Experiment No: 6 a) **Date:**

Aim:

To create custom widgets for specific UI elements.

Objective:

To understand how to create custom widgets for specific UI elements.

Code Snippet (main.dart):

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

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

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
return MaterialApp( home:
Scaffold( appBar: AppBar(
 title: Text('Custom Widget Example'),
 ),
 body: Column(
mainAxisAlignment: MainAxisAlignment.center,
 children: <Widget>[ Padding( padding: constEdgeInsets.all(8.0), child:
CustomTextField( hintText: 'Enter your name', onChanged: (value) {
print('Name changed: $value');
 },
 ),
 ),
SizedBox(height: 20),
Padding( padding: constEdgeInsets.all(8.0), child: CustomTextField( hintText:
'Enter Email', onChanged: (value) {
print('Name changed: $value');
 },
 ),
 ),
SizedBox(height: 20),
Padding( padding: constEdgeInsets.all(8.0), child: CustomTextField( hintText:
'Enter Roll Number', onChanged: (value) {
print('Name changed: $value');
 },
 ),
 ),
SizedBox(height: 20), CustomButton( text: 'Press Me', onPressed: () {
print('Button pressed!');
 },
),
 ],
 ),
 ),
```

```
 );
 }
}
class CustomButton extends StatelessWidget {
 final String? text; final VoidCallback? onPressed;
constCustomButton({ Key? key,
 @required this.text,
 @required this.onPressed,
 }) : super(key: key);
 @override
 Widget build(BuildContext context) { return ElevatedButton( onPressed: onPressed,
 child: Text(text!),
 );
 }
}

class CustomTextField extends StatelessWidget{ final String hintText; final
ValueChanged<String>onChanged;
constCustomTextField({ Key? key, required this.hintText, required this.onChanged, }) :
super(key: key);
 @override
 Widget build(BuildContext context) {
 return TextField( onChanged: onChanged, decoration: InputDecoration( hintText: hintText,
 border: OutlineInputBorder(),
 ),
 );
 }
}
```

Output:

Experiment No: 6 b) **Date:**

Aim:

To apply styling using themes and custom styles.

Objective:

To understand how to apply themes and custom styles to a Flutter app.

Code Snippet (main.dart):

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

void main() {
runApp(MyApp());
}
class MyApp extends Stateless Widget {
 @override
 Widget build(BuildContext context){ return
MaterialApp( theme: ThemeData(
 // Define the overall theme of the app primaryColor:
Colors.blue, accentColor: Colors.orange,
fontFamily: 'Roboto', textTheme: TextTheme(
 headline1: TextStyle(fontSize: 24, fontWeight:
FontWeight.bold), bodyText1: TextStyle(fontSize: 16),
 ),
elevatedButtonTheme: ElevatedButtonThemeData(
 style: ElevatedButton.styleFrom( primary: Colors.blue,
textStyle: TextStyle(fontSize: 18),
 padding: EdgeInsets.symmetric(horizontal: 20, vertical: 15), shape:
RoundedRectangleBorder(
borderRadius: BorderRadius.circular(10),
 ),
 ),
 ),
 ),
 home: HomePage()
);
 }
}
class HomePage extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
 return Scaffold( appBar: AppBar(
 title: Text('Styling Example'),
 ),
 body: Center( child: Column(
mainAxisAlignment: MainAxisAlignment.center, children: <Widget>[
Text(
 'Welcome to MyApp',
 style: Theme.of(context).textTheme.headline1,
 ),
SizedBox(height: 20), ElevatedButton( onPressed: () {},
```

```
child: Text('Get Started'),
),
],
),
),
);
}
}
```

Output:

Experiment No: 7 a) **Date:**

Aim:

To design a form with various input fields.

Objective:

To understand how to create and use custom text fields in a Flutter application and handle user input format.

Code Snippet (main.dart):

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

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

classMainApp extends StatelessWidget {
constMainApp({super.key});

  @override
  Widget build(BuildContext context) {
returnMaterialApp(
home: Scaffold(
appBar: AppBar(title: const Text('Form Application')),
body: Column(
children: <Widget>[
Image.asset('images/3d_avatar_21.png', width: 100, height: 100),
          // Custom text fields for user input
constCustomTextField(label: 'First Name'),
constCustomTextField(label: 'Last Name'),
constCustomTextField(label: 'Email', suffixText: '@mlritm.ac.in'),
constCustomTextField(
prefixText: '+91 ',
label: 'Phone Number',
keyboardType: TextInputType.phone,
maxLength: 10,
          ),
const Divider(indent: 8, endIndent: 8), // Divider
constCustomTextField(label: 'Username'),
constCustomTextField(label: 'Password', obscureText: true),
constCustomTextField(label: 'Confirm Password', obscureText: true),
ElevatedButton(
onPressed: () {},
child: const Text('Register'),
          ),
        ],
      ),
    ),
  );
 }
}

// Custom text field widget
classCustomTextField extends StatelessWidget {
final String label;
finalTextInputType? keyboardType;
final bool obscureText;
final String? prefixText, suffixText;
finalint? maxLength;
```

```
constCustomTextField({
super.key,
requiredthis.label,
this.keyboardType,
this.suffixText,
this.prefixText,
this.maxLength,
this.obscureText = false,
  });

  @override
  Widget build(BuildContext context) {
return Padding(
padding: constEdgeInsets.symmetric(vertical: 8, horizontal: 16),
child: TextFormField(
keyboardType: keyboardType,
obscureText: obscureText,
inputFormatters: maxLength != null
          ? [LengthLimitingTextInputFormatter(maxLength)]
          : null,
decoration: InputDecoration(
border: constOutlineInputBorder(),
labelText: label,
suffixText: suffixText,
prefixText: prefixText,
        ),
      ),
    );
  }
}
```

Get the image 3d_avatar from 3d_avatar_21.png and save it as the images/3d_avatar_21.png in your Flutter project.

Open the pubspec.yaml file in your Flutter project and add the following lines after flutter: to include the image asset in your project:

```
flutter:
  # ...
assets:
    - images/3d_avatar_21.png<-- Add this line
```

Save the file.

Output:

Experiment No: 7 b) **Date:**

Aim:

To implement form validation and error handling.

Objective:

To understand the concept of form validation and error handling in Flutter.

Code Snippet (main.dart):

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

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

classMainApp extends StatelessWidget {
constMainApp({super.key});

  @override
  Widget build(BuildContext context) {
returnMaterialApp(
home: Scaffold(
appBar: AppBar(title: const Text('Form Validation')),
body: constSingleChildScrollView(
child: RegisterForm(),
        ),
      ),
    );
  }
}

classRegisterForm extends StatefulWidget {
constRegisterForm({super.key});

  @override
  State<RegisterForm>createState() => _RegisterFormState();
}

class _RegisterFormState extends State<RegisterForm> {
final _formKey = GlobalKey<FormState>();
final _nameController = TextEditingController(),
      _emailController = TextEditingController();

  @override
  Widget build(BuildContext context) {
return Form(
key: _formKey,
child: Column(
children: <Widget>[
Padding(
padding: constEdgeInsets.symmetric(vertical: 8, horizontal: 16),
child: TextFormField(
controller: _nameController,
decoration: constInputDecoration(labelText: 'Name'),
validator: (value) {
if (value == null || value.isEmpty) {
return 'Please enter your name';
                }
if (!RegExp(r'^[a-zA-Z][a-zA-Z\s]*$').hasMatch(value)) {
return 'Name can only contain letters and spaces in between';
```

```
                }
return null;
              },
            ),
          ),
Padding(
padding: constEdgeInsets.symmetric(vertical: 8, horizontal: 16),
child: TextFormField(
controller: _emailController,
decoration: constInputDecoration(labelText: 'Email'),
validator: (value) {
if (value == null || value.isEmpty) {
return 'Please enter your email';
                }
if (!RegExp(
r'^[a-zA-Z0-9](?!.*\+\+)[a-zA-Z0-9._%+-]*[a-zA-Z0-9]@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$')
                    .hasMatch(value)) {
return 'Please enter a valid email address';
                }
return null;
              },
            ),
          ),
constSizedBox(height: 20),
ElevatedButton(
onPressed: () {
if (_formKey.currentState!.validate()) {
ScaffoldMessenger.of(context).showSnackBar(
constSnackBar(content: Text('Data is valid')),
                );
              }
            },
child: const Text('Submit'),
          ),
        ],
      ),
    );
  }
}
```

Output:

Experiment No: 8 a) **Date:**

Aim:

To add animations to UI elements using Flutter's animation framework.

Objective:

To understand how to add animations to UI elements using Flutter's animation framework.

Code Snippet (main.dart):

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

void main() => runApp(const MainApp());

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

  @override
  State<MainApp> createState() => _MainAppState();
}

class _MainAppState extends State<MainApp> {
  bool _isBig = false;

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Beginner Animation'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              const Text(
                'Tap the box to animate',
              ),
              GestureDetector(
                onTap: () {
                  setState(() {
                    _isBig = !_isBig;
                  });
                },
                child: AnimatedContainer(
                  decoration: BoxDecoration(
                    borderRadius: BorderRadius.circular(8),
                    color: _isBig ? Colors.deepPurpleAccent : Colors.blueAccent,
                  ),
                  duration: const Duration(seconds: 1),
                  curve: Curves.easeInOutBack,
                  width: _isBig ? 250 : 100,
                  height: _isBig ? 500 : 100,
                  alignment: Alignment.center,
                  child: const Text(
                    'Box',
                    style: TextStyle(
                        color: Colors.white, fontWeight: FontWeight.bold),
```

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

Output:

Experiment No: 8 b) **Date:**

Aim:

To experiment with different types of animations (fade, slide, etc.).

Objective:

To understand the different types of animations in Flutter and how to implement them in a Flutter app.

Code Snippet (main.dart):

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

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

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: AnimationDemo(),
    );
  }
}

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

  @override
  State<AnimationDemo> createState() => _AnimationDemoState();
}

class _AnimationDemoState extends State<AnimationDemo>
    with TickerProviderStateMixin {
  late AnimationController _scaleController;
  late Animation<double> _scaleAnimation;
  late AnimationController _fadeController;
  late Animation<double> _fadeAnimation;
  late AnimationController _slideController;
  late Animation<Offset> _slideAnimation;
  late AnimationController _rotateController;
  late Animation<double> _rotateAnimation;
  bool _visible = false;
  bool _sliding = false;
  bool _rotating = false;
  bool _scaling = true;

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

    _fadeController = AnimationController(
      vsync: this,
      duration: const Duration(seconds: 1),
    );
    _fadeAnimation = Tween<double>(begin: 1, end: 0).animate(_fadeController);
```

```
  _slideController = AnimationController(
    vsync: this,
    duration: const Duration(seconds: 1),
  );
  _slideAnimation = Tween<Offset>(
    begin: const Offset(-1.0, 0.0),
    end: const Offset(1.0, 0.0),
  ).animate(_slideController);

  _rotateController = AnimationController(
    vsync: this,
    duration: const Duration(seconds: 2),
  );
  _rotateAnimation =
      Tween<double>(begin: 0, end: 1).animate(_rotateController);

  _scaleController = AnimationController(
    vsync: this,
    duration: const Duration(seconds: 2),
  )..repeat(reverse: true);
  _scaleAnimation = CurvedAnimation(
    parent: _scaleController,
    curve: Curves.easeInOut,
  );
}

@override
void dispose() {
  _scaleController.dispose();
  _fadeController.dispose();
  _slideController.dispose();
  _rotateController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Animation Demo'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          FadeTransition(
            opacity: _fadeAnimation,
            child: SlideTransition(
              position: _slideAnimation,
              child: RotationTransition(
                turns: _rotateAnimation,
                child: ScaleTransition(
                  scale: _scaleAnimation,
                  child: const FlutterLogo(size: 100),
                ),
              ),
            ),
          ),
          SwitchListTile(
            title: const Text('Fade'),
```

```
            value: _visible,
            onChanged: (bool value) {
              setState(() {
                _visible = value;
                _visible
                    ? _fadeController.forward()
                    : _fadeController.reverse();
              });
            },
          ),
          SwitchListTile(
            title: const Text('Slide'),
            value: _sliding,
            onChanged: (bool value) {
              setState(() {
                _sliding = value;
                _sliding
                    ? _slideController.repeat(reverse: true)
                    : _slideController.stop();
              });
            },
          ),
          SwitchListTile(
            title: const Text('Rotate'),
            value: _rotating,
            onChanged: (bool value) {
              setState(() {
                _rotating = value;
                _rotating
                    ? _rotateController.repeat()
                    : _rotateController.stop();
              });
            },
          ),
          SwitchListTile(
            title: const Text('Scale'),
            value: _scaling,
            onChanged: (bool value) {
              setState(() {
                _scaling = value;
                _scaling
                    ? _scaleController.repeat(reverse: true)
                    : _scaleController.stop();
              });
            },
          ),
        ],
      ),
    ),
  );
 }
}
```

Output:

Experiment No: 9 a) **Date:**

Aim:

To fetch data from a REST API.

Description:

To understand how to fetch data from a REST API in Dart using the http package.

Procedure:

1. Create a new Dart console project by running the following command in your terminal:
   ```
   dart create console_app
   ```
   The command creates a Dart project directory called console_app that contains a simple demo app in the \bin directory.
2. Change to the Dart project directory.
   ```
   cd console_app
   ```
3. Add the http package to your project by adding the following dependency to the pubspec.yaml file:
4. ```
   dependencies:
    http: ^1.2.2
   ```
   Save the file.
5. Run the following command to get the dependencies:
   ```
   dart pub get
   ```
   This command downloads the http package and adds it to your project.
6. Create a new Dart file called git_hub_api.dart in the lib directory of your project.
   Add the following code to the git_hub_api.dart file:
   ```
   // Utilizing GitHub API service
   import 'package:http/http.dart' as http;
   import 'dart:convert';

   Future<List<dynamic>> fetchRepos(String username) async {
     final response = await http.get(Uri.parse('https://api.github.com/users/$username/repos'));
     if (response.statusCode == 200) {
      return json.decode(response.body);
   } else {
      throw Exception('Failed to load repos');
     }
   }
   ```
7. Open the bin/console_app.dart file in your Dart project.
8. Replace the existing code with the following code snippet:
   ```
   // Utilizing GitHub API service from console_app
   import 'package:console_app/git_hub_api.dart' as git_hub_api;
   import 'dart:io';

   void main(List<String> arguments) async {
     String? username = arguments.isNotEmpty ? arguments.first : null;

     if (username == null) {
       stdout.write('Enter GitHub username: ');
       username = stdin.readLineSync();
     }

     if (username != null && username.isNotEmpty) {
       try {
         print('Fetching repositories for user: $username');
         List<dynamic> repos = await git_hub_api.fetchRepos(username);
         // Sort by 'updated_at' in descending order
         repos.sort(
           (a, b) => DateTime.parse(b['updated_at']).compareTo(
             DateTime.parse(a['updated_at']),
   ```
   ```

```
      ),
    );
    print('First 5 recent repositories:');
    for (var repo in repos.take(5)) {
      print(repo['name']);
    }
  } catch (e) {
    print('Error: $e');
  }
 } else {
  print('Username cannot be empty');
 }
}
```

9. Save the file.
10. Run your Dart project using the following command:
    dart run

Output:

Experiment No: 9 b) **Date:**

Aim:

To display the fetched data in a meaningful way in the UI.

Objective:

In this lab experiment, we will learn how to fetch data from an API using the http package and display the data in a user-friendly format using Flutter widgets such as *ListView* and *FutureBuilder*.

Procedure :

1. Create a new Flutter project by running the following command in your terminal:
   flutter create fetch_data_app
   The command creates a Flutter project directory called fetch_data_app.
2. Change to the Flutter project directory.
   cd fetch_data_app
3. Add the http package to your project by updating the pubspec.yaml file:
```
dependencies:
  http: ^0.15.0
```
   Run flutter pub get to install the package.
4. Create a new file lib/src/services/api_service.dart and implement the API service to fetch data:
```
// filepath: lib/src/services/api_service.dart
import 'package:http/http.dart' as http;
import 'dart:convert';

class ApiService {
  Future<List<dynamic>> fetchData() async {
    final response = await http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));
    if (response.statusCode == 200) {
      return json.decode(response.body);
    } else {
      throw Exception('Failed to load data');
    }
  }
}
```
5. Create a new file lib/src/screens/home_screen.dart and implement the UI to display the fetched data:
```
// filepath: lib/src/screens/home_screen.dart
import 'package:flutter/material.dart';
import '../services/api_service.dart';

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

  @override
  State<HomeScreen> createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  late Future<List<dynamic>> _data;

  @override
  void initState() {
    super.initState();
    _data = ApiService().fetchData();
  }

  @override
```

```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Fetched Data'),
      ),
      body: FutureBuilder<List<dynamic>>(
        future: _data,
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return const Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          } else if (snapshot.hasData) {
            return ListView.separated(
              itemCount: snapshot.data!.length,
              separatorBuilder: (context, index) => const Divider(),
              itemBuilder: (context, index) {
                final item = snapshot.data![index];
                return Padding(
                  padding: const EdgeInsets.all(16.0),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(
                        item['title'],
                        style: Theme.of(context).textTheme.titleLarge?.copyWith(
                              fontWeight: FontWeight.bold,
                              fontSize: 20,
                            ),
                      ),
                      const SizedBox(height: 8),
                      Text(
                        item['body'],
                        style: Theme.of(context).textTheme.bodyMedium?.copyWith(
                              fontSize: 16,
                            ),
                      ),
                    ],
                  ),
                );
              },
            );
          } else {
            return const Center(child: Text('No data available'));
          }
        },
      ),
    );
  }
}
```

6. Update the lib/main.dart file to set the HomeScreen as the home widget:

```
// filepath: lib/main.dart
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';
import 'src/screens/home_screen.dart';

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        useMaterial3: true,
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.teal),
        textTheme: GoogleFonts.lobsterTwoTextTheme(
          ThemeData.light().textTheme,
        ),
      ),
      home: const HomeScreen(),
    );
  }
}
```

7. Save all files.
8. Run your Flutter project using the following command:

```
flutter run
```

   Select the appropriate device to run the app.
9. During the app execution, you can use the following commands:

   a. Enter r to hot reload the app and see the changes you made to the code.
   b. Enter q to quit the app.

Output:

Experiment No: 10 a) **Date:**

Aim:

To write unit tests for UI components.

Procedure:

1. First create the flutter project
   Ctrl+shift+P
   Select the folder as well as application name: app21

2. Create the following dart files in the corresponding folders.

Example UI component and tests:

Let's assume you have a simple widget:

```
// lib/widgets/counter_button.dart
import 'package:flutter/material.dart';
class CounterButton extends StatefulWidget {
  final String label;
constCounterButton({Key? key, required this.label}) :super(key: key);

  @override
  State<CounterButton>createState() => _CounterButtonState();
}

class _CounterButtonState extends State<CounterButton> {
  int _count = 0;

  void _increment() {
setState(() {
      _count++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Text(widget.label),
Text('Count: $_count', key: const Key('countText')),
ElevatedButton(
          key: const Key('incrementButton'),
onPressed: _increment,
          child: const Text('Increment'),
        ),
      ],
    );
  }
}
```

Unit tests for CounterButton

Create a test file e.g. test/widgets/counter_button_test.dart:

```
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:app21/widgets/counter_button.dart';
```

```
void main() {
testWidgets('CounterButton displays initial label and count zero', (WidgetTester tester) async {
   // Arrange
const label = 'Tap me';

   // Act
   await tester.pumpWidget(
constMaterialApp(
      home: Scaffold(body: CounterButton(label: label)),
    ),
   );

   // Assert
   expect(find.text(label), findsOneWidget);
expect(find.byKey(const Key('countText')), findsOneWidget);
expect(find.text('Count: 0'), findsOneWidget);
expect(find.byKey(const Key('incrementButton')), findsOneWidget);
 });

testWidgets('CounterButton increments count on tap', (WidgetTester tester) async {
   await tester.pumpWidget(
constMaterialApp(
      home: Scaffold(body: CounterButton(label: 'Test')),
    ),
   );

   // Tap the button
   await tester.tap(find.byKey(const Key('incrementButton')));
   // Let UI rebuild
   await tester.pump();

   // Expect count to be 1
expect(find.text('Count: 1'), findsOneWidget);

   // Tap again
   await tester.tap(find.byKey(const Key('incrementButton')));
   await tester.pump();

expect(find.text('Count: 2'), findsOneWidget);
 });
}
```

Output:

Explanation of what's being tested

In the first test: we verify initial state (label is shown, count starts at 0, button exists).
In the second test: we simulate a user interaction (tap), let the UI update (pump()), and verify the count changed accordingly.

3. Run the flutter project
   flutter test

4. The output in the terminal
   00:05 +3: All tests passed!

Experiment No: 10 b) **Date:**

Aim:

To use Flutter's debugging tools to identify and fix issues.

Description:

Flutter's debugging tools help developers:

- Inspect widget layout and hierarchy
- Identify performance bottlenecks
- Catch rendering, state, or logic errors
- Fix UI and runtime issues efficiently

Common Debugging Tools in Flutter

Tool	Purpose	Usage
Flutter DevTools	Visual UI inspector, performance, memory profiler	Runs via Chrome/VS Code
Dart DevTools Console	Shows runtime logs, print outputs, errors	flutter run or IDE terminal
Debug Banner	Indicates debug mode	debugShowCheckedModeBanner: false
Flutter Inspector	Examine widget tree & properties	VS Code / Android Studio → "Flutter Inspector"
Breakpoints	Pause execution at specific lines	Add via IDE or debugger(); in code
Hot Reload & Hot Restart	Quickly apply code changes	r (reload), R (restart) in terminal
Error Widgets & Stack Traces	Identify crashes	Automatically shown in red/yellow error screen

Example: Debugging a UI Issue

Buggy Code:

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

class DebugExample extends StatelessWidget {
  final int counter = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
appBar: AppBar(title: Text("Debug Demo")),
      body: Center(
        child: Column(
mainAxisAlignment: MainAxisAlignment.center,
          children: [
Text('Counter: $counter'),
ElevatedButton(
onPressed: () {
                counter++; // □ Error: counter is final and can't be modified
              },
              child: Text('Increment'),
            ),
          ],
        ),
      ),
    );
  }
```

}

Debugging Steps

- Run the app in debug mode
- flutter run

Observe the red error screen (Flutter's error widget) that tells you:
Unhandled Exception: Unsupported operation: counter is final

- Open Flutter Inspector
  a. In **VS Code → View → Command Palette → Flutter: Open DevTools**
  b. Examine the widget tree hierarchy
  c. Verify if stateful logic is incorrectly placed in a StatelessWidget

- Fix the issue

Corrected Code:

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

class DebugExample extends StatefulWidget {
  @override
  State<DebugExample>createState() => _DebugExampleState();
}

class _DebugExampleState extends State<DebugExample> {
  int counter = 0;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
appBar: AppBar(title: Text("Debug Demo")),
      body: Center(
        child: Column(
mainAxisAlignment: MainAxisAlignment.center,
          children: [
Text('Counter: $counter'),
ElevatedButton(
onPressed: () {
setState(() {
                counter++;
              });
            },
            child: Text('Increment'),
          ),
        ],
      ),
    ),
  );
  }
}
```

Use Breakpoints

- Place a breakpoint inside setState() to verify counter increments.
- Inspect variable changes in DevTools.

Experiment No: 1 a) **Date:**

Aim:

To Install Flutter and Dart SDK.

System Requirements

1. Hardware Requirements

To install and run Flutter, your Windows environment must meet the following hardware requirements:

Requirement	Minimum	Recommended
x86_64 CPU Cores	4	8
Memory in GB	8	16
Display resolution in pixels	WXGA (1366 x 768)	FHD (1920 x 1080)
Free disk space in GB	11.0	60.0

2. Software Requirements

To write and compile Flutter code for Android, you must have the following version of Windows and the listed software packages.
Note: Do not need to install Dart separately as the Flutter SDK includes the full Dart SDK.

Operating System

Flutter supports 64-bit version of Microsoft Windows 10 or later. These versions of Windows should include the required Windows PowerShell 5 or later.

Development Tools

Download and install the Windows version of the following packages:
- Git for Windows **2.27 or later** to manage source code.
- Android Studio **2023.3.1 (Jellyfish) or later** to debug and compile Java or Kotlin code for Android. Flutter requires the full version of Android Studio.

The developers of the preceding software provide support for those products. To troubleshoot installation issues, consult that product's documentation.

When you run the current version of flutter doctor, it might list a different version of one of these packages. If it does, install the version it recommends.

3. Configure a Text Editor or IDE

We can build apps with Flutter using any text editor or integrated development environment (IDE) combined with Flutter's command-line tools.

Using an IDE with a Flutter extension or plugin provides code completion, syntax highlighting, widget editing assists, debugging, and other features.

Popular options include:
- Visual Studio Code 1.77 or later with the Flutter extension for VS Code.
- Android Studio 2023.3.1 (Jellyfish) or later with the Flutter plugin for IntelliJ.
- IntelliJ IDEA 2023.3 or later with the Flutter plugin for IntelliJ.

Recommended: The Flutter team recommends installing Visual Studio Code 1.77 or later and the Flutter extension for VS Code This combination simplifies installing the Flutter SDK.

4. Procedure

Step 1: Download Flutter SDK

- Visit the Flutter official website.
- Choose your operating system (Windows, macOS, Linux).
- Download the Flutter SDK zip file.

Step 2: Extract Flutter SDK

- Extract the downloaded zip file to a desired location on your system.
- Add the flutter/bin directory to your system's PATH environment variable.

Step 3: Verify Installation

- Open a command line interface.
- Run the command: **flutter doctor**
- Follow any additional setup instructions provided by flutter doctor.

Step 4: Install Dart SDK

- Dart SDK is bundled with Flutter, so no separate installation is required.
- Verify Dart installation by running: **dart –version**

5. Conclusion

You have successfully installed Flutter and Dart SDK on your machine. You are now ready to start developing Flutter applications.

Experiment No: 1 b) **Date:**

Aim:

To write a simple Dart program to understand the language basics.

a) Create a program that asks the user to enter their name and their age. Print out a message that tells how many years they have to be 100 years old.

Program:

```
import 'dart:io';

void main() {
   stdout.write("What's your name? ");
   String? name = stdin.readLineSync();

   print("Hi, $name! What is your age?");
   int age = int.parse(stdin.readLineSync() ?? '0');

   int yearsToHunderd = 100 - age;
   print("$name, You have $yearsToHunderd years to be 100");
}
```

Output:

b) Create a program to ask the user for a number. Depending on whether the number is even or odd, print out an appropriate message to the user.

Program:

```
import 'dart:io';

void main() {
   stdout.write("Hi, please choose a number: ");
   int number = int.parse(stdin.readLineSync() ?? '-1');

   if (number % 2 == 0) {
      print("Chosen number is even");
   }
   else {
      print("Chosen number is odd");
   }
}
```

Output:

c) Create a program that asks the user for a number and then prints out a list of all the divisors of that number.

Program:

```
import 'dart:io';

void main() {
  stdout.write("Please choose a number: ");
  int number = int.parse(stdin.readLineSync() ?? '0');
  for (var i = 1; i <= number; i++) {
    if (number % i == 0) {
      print(i);
    }
  }
}
```

Output:

d) Create a program to ask the user for a string and print out whether this string is a palindrome or not.

Program:

```
import 'dart:io';

void main() {

  stdout.write("Please give a word: ");
  String input = stdin.readLineSync()?.toLowerCase() ?? ' ';
  String revInput = input.split('').reversed.join('');

  // Ternary operator
  input == revInput
      ? print("The word is palindrome")
      : print("The word is not a palindrome");
}
```

Output:

e) Write a program that asks the user how many Fibonacci numbers to generate and then generates them [use functions].

Program:

```
import 'dart:io';

void main() {
  stdout.write("How many Fibonacci numbers do you want? ");
  int chosenNumber = int.parse(stdin.readLineSync() ?? '0');

  List<int> result = fibonacciNumbers(chosenNumber);
  print(result);
}

// Function to calulcate the Fibonacci numbers
List<int> fibonacciNumbers(int chosenNumber) {
  List<int> fibList = [1, 1];

  for (var i = 0; i < chosenNumber; i++) {
    fibList.add(fibList[i] + fibList[i + 1]);
  }
  return fibList;
}
```

Output:

f) Write a program that ask the user for a number and determine whether the number is prime or not.

Program:

```
import 'dart:io';

void main() {
  stdout.write("Please give us a number: ");
  int chosenNumber = int.parse(stdin.readLineSync() ?? '0');

  checkPrime(chosenNumber);
}

void checkPrime(int number) {
  // List comprehensions
  List<int> a = [
    for (var i = 1; i <= number; i++)
      if (number % i == 0) i
  ];

  // Check for prime
  a.length == 2
      ? print("The chosen number is a prime")
      : print("The chosen number is not a prime");
}
```

Output:

Running the code

- Download the Dart SDK from here and install it on your machine.
- Write your code in .dart extension files or clone the git repository
- Run your code from the terminal

```
PowerShell 6.2.4
Copyright (c) Microsoft Corporation. All rights reserved.

PS C:\Users\Desktop dart code-file.dart
```