Module-3

Introduction to Flutter Widgets and UI Components.

1. Explain the difference between Stateless and Stateful widgets with examples.

Stateless Widget:

(1.) A StatelessWidget does not change once it's built.

(2.) It's like a photo - same every time we look at it.

(3.) When to use: when the screen or UI doesn't need to change.

(4.) It's used for static content like labels, icons, or titles.

(5.) Example:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp(home: MyWidget()));
}
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Text("I never change!")
      ),
    );
  }
}
```

(2) Stateful Widget:

(1.) A StatefulWidget can change its content while the app is running.

(2.) It's like a video - it moves and updates.

(3.) When to use: when the UI changes based on user actions.

(4.) It's used for counting button taps, switching themes, showing or hiding items, etc.

(5.) Example:

```
void main() {
  runApp(MaterialApp(home: MyWidget()));
}
class MyWidget extends StatefulWidget {
  @override
  MyWidgetState createState() =>
                    MyWidgetState();
}
class MyWidgetstate extends State {
  int value = 0;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: Column(
          SizedBox(height: 15),
          Text("Counter Value: $value"),
          SizedBox(height: 15),
          ElevatedButton(
            onPressed: () {
              setstate(() {
```

```
                    value++;
                });
            },
            child: Text("Count")
        )
    ],
  ),
 ),
 );
 };
}
```

2. Describe the widget lifecycle and how state is managed in stateful widgets.
When we use a Stateful Widget, Flutter needs to:
- Create it
- Update it when things change
- Destroy it when no longer needed

This journey from birth → changes → death is called the widget lifecycle.

Lifecycle of a StatfulWidget:
A StatefulWidget has two parts:
- The widget (the blueprint/UI description)
- The state (the actual data that can change).

Here are the important lifecycle methods (steps):

(1.) createState(): Called only once when the widget is created.
It makes the state object that holds data.

(2.) initState(): Runs once at the start, when the state object is first created.
Best place to set up things like:
- Fetching initial data
- Starting animation
- Setting default values.

(3.) build(): Called whenever the UI needs to be drawn.
It shows the current state on the screen.
Runs again when setState() is called.

(4.) setState(): Used to update the data (state).
When called, it rebuilds the widget so the UI shows the new data.
Example: incrementing a counter.

(5.) didUpdateWidget(): Runs when the widget is replaced by another widget of the same type.
Example: When parent widget sends new data.

(6.) dispose(): Called when the widget is removed permanently.

Good place to:
- Close streams
- Stop animations
- Free up resources.

Managing State: The state (data that can change) lives incide the State object, not in the widget itself. To update UI, we can never build rebuild the widget manually, instead we call setState(), and Flutter rebuilds it for us.

3. List and describe for common Flutter layout widgets (e.g., Container, Column, Row).

(1.) Container: A box that can hold a single child widget. We can style it with padding, margin, color, borders and size.
- Use when we need to decorate or position a widget.

(2.) Row: Arranges widgets in horizontal line (left → right).
- Use when we want item side by side.
- Example: buttons next to each other.

(3.) Column: Arranges widgets in a vertical line (top → bottom).
- Use when we want items stacked.

• Example: form fields placed one below another.

(4.) Stack: Places widgets on top of each other (like layers).
• Use when we want to overlap items.
• Example: putting text over an image.

(5.) Expanded: Used inside a Row or Column to make a widget take available space.
• Use when we want one widget to stretch and fill space while others take their needed size.