* **Explain the difference between Stateless and Stateful widgets with examples**

1. **Stateless Widget: -Stateless widget don’t change-once built, their properties are fixed and they stay the same throughout their lifetime.**

**Example: -**

**class My Label extends Stateless Widget {**

**@override**

**Widget build (Build Context context) {**

**return Text ("This is static!");**

**}**

**}**

1. **Stateful Widget: - stateful widget do change – they can update when users interact with them, receive new data, or undergo any action that changes their state.**

**Example:**

**class My Counter extends Stateful Widget {**

**@override**

**State<My Counter> create State () => \_MyCounterState ();**

**}**

**class \_MyCounterState extends State<My Counter> {**

**int \_count = 0;**

**@override**

**Widget build (Build Context context) {**

**return Column (**

**children: [**

**Text ('Count: �\_count'),**

**Elevated Button (**

**on Pressed: () {**

**set State (() {**

**\_count++;**

**});**

**},**

**child: Text('Increment'),**

**),**

**],**

**);**

**}**

**}**

* **Describe the widget lifecycle and how state is managed in Stateful widgets.**

**Widget Lifecycle in Flutter: The lifecycle of a widget describes the sequence of events from creation to disposal. Flutter provides a set of methods you can override to hook into these stages:**

**For Stateless Widgets:**

**build (): Called when the widget needs to be rendered; they don’t have lifecycle methods like initState () because they are immutable.**

**For Stateful Widgets**

**Stateful widgets have a more complex lifecycle involving 7 stages:**

1. **create State (): When a widget is inserted into the widget tree, Flutter calls this method to create a mutable State object associated with the widget.**
2. **initState (): Called once when the State object is first created. You can initialize data here, such as setting up controllers or fetching initial data.**
3. **didChangeDependencies (): Called immediately after initState () and whenever dependencies change; useful for responding to inherited widget changes.**
4. **build (): Called to describe the part of the user interface represented by this widget. It can be called multiple times during the lifecycle.**
5. **didUpdateWidget (): Called when the parent widget changes and needs to rebuild this widget. It provides the old widget for comparison.**
6. **setState (): When the widget's data changes, calling this method signals Flutter to rebuild UI with the latest state.**
7. **deactivate (): Called when the widget is removed from the widget tree but might be reinserted later.**

**Example Diagram of lifecycle phases: text create State () → initState () → build () → (interactions & updates) → didUpdateWidget () → setState () → build () → deactivate () → dispose ()**

**How is State Managed?**

* **You update the data within the State.**
* **Then, you call setState () with a callback that modifies the state.**
* **Flutter schedules a rebuild, calling build () again, which updates the UI.**
* **List and describe five common Flutter layout widgets (e.g., Container, Column, Row).**

1. **Container A Container is a versatile widget that allows for customization of its child widget by adding padding, margins, borders, and background colour. It is often used for drawing a box around a widget or creating spacing between widgets in a layout. ​**
2. **Column Column is a widget that arranges its child widgets vertically, stacking them from top to bottom. It is useful for displaying lists of elements or vertically organized layouts, like forms, message lists, or buttons. ​**
3. **Row Row arranges its child widgets horizontally, placing them side by side along the main axis. It is ideal for organizing content in a horizontal sequence, such as icons, buttons, or navigation options. ​**
4. **Stack Stack allows for overlapping its child widgets, layering them on top of each other. This is handy for creating complex layouts where widgets need to be positioned in absolute relation to the container, such as overlays, badges, or cards with background images. ​**
5. **List View List View is a scrollable widget that lays children out in a linear array vertically, providing automatic scrolling when its content exceeds the visible area. It is commonly used to display dynamic or lengthy lists, such as feeds or item catalogs. ​**