Module 3: Introduction to Flutter Widgets and UI Components

**Theory Assignments:**

**Q.** 1

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

**A. In Flutter, widgets are the basic building blocks of a Flutter app’s user interface. They come in two main types:**

**Stateless Widget**

**A Stateless widget is immutable — once it's built, it cannot change during the app's lifetime. These widgets do not store any state that affects how they look or behave.**

**Use When:**

* **The UI does not change dynamically**
* **No user interaction or data change is expected**

**Example: StatelessWidget**

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

**void main() => runApp(MyApp());**

**class MyApp extends StatelessWidget {**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**home: MyStatelessWidget(),**

**);**

**}**

**}**

**class MyStatelessWidget extends StatelessWidget {**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(title: Text('Stateless Widget')),**

**body: Center(child: Text('This is a Stateless Widget')),**

**);**

**}**

**}**

**Stateful Widget**

**A Stateful widget can change its state during its lifetime. This means it can rebuild itself when data changes or in response to user interaction.**

**Use When:**

* **UI needs to update dynamically**
* **Handling inputs, animations, API data, toggles, etc.**

**Example: StatefulWidget with a counter**

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

**void main() => runApp(MyApp());**

**class MyApp extends StatelessWidget {**

**@override**

**Widget build(BuildContext context) {**

**return MaterialApp(**

**home: MyStatefulWidget(),**

**);**

**}**

**}**

**class MyStatefulWidget extends StatefulWidget {**

**@override**

**\_MyStatefulWidgetState createState() => \_MyStatefulWidgetState();**

**}**

**class \_MyStatefulWidgetState extends State<MyStatefulWidget> {**

**int \_counter = 0;**

**void \_incrementCounter() {**

**setState(() {**

**\_counter++; // State changes here**

**});**

**}**

**@override**

**Widget build(BuildContext context) {**

**return Scaffold(**

**appBar: AppBar(title: Text('Stateful Widget')),**

**body: Center(**

**child: Text('Counter: $\_counter', style: TextStyle(fontSize: 24)),**

**),**

**floatingActionButton: FloatingActionButton(**

**onPressed: \_incrementCounter,**

**child: Icon(Icons.add),**

**),**

**);**

**}**

**}**

**Key Differences**

| **Feature** | **StatelessWidget** | **StatefulWidget** |
| --- | --- | --- |
| **Data Changes** | **Cannot change** | **Can change over time** |
| **Lifecycle Methods** | **Only build()** | **createState(), initState(), etc.** |
| **UI Rebuilding** | **Only on hot reload** | **Rebuilds using setState()** |
| **Performance** | **Slightly better** | **Slightly heavier due to state tracking** |
| **Example Use** | **Logo, text, icons** | **Forms, counters, animations** |

**In Short**

* **Use StatelessWidget when the UI doesn’t depend on any user interaction or data updates.**
* **Use StatefulWidget when the widget’s appearance or behavior needs to change dynamically during runtime.**

**Q.** 2

**Q.** 2. Describe the widget lifecycle and how state is managed in Stateful widgets.

**A. Widget Lifecycle in Flutter (StatefulWidget)**

**In Flutter, every StatefulWidget is associated with a State object that holds mutable state. The lifecycle of a stateful widget involves the creation, updating, and disposal of this State object.**

**Key Lifecycle Methods of Stateful Widgets:**

1. **createState()**
   * **Called once when the widget is inserted into the widget tree.**
   * **Returns an instance of the State class associated with the widget.**
2. **initState()**
   * **Called once after createState() and before the widget is built.**
   * **Ideal for one-time initializations like data loading or animations.**
3. **build(BuildContext context)**
   * **Called every time the widget needs to be rendered.**
   * **Returns the UI of the widget based on the current state.**
4. **didUpdateWidget(oldWidget)**
   * **Called when the parent widget rebuilds and passes a new configuration.**
   * **Used to compare the old and new widgets to update state accordingly.**
5. **setState(fn)**
   * **Triggers a rebuild by marking the widget as dirty.**
   * **The function inside setState modifies the state, and Flutter calls build() again.**
6. **deactivate()**
   * **Called when the widget is removed temporarily from the widget tree.**
7. **dispose()**
   * **Called when the widget is permanently removed.**
   * **Used to release resources like controllers or streams.**

**How State is Managed**

* **The State object holds mutable state (data that changes over time).**
* **The widget itself is immutable; only the State class can change.**
* **Flutter calls the build() method again only when setState() is invoked.**
* **State is preserved as long as the StatefulWidget remains in the tree with the same key.**

**In Short**

| **Concept** | **Description** |
| --- | --- |
| **Widget** | **Configuration, immutable** |
| **State** | **Holds mutable data** |
| **setState()** | **Rebuilds UI on state change** |
| **Lifecycle** | **From creation → build → update → disposal** |

**Q.** 3

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

## A. Five Common Flutter Layout Widgets – Perfect Theory

## Flutter provides powerful layout widgets to design flexible, responsive UIs. Below are five of the most commonly used layout widgets with their theoretical descriptions:

## 1. Container

## Purpose: A versatile box that can hold a single child and apply padding, margin, color, size, alignment, and decoration.

## Use Case: For styling and positioning a widget or grouping visual properties.

## Key Properties: padding, margin, width, height, color, alignment, decoration

## 2. Column

## Purpose: Arranges its children vertically, from top to bottom.

## Use Case: When you want to stack widgets vertically like text, images, buttons.

## Key Properties: mainAxisAlignment, crossAxisAlignment, children

## 3. Row

## Purpose: Arranges its children horizontally, from left to right.

## Use Case: Used for placing widgets side by side, like icon + text, or buttons in a row.

## Key Properties: mainAxisAlignment, crossAxisAlignment, children

## 4. Stack

## Purpose: Allows widgets to overlap on top of each other, in layers.

## Use Case: Ideal for UI elements like badges, profile image overlays, or complex designs.

## Key Properties: alignment, fit, children

## 5. Expanded

## Purpose: Expands a child of a Row, Column, or Flex to take up remaining space.

## Use Case: Useful when you want one or more widgets to share available space flexibly.

## Key Properties: flex, child

## Conclution

| Widget | Layout Direction | Key Feature |
| --- | --- | --- |
| Container | N/A | Decoration and box model styling |
| Column | Vertical | Stacks children vertically |
| Row | Horizontal | Stacks children horizontally |
| Stack | Overlay | Overlaps widgets like layers |
| Expanded | Flexible | Shares space within Flex widgets |

## These widgets are foundational to nearly all Flutter UI designs.