `pubspec.yaml` includes dependencies for Flutter and the `http` package for API requests:

```

dependencies:
  flutter:
    sdk: flutter
  http: ^1.2.1

```

### Main Screen

The main screen displays a play button that navigates to the quiz player screen.

```

import 'package:flutter/material.dart';

void main() => runApp(const QuizApp());

class QuizApp extends StatelessWidget {
  const QuizApp({super.key});

```

```

@override
Widget build(BuildContext context) {
  return const MaterialApp(
    home: MainScreen(),
  );
}

class MainScreen extends StatelessWidget {
  const MainScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Trivia Quiz')),
      body: Center(
        child: ElevatedButton(
          child: const Text('Play'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const
PlayerScreen()),
            );
          },
        ),
      ),
    );
  }
}

```

## Player Screen

The player screen fetches and displays a question, presenting multiple choice answers.

```

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

  @override
  _PlayerScreenState createState() => _PlayerScreenState();
}

class _PlayerScreenState extends State<PlayerScreen> {
  int currentQuestionIndex = 0;
  int score = 0;
  List<Question> questions = [];
}

```

```

Future<void> fetchQuestions() async {
    final response = await
http.get(Uri.parse('https://opentdb.com/api.php?amount=2&category=18&diffi
culty=easy&type=multiple'));

    if (response.statusCode == 200) {
        final data = json.decode(response.body);
        setState(() {
            questions = List<Question>.from(data['results'].map((question) =>
Question.fromJson(question)));
        });
    } else {
        throw Exception('Failed to load questions');
    }
}

@override
void initState() {
    super.initState();
    fetchQuestions();
}

void checkAnswer(String selectedAnswer) {
    if (questions[currentQuestionIndex].correctAnswer == selectedAnswer) {
        score++;
    }

    if (currentQuestionIndex < questions.length - 1) {
        setState(() {
            currentQuestionIndex++;
        });
    } else {
        Navigator.push(context, MaterialPageRoute(builder: (context) =>
ScoreScreen(score: score)));
    }
}

@override
Widget build(BuildContext context) {
    if (questions.isEmpty) {
        return Scaffold(
            appBar: AppBar(title: const Text('Question')),
            body: const Center(child: CircularProgressIndicator()),
        );
    }

    final question = questions[currentQuestionIndex];

```

```

return Scaffold(
  appBar: AppBar(title: const Text('Question')),
  body: ListView(
    children: <Widget>[
      ListTile(title: Text(question.question)),
      ...question.answers.map((answer) => ListTile(
        title: Text(answer),
        onTap: () => checkAnswer(answer),
      )).toList(),
    ],
  ),
);
}
}

```

## Question Model

Create a model to represent a question and its answers.

```

class Question {
  final String question;
  final List<String> answers;
  final String correctAnswer;

  Question({required this.question, required this.answers, required
this.correctAnswer});

  factory Question.fromJson(Map<String, dynamic> json) {
    var incorrectAnswers = List<String>.from(json['incorrect_answers']);
    var correctAnswer = json['correct_answer'];
    incorrectAnswers.add(correctAnswer);
    incorrectAnswers.shuffle();
    return Question(
      question: json['question'],
      answers: incorrectAnswers,
      correctAnswer: correctAnswer,
    );
  }
}

```

## Score Screen

After answering questions, navigate to the results screen to display the score.

```

import 'package:flutter/material.dart';

```

```

class ScoreScreen extends StatelessWidget {
  final int score;

  ScoreScreen({super.key, required this.score});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Score')),
      body: Center(
        child: Text('Your score is: $score', style: const
TextStyle(fontSize: 24)),
      ),
    );
  }
}

```

## QUESTIONS FOR PRACTICE 10

### Flutter Practice Questions

This section includes practice questions designed to enhance your understanding and skills in working with REST APIs in Flutter.

1. Create a application that fetches data from <https://jsonplaceholder.typicode.com/users> and displays the list of users.
2. Create a form where users can input their name and message. Post this data to a <https://jsonplaceholder.typicode.com/posts>. Display a confirmation message on successful submission.
3. Create a Weather app in Flutter that fetches weather data from a weather API (e.g., OpenWeatherMap) and displays the current weather information for a specific location.
4. Write a function in Flutter that converts the JSON data fetched from the REST API into a list of Dart objects.

## 11) \_State Management in Flutter

This section will help you understand the core concepts of state management with real-world examples. Here you will learn the following topics:

- [Introduction to State Management](#),
- [Provider in Flutter](#), and
- [Building a Quiz App with Provider](#).

### Practice Questions

After completing this section, [practice these questions](#) to test your understanding and improve your skills in managing state within Flutter applications.

## INTRODUCTION TO STATE MANAGEMENT

### Introduction

State means the data or the properties of an app that can change over time. State management is the process of managing and updating the state in your app.

**Note:** Proper state management helps you build scalable and maintainable Flutter apps.

### Why State Management?

You should use state management in your Flutter apps to:

- Keep UI and business logic separate.
- Keep your app's UI in sync with its data.
- Update the UI in response to user interactions or other events.
- Manage app-wide state and data.

### State Management Approaches in Flutter

Flutter offers multiple ways to manage state, including:

- **Local State Management:** Using **StatefulWidget** to manage state within a widget.
- **Global State Management:** Managing app-wide state using solutions like Provider, Riverpod, Bloc, and more.

### Popular State Management Packages

Here are some popular state management packages for Flutter:

- **Provider:** A simple and effective way to manage app-wide state.
- **Riverpod:** A provider package that offers a more powerful and flexible way to manage state.
- **Bloc:** A predictable state management library that helps you manage app state and UI separately.
- **GetX:** A lightweight and fast state management solution for Flutter.

### Conclusion

This is all about State Management in Flutter. In the next section, you will learn **Provider**, and build quiz app using it.

## PROVIDER IN FLUTTER

### Provider in Flutter

The Provider package is a recommended way to manage state in Flutter applications. It simplifies data flow in your app and makes it more manageable and scalable. Provider allows for efficient and scalable state management, making it easier to share state across different parts of your Flutter application.

### Why Use Provider?

- **Simplicity:** Provider simplifies state management in Flutter, making it more readable and maintainable.
- **Scalability:** It supports scalable architecture, making it suitable for small to large applications.
- **Performance:** Provider is optimized for performance, ensuring minimal rebuilds.

### Example 1: Counter App Using Provider

In this example below, you will learn to create a simple counter app using Provider.

#### *Step 1: Setup Provider*

First, add the Provider package to your `pubspec.yaml` file.

```
dependencies:  
  flutter:  
    sdk: 'flutter'  
  provider: '^5.0.0'
```

or

```
flutter pub add provider
```

#### *Step 2: Create a Counter Model*

Define a model class for your counter logic.

```
import 'package:flutter/foundation.dart';  
  
class Counter extends ChangeNotifier {  
  int _count = 0;  
  
  int get count => _count;  
  
  void increment() {  
    _count++;  
  }  
}
```



```
    notifyListeners();  
  }  
}
```

### *Step 3: Use Provider in Your App*

Wrap your app or widget with a **ChangeNotifierProvider** to provide the counter model.

```
import 'package:flutter/material.dart';  
import 'package:provider/provider.dart';  
  
void main() => runApp(const MyApp());  
  
class MyApp extends StatelessWidget {  
  const MyApp({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    return ChangeNotifierProvider(  
      create: (context) => Counter(),  
      child: MaterialApp(  
        home: CounterPage(),  
      ),  
    );  
  }  
}  
  
class CounterPage extends StatelessWidget {  
  const CounterPage({Key? key}) : super(key: key);  
  
  @override  
  Widget build(BuildContext context) {  
    final counter = Provider.of<Counter>(context);  
  
    return Scaffold(  
      appBar: AppBar(  
        title: const Text('Provider Counter App'),  
      ),  
      body: Center(  
        child: Text('Counter: ${counter.count}'),  
      ),  
      floatingActionButton: FloatingActionButton(  
        onPressed: counter.increment,  
        tooltip: 'Increment',  
        child: const Icon(Icons.add),  
      ),  
    );  
  }  
}
```

```
}
```

## Challenge

Add a decrement button to the counter app and implement the logic to decrement the counter value.

## Example 2: Todo App Using Provider

In this example, you will learn to create a simple todo app using Provider.

### *Step 1: Create a Todo Model*

Define a model class for your todo logic.

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

class Todo extends ChangeNotifier {
  List<String> _todos = [];

  List<String> get todos => _todos;

  void add(String todo) {
    _todos.add(todo);
    notifyListeners();
  }

  void remove(String todo) {
    _todos.remove(todo);
    notifyListeners();
  }
}
```

### *Step 2: Use Provider in Your App*

Wrap your app or widget with a **ChangeNotifierProvider** to provide the todo model.

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

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

```

return ChangeNotifierProvider(
  create: (context) => Todo(),
  child: MaterialApp(
    home: TodoPage(),
  ),
);
}
}

class TodoPage extends StatelessWidget {
  const TodoPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final todo = Provider.of<Todo>(context);
    TextEditingController todoController = TextEditingController();

    return Scaffold(
      appBar: AppBar(
        title: const Text('Provider Todo App'),
      ),
      body: Column(
        children: [
          TextField(
            decoration: const InputDecoration(
              labelText: 'Type todo and hit enter',
              contentPadding: EdgeInsets.all(10),
            ),
            controller: todoController,
            onSubmitted: (value) {
              todo.add(value);
              todoController.clear();
            },
          ),
          Expanded(
            child: ListView.builder(
              itemCount: todo.todos.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(todo.todos[index]),
                  trailing: IconButton(
                    icon: const Icon(Icons.delete),
                    onPressed: () {
                      todo.remove(todo.todos[index]);
                    },
                  ),
                );
              },
            ),
          );
        ],
      ),
    );
  }
}

```

```
}  
}  
);  
,  
],  
,  
,  
),
```

## Challenge

Add edit functionality to the todo app. When a user taps on a todo item, it should be editable.

## QUIZ APP USING PROVIDER

### Creating a Quiz App Using Provider in Flutter

Building a quiz app in Flutter with Provider involves managing the quiz state, including questions, answers, and user scores. This tutorial will guide you through creating a simple quiz app using Provider for state management.

#### Step 1: Setup Provider

First, ensure the Provider package is included in your `pubspec.yaml` file:

```
dependencies:  
  flutter:  
    sdk: flutter  
  provider: ^5.0.0
```

#### Step 2: Define the Quiz Model

Create a model to represent your quiz. It should include questions, options, and the logic to check answers and track scores.

```
class Quiz with ChangeNotifier {  
  final List<Map<String, dynamic>> _questions = [  
    {  
      'questionText': 'What is Flutter?',  
      'answers': [  
        {'text': 'A programming language', 'score': 0},  
        {'text': 'A web framework', 'score': 0},  
        {'text': 'A mobile UI framework', 'score': 1},  
        {'text': 'A game', 'score': 0},  
      ],  
    },  
  ],  
}
```

```

{
  'questionText': 'What language is Flutter written in?',
  'answers': [
    {'text': 'Dart', 'score': 1},
    {'text': 'Java', 'score': 0},
    {'text': 'Kotlin', 'score': 0},
    {'text': 'Swift', 'score': 0},
  ],
},
// Add more questions here
];

int _questionIndex = 0;
int _totalScore = 0;

void answerQuestion(int score) {
  _totalScore += score;
  _questionIndex++;
  notifyListeners();
}

int get questionIndex => _questionIndex;
int get totalScore => _totalScore;
List<Map<String, dynamic>> get questions => _questions;

bool get isQuizFinished => _questionIndex >= _questions.length;

void resetQuiz() {
  _questionIndex = 0;
  _totalScore = 0;
  notifyListeners();
}
}

```

### Step 3: Setup Provider in Main

Wrap your app with `ChangeNotifierProvider` to provide the quiz model.

```

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'quiz.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(

```

```

        create: (ctx) => Quiz(),
        child: MaterialApp(
          home: QuizPage(),
        ),
      );
    }
  }
}

```

## Step 4: Build the Quiz UI

Create a widget to display the quiz question and options. Use `Provider.of` to access and display quiz data, and to handle answer selection.

```

class QuizPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    var quiz = Provider.of<Quiz>(context);
    return Scaffold(
      appBar: AppBar(title: Text('Quiz App')),
      body: quiz.isQuizFinished
        ? Center(
            child: Text('Your score: ${quiz.totalScore}'),
          )
        : Column(
            children: <Widget>[
              Text(quiz.questions[quiz.questionIndex]['questionText']),
              ...(quiz.questions[quiz.questionIndex]['answers'] as
List<Map<String, dynamic>>)
                .map((answer) => ElevatedButton(
                  onPressed: () =>
quiz.answerQuestion(answer['score']),
                  child: Text(answer['text']),
                ))
                .toList(),
            ],
          ),
      floatingActionButton: FloatingActionButton(
        onPressed: quiz.resetQuiz,
        child: Icon(Icons.refresh),
      ),
    );
  }
}

```

## QUESTIONS FOR PRACTICE 11

### State Management Practice Questions

1. Explain the concept of state management in Flutter and why it's important for app development.
2. What is the Provider package in Flutter? Describe how it works for managing state.
3. Create a simple todo list application in Flutter using Provider where users can add, remove, and mark tasks as completed.
4. Discuss the advantages of using Provider over global variables or inherited widgets for state management in Flutter apps.
5. Implement a theme switching feature in a Flutter app using Provider. Users should be able to switch between light and dark themes.
6. Describe the difference between **ChangeNotifierProvider** and **Consumer** widgets in the context of the Provider package. Give an example of how each is used.

## 12) \_Notification in Flutter

Notifications in Flutter are messages or alerts that provides as visual or audio signals to notify users of significant events or updates inside an app. They improve the user experience of the app by attracting users with timely information, such as messages, reminders, or system related events. This section includes:

- [Display Notification in Flutter](#),
- [Display Local Notification in Flutter](#),
- [Display Push Notification in Flutter](#).

### Practice Questions

After completing this section, [attempt the practice questions](#) to test and solidify your knowledge on Notification in Flutter.

## DISPLAY NOTIFICATION IN FLUTTER

### Notification in Flutter

A important part of app development, regardless of the platform, is the user's ability to receive notifications. This article dives deep into the world of notifications in Flutter.

**Note:** Grasping the fundamentals of notifications can significantly amplify user engagement and satisfaction with your Flutter applications.

### Why Notifications Is Important?

Through rapidly alerting users to changes, messages, or events improving user experience, holding attention and promoting interaction, notifications play a crucial role

in app engagement. In the end, they keep users informed and connected with relevant information. Notification are useful because of the following reasons:

**User Engagement:** Timely notifications can keep the user engaged by providing updates, reminders, or news, making them revisit the application more frequently.

**Real-time Updates:** Notifications are crucial for apps that require real-time updates, such as news or chat apps.

**Improved User Experience:** They can provide seamless experiences by informing users about completed tasks, new features, or essential actions needed.

## Different Notification Options in Flutter

Flutter offers various packages and plugins that facilitate notification management, from local notifications to push notifications:

**Local Notifications:** Using plugins like `flutter_local_notifications`, developers can schedule and display notifications locally without the need for an external server. You can learn more about local notifications in Flutter [here](#).

**Firestore Cloud Messaging (FCM):** A cloud solution for messages on iOS, Android, and web applications. The `firebase_messaging` plugin in Flutter allows integrating push notifications in your applications.

**OneSignal:** A popular third-party service for push notifications. Flutter developers can utilize the `onesignal_flutter` package to integrate OneSignal into their apps.

## Local Vs Push Notifications

While both serve the primary purpose of notifying users, they are distinct:

**Local Notifications:** These alerts are internal to the app and are not dependent on an internet connection. For things like daily reminders or alarms, they are ideal.

**Push Notifications:** These need an internet connection and are transmitted to the user's smartphone through the app. They originate from an external server and are perfect for marketing messages, news alerts, and real-time updates.

## DISPLAY LOCAL NOTIFICATION IN FLUTTER

### Display Local Notification in Flutter

The `flutter_local_notifications` package can be used with Flutter to show local notifications. You may use this package to schedule and show notifications in your Flutter App.



## Step 1: Create Flutter Project

Let's create a flutter project called **local\_notification** by flutter create command. Follow the below steps to create a flutter project.

```
flutter create local_notification
```

## Step 2: Add flutter\_local\_notifications Package

To add the **flutter\_local\_notifications** package to your Flutter project, open the pubspec.yaml file and add the following code:

```
dependencies:  
  flutter_local_notifications: ^15.1.1
```

## Step 3: Create File

Without increasing the complexity of the app, let's create 3 files. `main.dart`: In this file we will do the initiation `home_screen.dart`: This file will be used as UI to implement and show notifications in the app `notification_service.dart`: This will maintain the core logic of the notification.

Here is code for `main.dart` file:

```
import 'package:flutter/material.dart';  
import  
'package:flutter_local_notifications/flutter_local_notifications.dart';  
import 'package:local_noti/home_screen.dart';  
import 'package:local_noti/service/notification_service.dart';  
  
FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin =  
  FlutterLocalNotificationsPlugin();  
void main() async {  
  WidgetsFlutterBinding.ensureInitialized();  
  await NotificationService.init();  
  runApp(const MainApp());  
}  
  
class MainApp extends StatelessWidget {  
  const MainApp({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return const MaterialApp(  
      home: HomeScreen(),  
    );  
  }  
}
```

```
}  
}
```

Here is code for `home_screen.dart` file:

```
import 'package:flutter/material.dart';  
import 'notification_service.dart';  
  
class HomeScreen extends StatelessWidget {  
  const HomeScreen({super.key});  
  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      body: Center(  
        child: InkWell(  
          onTap: () => NotificationService.showNotification(  
            title: "This is the title of the notification",  
            body: "This is the body of the notification",  
          ),  
          child: const Text('Send Notification')),  
        ),  
      ),  
    );  
  }  
}
```

Here is code for `notification_service.dart` file:

```
/// Importing this for showing log  
import 'dart:developer';  
  
/// Dependency import for local notifications  
import  
'package:flutter_local_notifications/flutter_local_notifications.dart';  
  
import 'main.dart';  
  
/// Remember the global variables I have created in main.dart  
/// this import allows us to use that variable  
  
/// Notification Service class handles everything regarding notifications  
class NotificationService {  
  /// the method is static, for making it easier to use. You can follow  
  the object  
  /// instance method as well  
  
  static Future<void> init() async {
```

```

    /// This is where start the initialization

// ***** Andriod Initialization *****

    /// '@mipmap/ic_launcher' is the icon for the notification, which is
    deafult in flutter
    /// if you want to use your custom icon make sure to provide the path
    to that icon.

    AndroidInitializationSettings androidInitializationSettings =
        const AndroidInitializationSettings('@mipmap/ic_launcher');

// ***** IOS Initialization *****

    DarwinInitializationSettings darwinInitializationSettings =
        const DarwinInitializationSettings(

// this will ask permissions to display alert and others are clean with
the name
        requestAlertPermission: true,
        requestBadgePermission: true,
        defaultPresentSound: true);

// ***** Combining Initializations *****

    InitializationSettings initializationSettings =
InitializationSettings(
        iOS: darwinInitializationSettings,
        android: androidInitializationSettings);

    /// now using the global instance of FlutterLocalNotificationsPlugin()
    /// Let's Initialize the notification
    bool? init = await flutterLocalNotificationsPlugin
        .initialize(initializationSettings);
    log('Noti $init');
}

/*

Notifications have few things : id, title, body and payload,
in this method the id is optional, however you're recommended to
provide id for different notifications.

*/
static showNotification({
    int? id = 0,
    required String title,
    required String body,

```

```
    var payload,  
  }) async {  
/*
```

In notifications, there are few things that we keep in mind. The sound, how important the notification is and what is the priority of the notification, payload ( this basically means what data does your notifications hold ) of the notification.

We need channel id and channel name in the notification, here "flutter-tut" is channel id and "flutter tutorial" is channel name.

You can give whatever name you wish to give.

```
*/
```

```
// ***** Android Configuration *****
```

```
    AndroidNotificationDetails androidNotificationDetails =  
      const AndroidNotificationDetails(  
        'flutter-tut', 'flutter tutorial',  
        playSound: true,
```

```
// Higher the Priority and Importance,  
// will result in high visibility of notification
```

```
        priority: Priority.max,  
        importance: Importance.max,  
      );
```

```
// ***** iOS Configuration : with the necessary requirements *****
```

```
    DarwinNotificationDetails darwinNotificationDetails =  
      const DarwinNotificationDetails(  
        presentAlert: true,  
        presentSound: true,  
        presentBanner: true,  
        presentBadge: true,  
      );
```

```
    /// let's combine both details.
```

```
    NotificationDetails noti = NotificationDetails(  
      iOS: darwinNotificationDetails, android:  
      androidNotificationDetails);
```

```

    /*
    This line will show the notification, after all the configuration is done
    Passing the payload is totally option. However if your want to navigate to
    a
    particular screen on click of the notification of perform certain action.
    Payload
    become crutial.
    */

    await flutterLocalNotificationsPlugin.show(0, title, body, noti,
        payload: payload);
  }
}

```

## Step 4: Run the App

Now, run the app and click on the **Send Notification** button to display the local notification.

## DISPLAY PUSH NOTIFICATION IN FLUTTER

### Display Push Notification in Flutter

In this section, you will learn how to display push notification in flutter using **firebase\_messaging** package. This package allows you to display push notification in flutter for android, ios, web, and desktop.

### Step 1: Create Flutter Project

Let's create a flutter project called **push\_notification** by flutter create command. Follow the below steps to create a flutter project.

```
flutter create push_notification
```

**Note:** Open the project in your favorite editor and make sure you are in the root directory of the project in terminal.

### Step 2: Create Firebase Project

Go to [Firebase Console](#) and create a new project. If you already have a project, then you can use that project.

### Step 3: Add Firebase to Flutter App

Click on Flutter icon to add firebase to flutter app. You also need to install **firebase cli** to add firebase to flutter app. You can install firebase cli using the following command. Also follow the instructions from firebase.

**Note:** If you don't have npm installed, then you can install it from [here](#).

```
npm install -g firebase-tools
```

Command to add firebase to flutter app.

```
firebase login
firebase init
dart pub global activate flutterfire_cli
flutterfire configure --project=<projectId>
```

### Step 4: Add Packages

Now it's time to add **firebase\_core** and **firebase\_messaging** package in your pubspec.yaml file. You can add these packages in your pubspec.yaml file as shown below.

```
dependencies:
  flutter:
    sdk: flutter
  firebase_core: ^2.16.0
  firebase_messaging: ^14.6.8
```

### Step 5: Create File

In the lib folder, create a new file called **firebase\_api.dart** and add the following code.

```
import 'package:firebase_messaging/firebase_messaging.dart';

class FirebaseApi{
  final _firebaseMessaging = FirebaseMessaging.instance;

  Future<void> initNotification() async {
    await _firebaseMessaging.requestPermission(
      alert: true,
      badge: true,
      criticalAlert: true,
      sound: true,
    );
  }
}
```

```

    final token = await _firebaseMessaging.getToken();
    print('Token: $token');
  }

  void subscribeToTopic(String topic) async {
    await _firebaseMessaging.subscribeToTopic(topic);
  }
}

```

## Change Main File

In the main.dart file, add the following code.

```

import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:firebase_messaging/firebase_messaging.dart';
import 'firebase_api.dart';
import 'firebase_options.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  await FirebaseApi().initNotification();
  FirebaseApi().subscribeToTopic('all');

  FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);
  runApp(MyApp());
}

Future<void> _firebaseMessagingBackgroundHandler(RemoteMessage message)
async {
  print('Handling a background message ${message.messageId}');
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: NotificationHandler(),
    );
  }
}

class NotificationHandler extends StatefulWidget {

```

```

@override
_NotificationHandlerState createState() => _NotificationHandlerState();
}

class _NotificationHandlerState extends State<NotificationHandler> {
  final FirebaseMessaging _firebaseMessaging = FirebaseMessaging.instance;

  @override
  void initState() {
    super.initState();

    _firebaseMessaging.getToken().then((String? token) {
      assert(token != null);
      print("Push Messaging token: $token");
    });

    FirebaseMessaging.onMessage.listen((RemoteMessage message) {
      print('Got a message whilst in the foreground!');
      print('Message data: ${message.data}');
      if (message.notification != null) {
        print('Message also contained a notification:
${message.notification}');
        showDialog(
          context: context,
          builder: (context) => AlertDialog(
            content: ListTile(
              title: Text(message.notification!.title!),
              subtitle: Text(message.notification!.body!),
            ),
            actions: <Widget>[
              MaterialButton(
                child: Text('Ok'),
                onPressed: () => Navigator.of(context).pop(),
              ),
            ],
          ),
        );
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Notification Handler'),
      ),
      body: Center(

```



```
        child: Text('Waiting for message...'),  
      ),  
    );  
  }  
}
```

## Step 6: Run Flutter Project

Run your flutter project using the following command.

```
flutter run
```

## Step 7: Send Notification

Open your firebase console and go to **Cloud Messaging**. Click on **New Notification** and send a notification. You will see the notification on your device.

## QUESTIONS FOR PRACTICE 12

### Flutter Notification Practice Questions

- Develop a Flutter app that triggers local notifications at a specified time. Include options for scheduling and cancelling notifications.
- Build a Flutter application that receives push notifications using Firebase Cloud Messaging. Test with different message types.
- Create an app that allows users to set reminders for various tasks. Use local notifications to alert users when a task is due.
- Build a news application that sends push notifications to users when new articles are published or when there is breaking news.
- Build an app that allows users to select custom sounds for different types of notifications.
- Create an app where notifications come with actions (like 'Reply', 'Mark as Read', etc.), and demonstrate handling these actions within the Flutter app.

## 13) \_Flutter How To?

This part will teach you how to use flutter concepts to make life easier and more enjoyable.

This will include following topics:

- [Update All Packages in Flutter](#),
- [Add Internet Permission in Flutter](#),
- [Use Google Maps in Flutter](#),
- [Get Jobs in Flutter](#),
- [Convert Website to Flutter App](#).

## Practice & Implement

After going through these guides, implement the steps in your projects and practice to perfect your skills in Flutter.

## UPDATE ALL PACKAGES IN FLUTTER

### Update All Packages in Flutter

We shall discover how to update every package in Flutter by looking at this example. To update all of the flutter packages, we will use the flutter pub upgrade command.

### See Available Upgrades

If you want to see the available upgrades, you can use the flutter pub outdated command. It will show you the available upgrades for your flutter project.

```
flutter pub outdated
```

### Update All Packages

To update all packages in flutter, you can use the flutter pub upgrade command. It will update all packages in your flutter project.

```
flutter pub upgrade
```

### Update a Specific Package

If you want to update a specific package, you can use the flutter pub upgrade command with the package name. It will update the specific package in your flutter project.

```
flutter pub upgrade <package_name>
```

### Example: Update Specific Package

Let's suppose you want to update the firebase\_auth package. To update the firebase\_auth package, you can use the following command.

```
flutter pub upgrade firebase_auth
```

### Update a Specific Package to a Specific Version

If you want to update a specific package to a specific version, you can use the flutter pub upgrade command with the package name and version. It will update the specific package to a specific version in your flutter project.

```
flutter pub upgrade <package_name> <version>
```

## Example: Update Specific Package to a Specific Version

Let's suppose you want to update the `firebase_auth` package to version 1.0.0. To update the `firebase_auth` package to version 1.0.0, you can use the following command.

```
flutter pub upgrade firebase_auth 1.0.0
```

## ADD INTERNET PERMISSION IN FLUTTER

### Internet Permission in Flutter

In this example, you will learn how to add internet permission in flutter. And then you will learn how to add internet permission in flutter for android and ios.

### Add Internet Permission in Android

To add internet permission in android, you need to add the following line in the `android/app/src/main/AndroidManifest.xml` file.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

File Should Look Like This:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
<uses-permission android:name="android.permission.INTERNET"/>
  <application
    android:label="Learn Computer Basic"
    android:icon="@mipmap/ic_launcher">
```

### Add Internet Permission in iOS

To add internet permission in iOS, you need to add the following line in the `ios/Runner/Info.plist` file.

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSAllowsArbitraryLoads</key>
  <true/>
</dict>
```

## CONVERT WEBSITE TO FLUTTER APP

### Convert Website to Flutter App

This part will teach you how to turn a website into a Flutter application. You can convert any website to flutter app using the **webview\_flutter** package. This package is used to convert any website to flutter app.

## Step 1: Create a Flutter Project

We will convert google.com website to flutter app. So, create a flutter project using the below command.

```
flutter create website_to_flutter_app
```

## Step 2: Add Package

Add **webview\_flutter** package to your pubspec.yaml file.

```
dependencies:  
  flutter:  
    sdk: flutter  
  webview_flutter: ^4.4.1
```

## Step 3: Add Code

Add the below code to your main.dart file.

```
import 'package:flutter/material.dart';  
import 'package:webview_flutter/webview_flutter.dart';  
  
void main() {  
  runApp(MyApp());  
}  
  
class MyApp extends StatelessWidget {  
  // Controller  
  final WebViewController controller = WebViewController()  
    ..loadRequest(Uri.parse('https://www.google.com'));  
  @override  
  Widget build(BuildContext context) {  
    return MaterialApp(  
      home: Scaffold(  
        appBar: AppBar(  
          title: const Text('Google Website'),  
        ),  
        body: WebViewWidget(controller: controller),  
      );  
    );  
  }  
}
```

## Step 4: Run App

Run your app using the below command.

```
flutter run
```

## Convert HTML to Flutter App

You can also convert website content to dart variable and then use it in your flutter app. For example, you can convert the below html code to dart variable and then use it in your flutter app.

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

String htmlContent = '''
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>

</body>
</html>
''';
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // Controller
  final WebViewController controller = WebViewController()
    ..loadHtmlString(htmlContent);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Google Website'),
        ),
        body: WebViewWidget(controller: controller),
      );
    );
  }
}
```

## Permissions

You need to add internet permission in your AndroidManifest.xml file. For iOS, you don't need to add any permission.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

## Conclusion

Congratulations, you have successfully converted website to flutter app.

## HOW TO GET JOB IN FLUTTER

### How to Get Job in Flutter

Getting a job as a Flutter developer involves a combination of technical proficiency, networking, and job search strategies. Here's a step-by-step guide to help you land a job in Flutter development:

#### 1. Master Flutter

Before you start applying for jobs, you need to master Flutter. You can learn Dart Programming Language and Flutter from our [Dart Tutorial](#) and [Flutter Tutorial](#) respectively. Make sure you have a solid understanding of the following concepts:

- Dart Programming Language Skills
- Object Oriented Programming concepts
- Flutter's widgets and the widget tree
- State management solutions like Provider, Riverpod, BLoC, etc.
- CI/CD tools like GitHub Actions, Codemagic, etc.
- Building small apps and gradually move to more complex projects.

#### 2. Build a Portfolio

A portfolio is a collection of your best work. It can be a website, a GitHub repository, or a mobile app. It should showcase your skills and experience in Flutter development. You can build a portfolio by working on open-source projects, contributing to existing projects, or creating your own apps.

- Create a portfolio of Flutter projects. These could range from simple apps to more intricate ones.
- Use platforms like GitHub to showcase your code. It's essential to show potential employers that you can write clean and maintainable code.
- Deploy a few of your apps to the App Store or Google Play to demonstrate your ability to complete the full app development cycle.

#### 3. Create a Resume

A resume is a written description of your education, professional background, and skills. When recruiters get your application, they see it right away. You can get an interview and differentiate yourself from the competition with a strong CV. Here are some tips for creating a resume that will get you noticed:

- Highlight your Flutter and Dart experience prominently.
- Mention other relevant skills like familiarity with CI/CD, integration with Firebase, web development, etc.
- Include links to your GitHub projects and deployed apps.
- Include objective statements that describe what you're looking for in a job.

**Note:** Don't forget to create a LinkedIn profile. You will find it quite beneficial to network with other Flutter developers, possible employers, and other professionals in this sector.

## 4. Networking

Networking is a great way to find job opportunities and build relationships with other Flutter developers. Here are some tips for networking effectively:

- Join local Flutter meetups and conferences.
- Attend Flutter events like Flutter Engage and Flutter Live.
- Connect with other Flutter developers on LinkedIn and Twitter.

## 5. Communication

Communication is an essential skill for any developer. It's important to be able to communicate your ideas clearly and concisely. Here are some tips for improving your communication skills:

- Practice writing emails and blog posts.
- Join a Toastmasters club or take public speaking classes.
- Read books on communication and presentation skills.

## 6. Job Search

Once you've mastered Flutter and built a portfolio, it's time to start looking for jobs. Here are some top flutter job sites where you can apply for a flutter job.

- [Indeed](#)
- [Glassdoor](#)
- [LinkedIn](#)
- [Flutter Jobs](#)

**Note:** Customize your cover letter for each position. Address the specific requirements mentioned in the job description. Also Prepare for technical interviews. This might include coding challenges, whiteboarding sessions, or discussions about Flutter-specific topics.

## 7. Stay Updated

Flutter is a rapidly evolving technology. It's important to stay up-to-date with the latest developments in the Flutter ecosystem.

## 8. Consider Freelancing

If you're finding it challenging to land a full-time role initially, consider taking up freelance Flutter projects. Platforms like Upwork, Freelancer, and Toptal often have Flutter-related gigs. This will help you gain more experience and build a stronger portfolio.

### REMOVE DEBUG BANNER IN FLUTTER

#### Remove Debug Banner in Flutter

To remove the debug banner in flutter, you need to set the **debugShowCheckedModeBanner** property to **false** in the **MaterialApp** widget.

```
MaterialApp(  
  // set this property to false  
  debugShowCheckedModeBanner: false,  
)
```

**Note:** This change is only for visual purposes in the debug mode and does not affect the release build of your app.

#### For CupertinoApp

If you are using the **CupertinoApp** widget, then you need to set the **debugShowCheckedModeBanner** property to **false** in the **CupertinoApp** widget.

```
CupertinoApp(  
  // set this property to false  
  debugShowCheckedModeBanner: false,  
)
```

## 14) \_Flutter Admob Integration

In this section, you will learn how to integrate different ads in your flutter application.

This will include following topics:

- [Show Banner Ad In Flutter](#)



- [Show Interstitial Ad In Flutter](#)

## SHOW BANNER AD IN FLUTTER

### How To Show Banner Ad In Flutter

In this section, you will learn how to show banner ad in flutter. You will learn how to integrate admob banner ad in flutter with these steps:

#### Step 1: Create Flutter Project

First of all, create a flutter project with the name **my\_admob\_app**. To create a flutter project, open the command prompt and type the following command.

```
flutter create my_admob_app
```

#### Step 2: Add Admob Package

Now, open your project in android studio and add the admob package in your project. To add the admob package, open the pubspec.yaml file and add the following code in it.

```
flutter pub add google_mobile_ads
```

#### Step 3: Add Admob App Id

Now, open the android folder and add the admob app id in the AndroidManifest.xml file. To get the admob app id, go to the admob website and create an account. After creating an account, create an app will appear and you have to click it, after that get the app id from there then add the app id in the AndroidManifest.xml file.

#### Add Internet Permission

Open the AndroidManifest.xml file and add the following code inside the tag.

```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
```

#### Open AndroidManifest.xml File And Add Code

Now, open the AndroidManifest.xml file and add the following code inside the tag. You can find the value of the app id in the admob website.

```
<!-- For Admob. change the value to your own Admob App ID -->  
<meta-data  
  android:name="com.google.android.gms.ads.APPLICATION_ID"
```

```
android:value="ca-app-pub-3940256099942544~3347511713"/>
```

## Change Minimum SDK Version

After adding `google_mobile_ads`, you need to make sure that your app is using Android SDK version 19 or higher. Open `my_admob_app\android\app\build.gradle` and inside `defaultConfig`, change the minimum SDK version to 19. Also set `multiDexEnabled` to `true`.

```
defaultConfig {  
    minSdkVersion 19  
    multiDexEnabled true  
}
```

## Create Banner Ad Widget

Now, create a banner ad widget. To create a banner ad widget, let's create a file `ad.dart` inside the `lib` folder and add the following code in it.

```
import 'package:flutter/material.dart';  
import 'package:google_mobile_ads/google_mobile_ads.dart';  
  
// Class to manage ads  
class AdManager {  
    // Replace with your own Ad Unit ID  
    static const String bannerAdUnitId = 'ca-app-pub-3940256099942544/6300978111';  
  
    // Banner ad instance  
    BannerAd? myBanner;  
  
    // Initialize banner ad  
    void initialize() {  
        myBanner = createBannerAd();  
    }  
  
    // Create a BannerAd  
    BannerAd createBannerAd() {  
        return BannerAd(  
            adUnitId: bannerAdUnitId,  
            size: AdSize.banner,  
            request: const AdRequest(),  
            listener: BannerAdListener(  
                // Event handlers for the ad lifecycle  
                onAdLoaded: (Ad ad) => debugPrint('Ad loaded.'),  
                onAdFailedToLoad: (Ad ad, LoadAdError error) =>  
                    debugPrint('Ad failed to load: $error'),  
            ),  
        );  
    }  
}
```

```

        onAdOpened: (Ad ad) => debugPrint('Ad opened.'),
        onAdClosed: (Ad ad) => debugPrint('Ad closed.'),
        onAdImpression: (Ad ad) => debugPrint('Ad impression.'),
      ),
    );
  }
}

```

## Add Banner Ad Widget

Finally, add the banner ad widget in your flutter application. To add the banner ad widget, open the main.dart file and add the following code in it.

```

import 'package:flutter/material.dart';
import 'package:google_mobile_ads/google_mobile_ads.dart';
import 'ad.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  MobileAds.instance.initialize();
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Admob',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key}) : super(key: key);

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  // Instance of AdManager

```

```

final AdManager adManager = AdManager();

// Banner ad instance
late BannerAd myBanner;

@override
void initState() {
  super.initState();
  // Initialize AdManager and load the banner ad
  adManager.initialize();
  myBanner = adManager.myBanner!;
  myBanner.load();
}

@override
void dispose() {
  super.dispose();
  // Dispose of the banner ad when it's no longer needed
  myBanner.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text("Admob Example"),
      centerTitle: true,
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          // Display the banner ad
          SizedBox(
            height: 50,
            child: AdWidget(ad: myBanner),
          ),
        ],
      ),
    ),
  );
}

```

## Run Flutter App

Finally, run your flutter app using the following command.

```
flutter run
```

## Conclusion

In this tutorial, you learned how to integrate admob banner ad in flutter with these step by step tutorials. You have also learned how to show banner ad in flutter. If you like this tutorial, please share it with others.

[SHOW INTERSTITIAL AD IN FLUTTER](#)

## How To Show Interstitial Ad In Flutter?

In this section, you will learn how to show interstitial ad in flutter. You will learn how to integrate admob interstitial ad in flutter with step by step tutorial.

### Step 1: Create Flutter Project

First of all, create a flutter project. Let's create a flutter project with the name **my\_admob\_app**. To create a flutter project, open the command prompt and type the following command.

```
flutter create my_admob_app
```

### Step 2: Add Admob Package

Now, open your project in android studio and add the admob package in your project. To add the admob package, open the pubspec.yaml file and add the following code in it.

```
flutter pub add google_mobile_ads
```

### Step 3: Add Admob App Id

Now, open the android folder and add the admob app id in the AndroidManifest.xml file. To get the admob app id, go to the admob website and create an account. After creating an account, create an app will appear and you have to click it, after that get the app id from there. Now, add the app id in the AndroidManifest.xml file.

### Add Internet Permission

Open the AndroidManifest.xml file and add the following code inside the tag.

```
<uses-permission android:name="android.permission.INTERNET"/>  
<uses-permission android:name="com.google.android.gms.permission.AD_ID"/>
```

### Open AndroidManifest.xml File And Add Code

Now, open the AndroidManifest.xml file and add the following code inside the tag. You can find the value of the app id in the admob website.

```
<!-- For Admob. change the value to your own Admob App ID -->
<meta-data
    android:name="com.google.android.gms.ads.APPLICATION_ID"
    android:value="ca-app-pub-3940256099942544~3347511713"/>
```

## Change Minimum SDK Version

After adding google\_mobile\_ads, you need to make sure that your app is using Android SDK version 19 or higher. Open **my\_admob\_app\android\app\build.gradle** and inside defaultConfig, change the minimum SDK version to 19. Also set multiDexEnabled to true.

```
defaultConfig {
    minSdkVersion 19
    multiDexEnabled true
}
```

## Create Interstitial Ad Widget

Now, create a interstitial ad widget. To create a interstitial ad widget, create a new file named **ad.dart** and add the following code in it.

```
import 'package:google_mobile_ads/google_mobile_ads.dart';

// Class to manage ads
class AdManager {
    // Replace with your own Interstitial Ad Unit ID
    static const String interstitialAdUnitId = 'ca-app-pub-3940256099942544/1033173712';

    // Interstitial ad instance
    InterstitialAd? myInterstitialAd;
    // Flag to keep track of whether interstitial ad is ready
    bool isInterstitialAdReady = false;

    // Initialize interstitial ad
    void initialize() {
        createInterstitialAd();
    }

    // Create an InterstitialAd
    void createInterstitialAd() {
        InterstitialAd.load(
            adUnitId: interstitialAdUnitId,
```

```

request: AdRequest(),
adLoadCallback: InterstitialAdLoadCallback(
  onAdLoaded: (InterstitialAd ad) {
    // Keep a reference to the ad so you can show it later
    myInterstitialAd = ad;
    isInterstitialAdReady = true;

    ad.fullScreenContentCallback = FullScreenContentCallback(
      // Reload the ad when it's dismissed
      onAdDismissedFullScreenContent: (InterstitialAd ad) {
        createInterstitialAd();
      },
    );
  },
  onAdFailedToLoad: (LoadAdError error) {
    print('InterstitialAd failed to load: $error');
  },
),
);
}

// Show the interstitial ad if it's ready
void showInterstitialAd() {
  if (myInterstitialAd == null) {
    print('Warning: attempt to show interstitial before loaded.');
```

```

    return;
  }
  myInterstitialAd!.show();
  myInterstitialAd = null;
  isInterstitialAdReady = false;
}
}

```

## Add Interstitial Ad Widget In Main.dart File

Finally, add the interstitial ad widget in the main.dart file. To add the interstitial ad widget, open the main.dart file and add the following code in it.

```

import 'package:flutter/material.dart';
import 'package:google_mobile_ads/google_mobile_ads.dart';
import 'ad.dart';

void main() {
  WidgetsFlutterBinding.ensureInitialized();
  MobileAds.instance.initialize();
  runApp(const MyApp());
}

```

```

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Admob',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: const MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key}) : super(key: key);

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  // Instance of AdManager
  final AdManager adManager = AdManager();

  @override
  void initState() {
    super.initState();
    // Initialize AdManager and load the interstitial ad
    adManager.initialize();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text("Admob Example"),
        centerTitle: true,
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Button to show the interstitial ad
            ElevatedButton(
              onPressed: () {

```



```

        if (adManager.isInterstitialAdReady) {
            adManager.showInterstitialAd();
        }
    },
    child: const Text('Show Interstitial Ad'),
),
],
),
),
),
);
}
}

```

## Run Flutter App

Finally, run your flutter app using the following command.

```
flutter run
```

## Conclusion

In this tutorial, you learned how to show interstitial ad in flutter and how to integrate admob interstitial ad in flutter with this tutorial. Additionally, you now know how to make a flutter interstitial ad widget. Now, you can show interstitial ad in your flutter app.

## 15) \_Publishing Flutter App

In this section, you will learn how to publish your flutter app on play store and app store. The learning topics are:

- [Publish Flutter App On Play Store](#)
- [Publish Flutter App On Apple Store](#)

## Practice Questions

After completing this section, [attempt the practice questions](#) to test and solidify your knowledge on Flutter Security.

## PUBLISH FLUTTER APP ON APP STORE

### Introduction

This section will help you to learn how to publish app on app store(iOS) step by step.

### Requirements

Before publishing android app to app store make sure you have the following requirements.

- **App Logo (1024x1024px)**
- **Screenshots iPhone("6.7,6.5,5.5"), iPad("12.9 6th Gen, 12.9 2nd Gen")**
- **App Name, Description, Short Description, Subtitle, Category, Sub category, Pricing**
- **App Private Policy**
- **Support URL**
- **Marketing URL**
- **About Us**
- **Flutter App**
- **Apple developer account enrolled in developer program**
- **Mac that supports Xcode**
- **Xcode latest version**

#### *Step 1: Create certificate*

- Create a certificate in app developer then fill the form
- Download and install certificate

#### *Step 2: Get identifier*

- Click the identifiers
- Fill the form i.e., create the bundle id
- Go to Apps and click new app
- Fill the form(name, bundle id)

#### *Step 3: Define the signing key*

- Open the project
- Fill the data form in Sign In
- You have to define the signing key (by signing the developer account in Xcode automatically signing will be happen)

#### *Step 4: Update App Icon*

- Open Android Studio and click ios
- Go to runner and put the image of you app icon in **assets.xcassets**

#### *Step 5: Upload your App*

- Go to Xcode and click the Build then click archive.
- After clicking archive the app will directly come to the TestFlight.
- After that, go to the AppStore and click Add Build then again click the app and click done button.
- If you have any remaining to fill the form you can check and fill it like Description, Support URL, Marketing URL, Contact Information.
- Put the link of App Privacy Policy

- Click to Add for Review

**Note:** After 2, 3 business days your app will be approved after reviewing and it will be publishes

## PUBLISH FLUTTER APP ON PLAY STORE

### How to Publish App On Play Store

This section will help you to learn how to publish app on play store step by step.

#### Requirements

Before publishing android app to google play store make sure you have the following requirements.

- App Logo (512x512px)
- App Banner (1024x500px)
- Screenshots for mobile, tablet, and Chromebook (1920x1024px, 16:9 ratio)
- App signing key (key.jks and key.properties)
- App Name and Description
- Privacy Policy
- About Us
- App Video (optional)
- App Bundle
- [Verified Google Play Console Account](#)

#### Step 1: Generate App Icon

- Generate Icon from [App Icon Generator](#)
- Paste the generated icon in **android/app/src/main/res** folder.

#### Step 2: Create KeyStore File

You can paste the following command in the terminal to generate keystore file.

```
keytool -genkey -v -keystore ~/upload-keystore.jks -keyalg RSA -keysize 2048 -validity 10000 -alias upload
```

- Enter the password and other details.
- You will get **upload-keystore.jks** file in your home directory.

In my case location is

```
C:/Users/iambi/upload-keystore.jks
```

### Step 3: Create Key Properties File

Create a file named **key.properties** in the **android** folder and paste the following code.

```
storePassword=<password-from-previous-step>
keyPassword=<password-from-previous-step>
keyAlias=upload
storeFile=<keystore-file-location>
```

### Step 4: Update Build Gradle File

Now open the **android/app/build.gradle** file and paste the following code.

```
def keystoreProperties = new Properties()
def keystorePropertiesFile = rootProject.file('key.properties')
if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new
FileInputStream(keystorePropertiesFile))
}

android {
    ...
}
```

Also, update the **signingConfigs** and **buildTypes** section.

```
signingConfigs {
    release {
        keyAlias keystoreProperties['keyAlias']
        keyPassword keystoreProperties['keyPassword']
        storeFile keystoreProperties['storeFile'] ?
file(keystoreProperties['storeFile']) : null
        storePassword keystoreProperties['storePassword']
    }
}
buildTypes {
    release {
        signingConfig signingConfigs.release
    }
}
```

### Step 5: Change App Name

Open the **android/app/src/main/AndroidManifest.xml** file and change the **android:label** value.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android">
  <application
    android:label="Your App Name"
    android:icon="@mipmap/ic_launcher">
```

## Step 6: Change App ID

Open the **android/app/build.gradle** file and change the **applicationId** value.

```
defaultConfig {
  applicationId "com.example.app"
  ...
}
```

## Step 7: Build Signed App Bundle on VS Code:

```
flutter build appbundle
```

-Alternatively, you Use Android Studio to build a signed app bundle.

## Step 8: Publish and Release for Google Review

- Fill the necessary details under section of "Set up your app".
- Goto Create and Publish a Release(app bundle ) and Submit the release for Google's review.
- After Google's review, if no issues are found, your app will be ready for publishing.

## 16) \_Quiz Game

Flutter Quiz By Technology Channel

Take our quiz and test your flutter skill. Click on link to get started.

- [Start Basic Flutter Quiz](#)

### 1. What is Flutter?

- a) A programming language
- b) An operating system
- c) Dart framework
- d) A database management system

### 2. Which programming language is used in Flutter for app development?

- a) Swift
- b) Kotlin
- c) Java
- d) Dart

### **3. What is a widget in Flutter?**

- a) A design pattern
- b) A graphical representation of an app's layout or UI
- c) A programming language
- d) A data storage structure

### **4. How is the layout in Flutter defined?**

- a) Using Dart code
- b) Using HTML
- c) Using XML
- d) Using CSS

### **5. Which widget is used to create a container with a fixed size in Flutter?**

- a) Card
- b) Container
- c) Padding
- d) Scaffold

### **6. What does 'hot reload' do in Flutter?**

- a) Reloads the entire app
- b) Refreshes the device
- c) Reloads only the modified parts of the app
- d) Pauses the app execution

### **7. What is the purpose of the 'setState' method in Flutter?**

- a) Initializes the state of a widget
- b) Resets the widget to its default state
- c) Updates the UI based on changes in the widget's state
- d) Removes the state of a widget

### **8. Which widget is used to display text in Flutter?**

- a) Text
- b) TextView
- c) Label
- d) DisplayText

### **9. What is the entry point of a Flutter app?**

- a) main()
- b) runApp()
- c) start()
- d) run()

**10. Which command is used to create a new Flutter project?**

- a) flutter build
- b) flutter init
- c) flutter create
- d) flutter new

**11. What is the purpose of the 'MaterialApp' widget in Flutter?**

- a) It handles user authentication
- b) It displays images in the app.
- c) It manages stateful widgets.
- d) It creates a material design app.

**12. Which widget is used to create a row of child widgets in Flutter?**

- a) Column
- b) Stack
- c) Row
- d) Grid

**13. What is the purpose of the 'Navigator' in Flutter?**

- a) To navigate between different screens or pages in an app
- b) To manage data storage
- c) To control the layout of the app
- d) To handle user inputs

**14. How can you add padding around a widget in Flutter?**

- a) Using the Margin property
- b) Using the Padding widget
- c) Using the Border property
- d) Using the Spacing widget

**15. Which widget is used to draw shapes in Flutter?**

- a) Paint
- b) ShapePainter
- c) CustomPainter
- d) Canvas

**16. What is the purpose of the 'ListView' widget in Flutter?**

- a) To create a grid layout
- b) To display a single item
- c) To manage state in the app
- d) To create a list of items that can be scrolled vertically or horizontally

**17. Which method is called when a State object is removed from the tree in Flutter?**

- a) destroy()
- b) remove()
- c) dispose()
- d) cleanUp()

**18. What is the purpose of the 'GestureDetector' widget in Flutter?**

- a) To detect device gestures like swipes and taps
- b) To manage text input
- c) To handle HTTP requests
- d) To create animations

**19. Which widget is used to add an image in Flutter?**

- a) ImageBox
- b) ImageView
- c) Image
- d) Picture

**20. What is the purpose of the 'async' keyword in Dart?**

- a) It defines a method as a coroutine.
- b) It specifies a method as asynchronous.
- c) It creates a new thread for a method.
- d) It handles exceptions in a method.







