Q1. Explain the difference between Stateless and Stateful widgets with examples.

Stateless Widget

- A **StatelessWidget** is a widget that does not maintain any state.
- Once created, its properties cannot change during the lifetime of the widget.
- Commonly used for **static UI elements** that remain constant, such as labels, icons, or decorative UI.
- **Advantages**: Lightweight, fast, and efficient since it doesn't require state management.

Example (Stateless Widget):

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

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

  @override
  Widget build(BuildContext context) {
    return const Center(
      child: Text(
        " Stateless Widget",
        style: TextStyle(fontSize: 20),
      ),
    );
  }
}
```

Stateful Widget

- A **StatefulWidget** is a widget that can **change dynamically** during runtime.
- It maintains a State object, which holds data that may change during the widget's lifetime.
- When the state changes, setState() is called, and the widget **rebuilds itself** to reflect new data.
- **Advantages**: Allows interactive and dynamic UIs such as counters, forms, lists, animations.

Example (Stateful Widget):

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

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

  @override
  _MyStatefulState createState() => _MyStatefulState();
}

class _MyStatefulState extends State<MyStateful> {
  int counter = 0;

  void increaseCounter() {
    setState(() {
      counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text("Counter: $counter", style: const TextStyle(fontSize: 20)),
        ElevatedButton(
          onPressed: increaseCounter,
          child: const Text("Increase"),
        ),
      ],
    );
  }
}
```

Key Differences

Feature	**Stateless Widget**	**Stateful Widget**
State	Immutable	Mutable
Rebuild	Only rebuilt if parent updates	Rebuilt when setState() is called
Performance	Lightweight, faster	Heavier due to state management
Use Case	Static UI (icons, titles, logos)	Interactive UI (counters, forms, sliders)

Q2. Describe the widget lifecycle and how state is managed in Stateful widgets.

Widget Lifecycle in Flutter

Flutter widgets go through a **lifecycle** when they are created, updated, and destroyed. For **StatefulWidgets**, lifecycle is more important because they maintain state.

Lifecycle Methods of StatefulWidget

1. **createState()**
   - Called when the widget is created.
   - Creates an instance of the State class.
2. **initState()**
   - Called once when the state object is first created.
   - Used for **initializations** like fetching data, setting variables, starting animations.
3. **build()**
   - Called every time the widget is rebuilt.
   - Returns the widget's UI.
4. **didUpdateWidget()**
   - Called when the widget configuration changes (e.g., parent provides new data).
5. **setState()**
   - Tells Flutter that the internal state has changed.
   - Triggers a rebuild of the widget.

6. **dispose()**
   - Called when the widget is removed from the widget tree.
   - Used for cleanup (closing streams, controllers, etc.).

State Management in Stateful Widgets

- **State** is stored in the State class.
- When something changes (like a button click), setState() is called.
- This rebuilds only that widget, making Flutter efficient.

Example with Lifecycle:

```
class LifecycleExample extends StatefulWidget {
  @override
  _LifecycleExampleState createState() => _LifecycleExampleState();
}

class _LifecycleExampleState extends State<LifecycleExample> {
  @override
  void initState() {
    super.initState();
    print("initState called");
  }

  @override
  Widget build(BuildContext context) {
    return const Center(child: Text("Lifecycle Example"));
  }

  @override
  void dispose() {
    print("dispose called");
    super.dispose();
  }
}
```

Q3. List and describe five common Flutter layout widgets (e.g., Container, Column, Row).

1. Container

- A versatile widget used for **styling and positioning**.
- Can have padding, margin, color, border, and child widgets.

---

2. Column

- Arranges child widgets in a **vertical direction**.
- Useful for stacking UI elements on top of each other

```
Column(
  children: [
    Text("Line 1"),
    Text("Line 2"),
  ],
)
```

---

3. Row

- Arranges child widgets in a **horizontal direction**.
- Often used with MainAxisAlignment and CrossAxisAlignment.

```
 Row(

  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Icon(Icons.home),
    Icon(Icons.star),
    Icon(Icons.settings),
  ],
)
```

---

4. Stack

- Places widgets on **top of each other** (like layers).
- Useful for overlapping widgets (e.g., text over image).

```
Stack(

  children: [
    Image.asset("background.png"),
    const Text("Overlay Text"),
  ],
)
```

---

5. ListView

- A scrollable list of widgets.
- Used for displaying large sets of data efficiently.

```
Stack(
  children: [
    Image.asset("background.png"),
    const Text("Overlay Text"),
  ],
)
```