Unit 1: Introduction & Setup

1. What is Flutter, and which programming language does it use?

   A/

   Flutter is an open-source UI software development toolkit created by Google. It is used to build natively compiled applications for mobile (Android & iOS), web, and desktop from a single codebase. The programming language used in Flutter is Dart.

2. List three advantages of using Flutter for app development.

   A/

   - Single codebase for multiple platforms.
   - Fast development with hot reload.
   - Rich widget library.
   - High performance with native compilation.

3. What command do you use to check if Flutter is installed correctly?

   A/

   flutter doctor

4. Explain the purpose of pubspec.yaml in a Flutter project.

   A/

   - Defines project dependencies
   - Manages assets.
   - Manages project metadata.
   - Control versions.

5. Describe the role of the lib/main.dart file in a Flutter project.

   A/

   It is the Main application file: is the entry point of the app which defines the root widget

6. Which of the following is NOT an advantage of Flutter?
   a) Single codebase for multiple platforms
   b) Requires Java for development
   c) Fast development with hot reload
   d) High performance with native compilation

7. What tool is used to create and manage Android emulators?

a) VS Code
b) Dart Analyzer
c) Android Studio AVD Manager
d) Flutter CLI

8. What is the minimum recommended RAM for smooth Flutter performance?
   a) 2GB
   b) 4GB
   c) 8GB
   d) 16GB

9. Write the command to create a new Flutter project named my_app.

   A/

   flutter create my_app

10. Modify the default main.dart file to display "Welcome to Flutter!" instead of "Hello, World!".

    A/

Replace Text('Hello, World!') with Text('Welcome to flutter!') .

11. List 4 key components you need to install in Android Studio as part of preparing an programming environment.

    A/

    - Android SDK Platform, API 35.0.2
    - Android SDK Command-line Tools
    - Android SDK Build-Tools
    - Android SDK Platform-Tools
    - Android Emulator
    - Google USB Driver

12. What is the difference between StatelessWidget and StatefulWidget?

    A/

    - A StatelessWidget is immutable, meaning its properties do not change once it is built. While A StatefulWidget is mutable, meaning it can change dynamically when the app runs.
    - A StatelessWidget is immutable and used for static content, while a StatefulWidget can change dynamically during runtime.

13. Explain the purpose of the setState method in a StatefulWidget.

A/

The setState method is used inside a StatefulWidget to notify the framework/Flutter that the state has changed, triggering a UI rebuild.

14. Name two layout widgets that help arrange UI elements in Flutter.

A/

- Column
- Row
- Stack
- ListView
- GridView

15. What is the main purpose of ListView.builder?

A/

ListView.builder is better for large lists because it builds items on demand means that it improves performance by creating list items only when they are visible, reducing memory usage for large lists.

16. Describe the purpose of Scaffold in a Flutter app

A/

Provides a basic Material Design layout structure.

17. Which of these is a valid way to create a StatelessWidget?

a) class MyWidget extends StatelessWidget {}

b) class MyWidget extends StatefulWidget {}

c) class MyWidget extends Stateless {}

d) class MyWidget extends BuildWidget {}

18. How does Flutter rebuild a StatefulWidget when its state changes?

a) It automatically detects changes

b) The setState method triggers a rebuild

c) The widget is recreated from scratch

d) You must restart the app

19. What is the purpose of the Column widget in Flutter?

a) To display a grid of items

b) To overlay widgets on top of each other

c) To arrange widgets horizontally

d) To arrange widgets vertically

20. Match the following widgets with their descriptions:

Widget	**Description**
ListView(C)	a) Displays items in a horizontal arrangement
Row(A)	b) Arranges widgets on top of each other
Stack(B)	c) Displays a scrollable list of items
GridView(D)	d) Displays items in a grid layout

21. Create a StatelessWidget that displays "Hello, Flutter!".
A/

22. Create a StatefulWidget with a counter that increments when a button is clicked.

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

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

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  var counter = 0;
  @override
  Widget build(BuildContext contet) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text("Counter:$counter\n"),
              ElevatedButton(
                onPressed: () {
                  setState(() {
                    counter++;
```

```
                });
              },
              child: Text("Increment"),
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

23. Examine the following code snippet:

```
class GreetingWidget extends StatelessWidget {
  final String name;

  const GreetingWidget({required this.name, super.key});

  @override
  Widget build(BuildContext context) {
    return Text('Hello, $name!');
  }
}

class HomeScreen extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Passing Data Example")),
      body: Center(
        child: GreetingWidget(name: "Alice"),
      ),
    );
  }
}
A/
//To be answered later
```

24. What is the purpose of `required this.name` in the `GreetingWidget` constructor?

A/

required this.name ensures the name parameter is mandatory.

Complete the following Flutter code to pass a function from a parent widget (`ParentWidget`) to a child widget (`ChildWidget`). The function should display a `Snackbar` when the button in `ChildWidget` is pressed.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: ParentWidget(),
    );
  }
}

class ParentWidget extends StatelessWidget {
  void showMessage(BuildContext context) {
    // TODO: Show a SnackBar with the message "Button Clicked!"
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Passing Functions Example")),
      body: Center(
        child: ChildWidget(
          onPressed: (), // TODO: Pass function here
        ),
      ),
    );
```

```
  }
}

class ChildWidget extends StatelessWidget {


  const ChildWidget({super.key})
```

```
  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: (){},
      child: Text("Click Me"),
    );
  }
}
```

a. Complete the `showMessage` function to display a `Snackbar`.

A/

void showMessage(BuildContext context) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Button Clicked!')),

);

}

b. Pass the function correctly from `ParentWidget` to `ChildWidget`.

A/

ChildWidget(onPressed: () => showMessage(context))

c. Update `ChildWidget` to receive the function passed from `ParentWidget`

A/

```
final VoidCallback onPressed;
   const ChildWidget({super.key, required this.onPressed});
```

d. In `ChildWidget`, replace the anonymous function the in the `ElevatedButton` with the function received in sub-question c above.

A/

ElevatedButton(onPressed: onPressed,)

Unit 3: Theming & Forms

25. What is ThemeData used for in Flutter?

A/

The ThemeData object defines a ColorScheme for the app.

It also defines app-wide visual styles (colors, fonts).

26. Name two form widgets used to accept user input.

A/

TextFormField,ChechBox,RadioBox,TextFifeld

27. What does the validator function do in a TextFormField?

A/

Validator function validate inputs and return the error message

28. What is the purpose of Navigator.pop(context)?
    a) Navigate to a new screen
    b) Close the current screen and return to the previous one
    c) Reset the navigation stack
    d) Refresh the current screen

29. Which of the following are valid form widgets in Flutter? (Select two)
    a) TextField
    b) Image.asset
    c) Checkbox
    d) ElevatedButton

28. Match the following properties with their descriptions:

Property	**Description**
primaryColor(A)	a) Sets the theme's primary color
accentColor(B)	b) Defines the background color of the app
Brightness(C)	c) Determines if the theme is light or dark

29. Create a Flutter form with a TextField for the username and a TextField for the password.
A/

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("FORM"),
        backgroundColor:     Colors.blueGrey),
```

```
            body: Center(
              child: SizedBox(
                width: 300,
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                 TextField(
                       decoration: InputDecoration(
                         border: OutlineInputBorder(),
                         // label: Text("Username"),
                         hintText: "enter your username",
                       ),
                     ),
                     SizedBox(height: 20),
                     TextField(
                       obscureText: true,
                       decoration: InputDecoration(
                         border: OutlineInputBorder(),
                         // label: Text("Username"),
                         hintText: "Enter Password",
                       ),
                     ),
                   ],
                ),
              ),
            ),
            backgroundColor: Colors.cyan,
          ),
        );
      }
    }
```

30. Modify the theme of a Flutter app to use a dark mode.

A/

```
//add this line or modify it if was already exist
Theme: ThemeData(brightness: Brightness.dark,),
```

31. Create an app that displays a local image (assets/images/sample.png) inside a Center widget. Apart from the code, explain another setup you must do first.

    A/

1. Add the Image to Your Project

- Place your image (`sample.png`) inside the `assets/images/` folder. If the folder doesn't exist, create it.

2. Update `pubspec.yaml` to Include the Image

You need to declare the assets in the `pubspec.yaml` file so that Flutter can load them.

```
assets:

  - assets/images/sample.png
```

3. Create the Flutter App (Codes)

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Center(
          child: Image.asset('assets/images/sample.png'),
        ),
      ),
    );
  }
}
```

Before running the app, ensure you have the following setup:

1. **Create the `assets` Folder**: This is where you place your images. The folder should be inside the root directory of your project.
2. **Add the Image in the Right Directory**: Place your image (`sample.png`) inside the `assets/images/` folder as described earlier.
3. **Declare the Image in `pubspec.yaml`**: This step is essential to let Flutter know where your assets are located. Without this, the app won't be able to find or load the image.
4. **Run `flutter pub get`**: After modifying the `pubspec.yaml` file, run `flutter pub get` in the terminal to fetch the assets and dependencies. This ensures that the assets are included in the app.

Unit 4: Navigation & Routes

32. Explain the difference between push and pushReplacement in Flutter navigation

- **For Back Navigation:**

➢ push: Allows the user to go back to the previous screen.
➢ pushReplacement: Does not allow the user to go back to the previous screen (the previous screen is removed from the stack).

- **For Use Case:**

➢ push: When you want to maintain the current screen in the stack and allow the user to go back.
➢ pushReplacement: When you want to replace the current screen, often used in scenarios like logging in or onboarding, where the user shouldn't return to the previous screen.

Generally, use `push` for standard navigation where the back button is needed, and use `pushReplacement` when you want to replace the current screen without the option to return to it.

33. How do you pass data between screens in Flutter?

1. Passing Data Using Constructor (Direct Method)

You can pass data to a new screen by passing it as an argument in the constructor of the destination screen.

Example:

```
// First Screen (Sender)
import 'package:flutter/material.dart';
import 'second_screen.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FirstScreen(),
    );
  }
}

class FirstScreen extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("First Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Passing data to the SecondScreen
            Navigator.push(
              context,
              MaterialPageRoute(
                builder: (context) => SecondScreen(data: "Hello from
First Screen!"),
              ),
            );
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}
// Second Screen (Receiver)
import 'package:flutter/material.dart';

class SecondScreen extends StatelessWidget {
  final String data;

  // Constructor to accept data
  SecondScreen({required this.data});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: Text(data),  // Displaying the passed data
      ),
    );
  }
}
```

Explanation:

- `FirstScreen` passes a string ("Hello from First Screen!") to `SecondScreen` using the constructor.
- The `SecondScreen` constructor accepts this data and displays it.

2. Passing Data Using `Navigator.pushNamed` (Named Routes)

In Flutter, you can also use named routes and pass arguments when navigating. This method is helpful when you have multiple screens and want to keep your code organized.

Example:

```
// First Screen (Sender)
```

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FirstScreen(),
      routes: {
        '/second': (context) => SecondScreen(),
      },
    );
  }
}

class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("First Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Passing data to the SecondScreen using Named Routes
            Navigator.pushNamed(
              context,
              '/second',
              arguments: "Hello from First Screen using Named Route!",
            );
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}
// Second Screen (Receiver)
import 'package:flutter/material.dart';

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Retrieving the passed data using ModalRoute
    final String data = ModalRoute.of(context)!.settings.arguments as
String;

    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: Text(data),  // Displaying the passed data
      ),
    );
  }
```

```
}
```

Explanation:

- In this method, we define the route in the `MaterialApp` widget using `routes`.
- When navigating, we use `Navigator.pushNamed` and pass data using the `arguments` parameter.
- The data is retrieved in the second screen using `ModalRoute.of(context)!.settings.arguments`.

3. Passing Data Using a Global State (Provider, Riverpod, etc.)

If you need to pass data globally between multiple screens, you can use state management solutions like `Provider`, `Riverpod`, or `Bloc`. These solutions allow you to manage and pass data without needing to explicitly pass it through constructors or route arguments.

Example (Using Provider):

```
// First Screen (Sender)
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'second_screen.dart';

void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => DataProvider(),
      child: MyApp(),
    ),
  );
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FirstScreen(),
    );
  }
}

class DataProvider with ChangeNotifier {
  String _data = "";
  String get data => _data;

  void setData(String newData) {
    _data = newData;
    notifyListeners();
  }
}

class FirstScreen extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("First Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Setting data globally using Provider
            Provider.of<DataProvider>(context, listen:
false).setData("Hello from Provider!");
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondScreen()),
            );
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}
// Second Screen (Receiver)
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'main.dart';

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final String data = Provider.of<DataProvider>(context).data;

    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: Text(data),  // Displaying the passed data
      ),
    );
  }
}
```

Explanation:

- `DataProvider` holds the shared data and notifies listeners when the data changes.
- Both screens can access and update this data using the `Provider` package.

4. Passing Data Using Shared Preferences (For Persisted Data)

If you need to persist data (e.g., user preferences or settings) between screens and sessions, you can use `SharedPreferences`.

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

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

```
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: FirstScreen(),
    );
  }
}

class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("First Screen")),
      body: Center(
        child: ElevatedButton(
          onPressed: () async {
            // Save data in SharedPreferences
            final prefs = await SharedPreferences.getInstance();
            prefs.setString('data', 'Hello from SharedPreferences!');
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondScreen()),
            );
          },
          child: Text("Go to Second Screen"),
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return FutureBuilder(
      future: _getData(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done) {
          return Scaffold(
            appBar: AppBar(title: Text("Second Screen")),
            body: Center(
              child: Text(snapshot.data ?? 'No data'),
            ),
          );
        }
        return CircularProgressIndicator();
      },
    );
  }

  Future<String?> _getData() async {
    final prefs = await SharedPreferences.getInstance();
    return prefs.getString('data');
```

```
  }
}
```

Explanation:

- `SharedPreferences` is used to store simple data (like strings) locally.
- Data is stored in `FirstScreen` and retrieved in `SecondScreen` after navigating.

34. What is the benefit of using named routes instead of direct navigation?

1. Better Code Organization:

- **Named routes** help to organize the navigation logic in a central place. All route names and their corresponding screens are defined in the `MaterialApp` or `CupertinoApp` widget, making the routing system more organized and easier to manage.
- It helps avoid scattering route logic across the app, as it centralizes all route definitions in one place.

Example:

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

2. Decoupling Navigation Logic from UI:

- With **named routes**, the navigation logic is separated from the UI code. You don't need to create new `MaterialPageRoute` objects or pass data explicitly in every screen transition. The app uses the predefined route names, which makes the code more abstract and easier to refactor.
- This decoupling makes the app more maintainable as route changes don't require modifications in each screen's navigation logic.

Example:

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

3. Simplifies Navigation with Arguments:

- **Named routes** provide a more convenient way to pass data (arguments) between screens without having to manually create `MaterialPageRoute` objects.
- By using `Navigator.pushNamed`, you can easily pass data and retrieve it in the destination screen using `ModalRoute.of(context).settings.arguments`.

Example:

```
Navigator.pushNamed(
```

```
  context,
  '/profile',
  arguments: 'User Data',
);
```

4. Consistency Across the App:

- When using **named routes**, all screen transitions follow a consistent approach. This reduces the chances of errors, such as forgetting to wrap routes with `MaterialPageRoute`, or using different approaches for navigation in different parts of the app.
- The navigation flow is more predictable, which helps developers to maintain consistency in navigation behavior.

5. Easier Route Management:

- As your app grows, you might have many screens. **Named routes** make it easier to manage and track all the routes in a centralized place (i.e., in the `routes` property of the `MaterialApp` widget).
- You can also define custom route transitions and manage them centrally without manually passing the route logic in every navigation call.

6. Deep Linking and Dynamic Routing:

- **Named routes** support deep linking, allowing the app to handle URLs that open specific screens directly. For example, you could link to a specific screen in your app from an external source (e.g., a web link or notification).
- Additionally, **dynamic route generation** (using `onGenerateRoute`) allows handling more complex navigation logic based on dynamic data like route parameters.

7. Better Navigation Back Stack Management:

- Named routes make it easier to manage the back stack. Since you know the names of all routes, you can control the back navigation flow more effectively.
- You can also pop routes or replace routes with specific names using `Navigator.popAndPushNamed` or `Navigator.pushReplacementNamed`, which helps in controlling the app's back stack in a more organized way.

8. Improved Testing:

- Using named routes makes it easier to write tests for navigation since route names are consistent and defined centrally. You can mock navigation calls and test screens without dealing with manual route creation.

35. What method is used to retrieve arguments passed to a new screen?

To retrieve arguments passed to a new screen in Flutter, you use the `ModalRoute.of(context)` method, which provides access to the current route's settings and its arguments.

Example:

If you're using **named routes** and passing data, you can retrieve the arguments as follows:

Passing Arguments (Sender Screen):

```
Navigator.pushNamed(
  context,
  '/second',
  arguments: 'Hello from the first screen!',
);
```

Retrieving Arguments (Receiver Screen):

```
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Retrieve the arguments passed from the previous screen
    final String data = ModalRoute.of(context)!.settings.arguments as
String;

    return Scaffold(
      appBar: AppBar(title: Text("Second Screen")),
      body: Center(
        child: Text(data),  // Display the passed data
      ),
    );
  }
}
```

Explanation:

- `ModalRoute.of(context)` provides access to the current route.
- `.settings.arguments` retrieves the arguments passed to that route.
- In this example, the argument is a string (`'Hello from the first screen!'`), and it's cast to `String` using `as String`.

36. What does Navigator.pushNamed(context, '/new-page') do?

a) Pushes a new screen onto the navigation stack

b) Replaces the current screen with a new screen

c) Pops the current screen off the navigation stack

d) Resets the navigation stack

37. Write the code to navigate from HomeScreen to DetailScreen using named routes.

A/

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/home', // Set the initial route
      routes: {
        '/home': (context) => HomeScreen(),  // Define the HomeScreen
route
        '/detail': (context) => DetailScreen(),  // Define the
DetailScreen route
      },
    );
  }
}

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Home Screen')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate to DetailScreen using the named route
            Navigator.pushNamed(context, '/detail');
          },
          child: Text('Go to Detail Screen'),
        ),
      ),
    );
  }
}

class DetailScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Detail Screen')),
      body: Center(
        child: Text('Welcome to the Detail Screen!'),
      ),
    );
  }
}
```

38. Implement a Flutter app with two screens: a home screen with a button that navigates to a second screen.

- A **Home Screen** with a button that navigates to the **Second Screen**.

- The **Second Screen** has a button to navigate back to the **Home Screen**.

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

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

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

class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Home Screen')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondScreen()),
            );
          },
          child: Text('Go to Second Screen'),
        ),
      ),
    );
  }
}

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

```
    return Scaffold(
      appBar: AppBar(title: Text('Second Screen')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text('Go Back to Home Screen'),
        ),
      ),
    );
  }
}
```

Unit 5: Local Database & Shared Preferences

39. What package is used for local database storage in Flutter?

    The most commonly used package for local database storage in Flutter is sqflite. It provides a lightweight SQLite database solution for storing structured data.

40. What is the purpose of SharedPreferences in Flutter?

    SharedPreferences is used to store small amounts of data persistently, such as user preferences, app settings, or simple key-value pairs. It is not meant for large datasets or complex relational data, as it stores data in simple key-value pairs using the shared_preferences package.

41. How do you delete an entry from an SQLite database in Flutter?

    You can delete an entry from an SQLite database in Flutter using the delete method of the sqflite package.

42. What function is used to insert data into an SQLite table?

    The insert function is used to insert data into an SQLite table.

43. What type of storage is best suited for saving user preferences in Flutter?
    a) Firebase
    b) SQLite
    c) **SharedPreferences**
    d) HTTP requests

44. Match the following database operations with their functions:

Operation	**Function**
Insert(C)	a) Remove an item from the database
Query(B)	b) Retrieve data from the database

Delete(A)	c) Add new data to the database
Update(D)	d) Modify existing data in the database

45. Write a function to store a username in SharedPreferences.

A/

```dart
import 'package:shared_preferences/shared_preferences.dart';
// Function to save username
Future<void> saveUsername(String username) async {
  final prefs = await SharedPreferences.getInstance();
  await prefs.setString('username', username);
}

// Function to retrieve username
Future<String?> getUsername() async {
  final prefs = await SharedPreferences.getInstance();
  return prefs.getString('username');
}
```

46. Implement a simple Flutter app that saves and retrieves a list of tasks using SQLite.

A/

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Task Manager',
      home: TaskScreen(),
    );
  }
}

class TaskScreen extends StatefulWidget {
  @override
  _TaskScreenState createState() => _TaskScreenState();
}
```

```dart
class _TaskScreenState extends State<TaskScreen> {
  late Database database;
  List<Map<String, dynamic>> tasks = [];
  TextEditingController taskController = TextEditingController();

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

  Future<void> initDatabase() async {
    database = await openDatabase(
      join(await getDatabasesPath(), 'tasks.db'),
      onCreate: (db, version) {
        return db.execute('CREATE TABLE tasks(id INTEGER PRIMARY KEY
AUTOINCREMENT, name TEXT)');
      },
      version: 1,
    );
    fetchTasks();
  }

  Future<void> addTask(String task) async {
    await database.insert('tasks', {'name': task});
    fetchTasks();
  }

  Future<void> fetchTasks() async {
    final List<Map<String, dynamic>> data = await database.query('tasks');
    setState(() {
      tasks = data;
    });
  }

  Future<void> deleteTask(int id) async {
    await database.delete('tasks', where: 'id = ?', whereArgs: [id]);
    fetchTasks();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Task Manager')),
```

```
      body: Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(8.0),
            child: TextField(
              controller: taskController,
              decoration: InputDecoration(labelText: 'Enter Task'),
            ),
          ),
          ElevatedButton(
            onPressed: () {
              if (taskController.text.isNotEmpty) {
                addTask(taskController.text);
                taskController.clear();
              }
            },
            child: Text('Add Task'),
          ),
          Expanded(
            child: ListView.builder(
              itemCount: tasks.length,
              itemBuilder: (context, index) {
                return ListTile(
                  title: Text(tasks[index]['name']),
                  trailing: IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () => deleteTask(tasks[index]['id']),
                  ),
                );
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

Unit 6: Firebase Integration

47. What is Firebase Firestore, and how is it different from Firebase Realtime Database?

Firebase Firestore is a cloud-hosted NoSQL database that stores data in **documents** and **collections**. It provides scalable, flexible data storage with real-time synchronization.

Differences:

Feature	Firestore	Realtime Database
Data Structure	Documents & Collections (NoSQL)	JSON Tree (NoSQL)
Querying	More advanced (complex queries, indexing)	Limited querying
Offline Support	Strong offline support with local persistence	Basic offline caching
Scalability	Designed for larger-scale apps	Less scalable for large data
Performance	Optimized for querying & real-time updates	Faster for simple real-time operations

48. What is required to connect a Flutter app to Firebase?

To connect a Flutter app to Firebase, you need to:

1. Set up Firebase Console:

   Create a new Firebase project at Firebase Console.

   Add an Android app to your project.

2. Configure Firebase in your Flutter project:

   >Install Firebase CLI:

   npm install -g firebase-tools

3. firebase login

   >Run Firebase setup for Flutter:

4. flutterfire configure

   > Add the required Google Services files:

   google-services.json (Android) → /android/app

5. Add Firebase packages (Firestore, Auth, etc.) to pubspec.yaml

49. How can you listen for real-time updates in Firebase Firestore?

    Use snapshots() to listen for real-time updates in Firestore.

50. What command do you run to add Firebase dependencies to a Flutter project?

    flutter pub add firebase_core firebase_firestore

flutter pub get

51. Implement a Flutter app that retrieves and displays a list of shopping items from Firebase Firestore.

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

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

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

class ShoppingListScreen extends StatelessWidget {
  final TextEditingController itemController = TextEditingController();
  final CollectionReference shoppingItems =
      FirebaseFirestore.instance.collection('shopping_items');

  void addItem() {
    if (itemController.text.isNotEmpty) {
      shoppingItems.add({'name': itemController.text});
      itemController.clear();
    }
  }

  void deleteItem(String id) {
    shoppingItems.doc(id).delete();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Shopping List')),
      body: Column(
        children: [
          Padding(
```

```
            padding: EdgeInsets.all(8.0),
            child: Row(
              children: [
                Expanded(
                  child: TextField(
                    controller: itemController,
                    decoration: InputDecoration(labelText: 'Enter item'),
                  ),
                ),
                IconButton(
                  icon: Icon(Icons.add),
                  onPressed: addItem,
                ),
              ],
            ),
          ),
          Expanded(
            child: StreamBuilder(
              stream: shoppingItems.snapshots(),
              builder: (context, AsyncSnapshot<QuerySnapshot> snapshot) {
                if (!snapshot.hasData)
                  return Center(child: CircularProgressIndicator());
                return ListView(
                  children: snapshot.data!.docs.map((doc) {
                    return ListTile(
                      title: Text(doc['name']),
                      trailing: IconButton(
                        icon: Icon(Icons.delete),
                        onPressed: () => deleteItem(doc.id),
                      ),
                    );
                  }).toList(),
                );
              },
            ),
          ),
        ],
      ),
    );
  }
}
```