

## 5 a) Learn about stateful and stateless widgets.

In Flutter, widgets can be categorized into two main types based on their behaviour regarding state management: stateful widgets and stateless widgets.

### Stateless Widgets:

**Definition:** Stateless widgets are widgets that do not have any mutable state. Once created, their properties (configuration) cannot change.

### Characteristics:

1. They are immutable and lightweight.
2. They only depend on their configuration and the build context provided during construction.
3. Their appearance (UI) is purely a function of their configuration.
4. They are ideal for UI elements that do not change over time, such as static text labels, icons, or simple buttons.

### Stateful Widgets:

**Definition:** Stateful widgets are widgets that maintain state, allowing them to change and update over time in response to user actions, network events, or other factors.

### Characteristics:

1. They have an associated mutable state that can change during the widget's lifetime.
2. The state is stored in a separate class that extends State and is associated with the stateful widget.
3. Changes to the state trigger a rebuild of the widget's UI, allowing dynamic updates.
4. They are ideal for UI elements that need to change or react to user interactions, such as input forms, animations, or scrollable lists.

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Cards Example',
      theme: ThemeData(primarySwatch: Colors.blue),
    );
  }
}
```

```

        home: const MyHomePage(),
    );
}
}

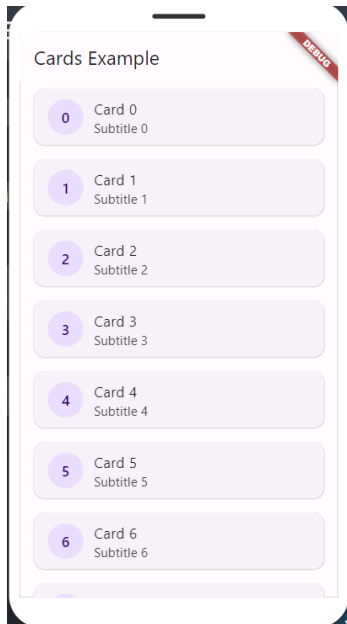
class MyHomePage extends StatelessWidget {
  const MyHomePage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Cards Example')),
      body: ListView.builder(
        itemCount: 10,
        itemBuilder: (context, index) {
          return Card(
            margin: const EdgeInsets.symmetric(horizontal: 16, vertical: 8),
            child: ListTile(
              title: Text('Card $index'),
              subtitle: Text('Subtitle $index'),
              leading: CircleAvatar(child: Text('$index')),
              onTap: () {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(content: Text('Card tapped: $index')),
                );
              },
            ),
          );
        },
      ),
    );
  }
}

```

```
}
```

## Output:



## 5 b) Implement state management using set state and provider.

A simple counter app using `setState` to manage the state.

**Using `setState`:** It's suitable for simple state management in small widgets or when you have straightforward state needs. The state is local to the widget and requires rebuilding the widget tree.

The counter state is managed locally within the `CounterScreen` widget. When the button is pressed, `setState` is called to update the `_counter` variable, and the widget rebuilds to reflect the new count.

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

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

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

```

return MaterialApp(
  title: 'SetState Example',
  home: CounterScreen(),
);
}
}

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

class _CounterScreenState extends State<CounterScreen> {
  int _counter = 0;
  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Counter')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Text('Counter: $_counter'),
            ElevatedButton(
              onPressed: _incrementCounter,
              child: Text('Increment'),
            ),
          ],
        ),
      ),
    );
  }
}

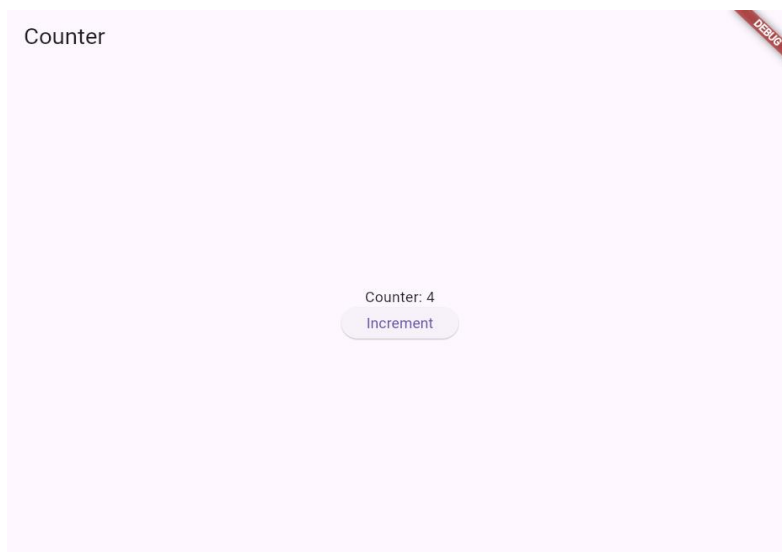
```

```

    ],
  ),
),
);
}
}

```

### Output:



**Using Provider:** This approach is better for larger applications or when you need to share state across multiple widgets. The Provider package makes it easier to manage and listen to changes in the state, keeping your code clean and organized.

The CounterModel class manages the counter state, and the ChangeNotifierProvider makes this state accessible to the CounterScreen and other potential widgets in the widget tree. When the button is pressed, the increment method is called, updating the state and notifying listeners to rebuild.

```

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

void main() {
  runApp(
    ChangeNotifierProvider(
      create: (context) => CounterModel(),

```

```

        child: MyApp(),
    ),
);
}

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

class CounterModel extends ChangeNotifier {
  int _counter = 0;

  int get counter => _counter;

  void incrementCounter() {
    _counter++;
    notifyListeners();
  }
}

class CounterPage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final counterModel = Provider.of<CounterModel>(context);

    return Scaffold(
      appBar: AppBar(

```

```

        title: Text('State Management Example'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'Counter Value: ${counterModel.counter}',
            style: TextStyle(fontSize: 20),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: counterModel.incrementCounter,
            child: Text('Increment Counter'),
          ),
        ],
      ),
    ),
  );
}

```

### Output:



