College of Science and Technology

Flutter

Royal University of Bhutan

# Unit IV:
# Cross-Platform App Development
# (Flutter Project)

## CTE308- AS2025

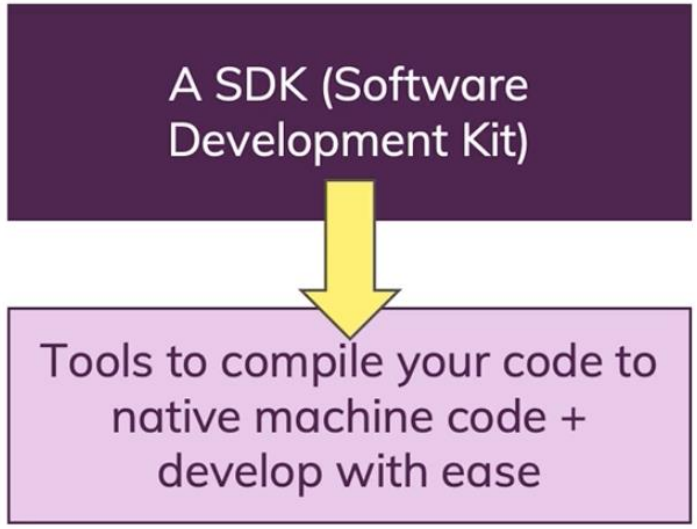
Tutor: Pema Galey
#17682761

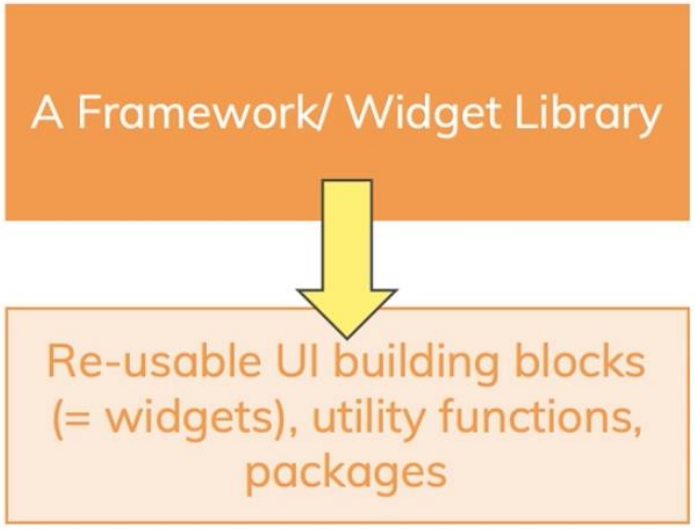
# Outlines

- Flutter Project
- Dart Programming
- Building App from Scratch

# What is Flutter?

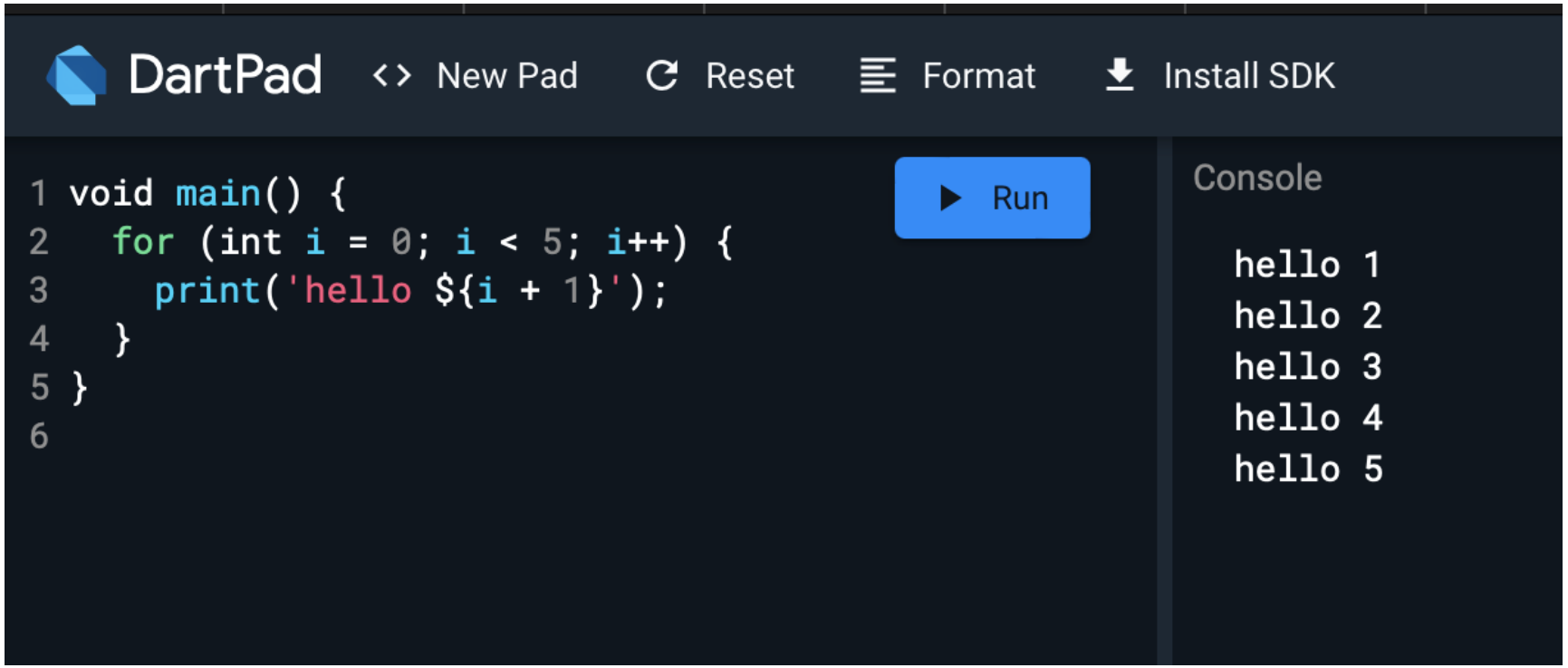
- A ”too” that allow you to build native cross-platform (iOS, Android) apps with  one programming language and codebase.



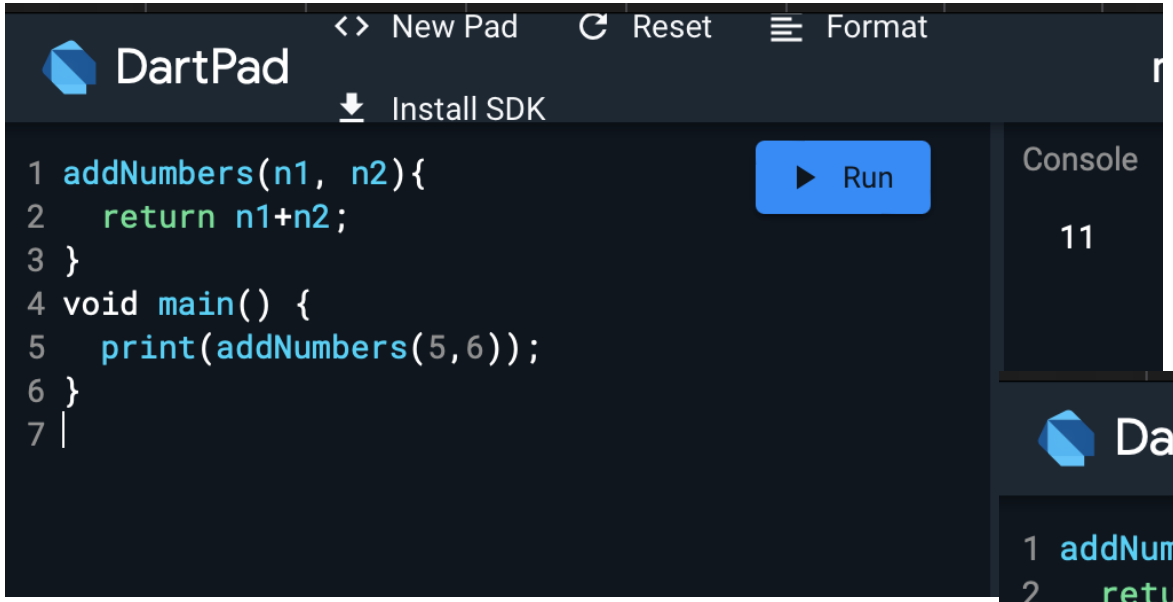



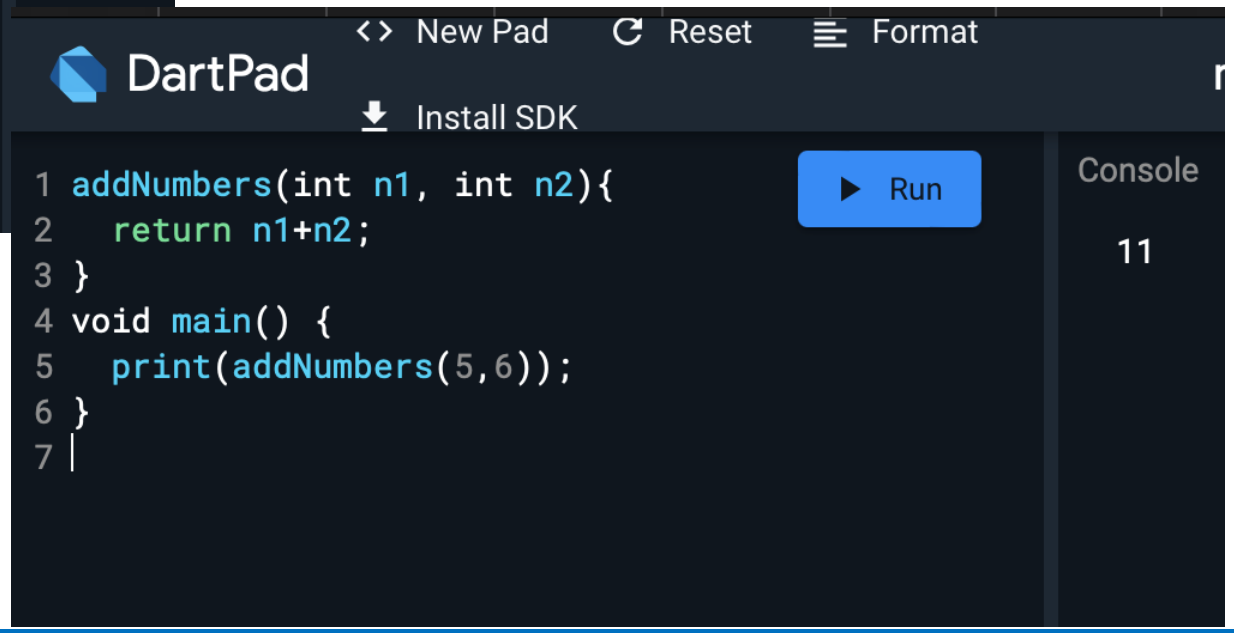
# Dart Programming

- Link: *https://dartpad.dev/?null_safety=true*

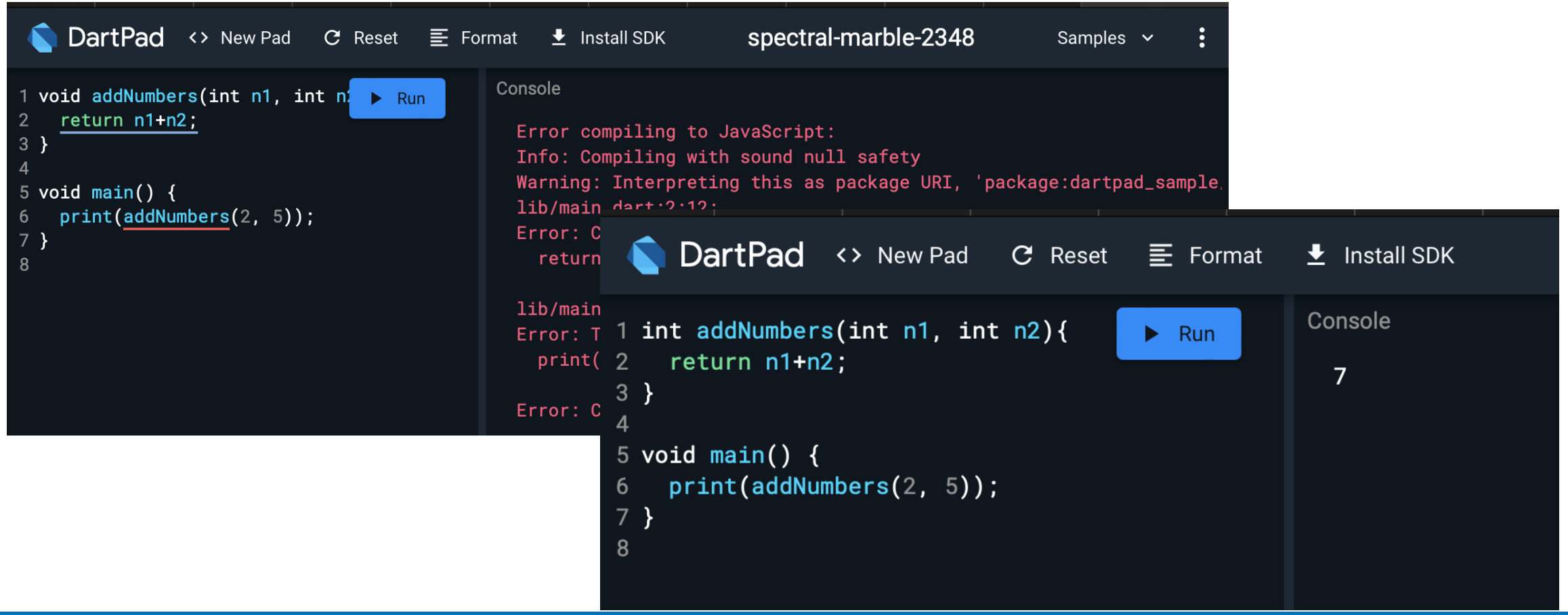


# Dart Programming



- Case Sensitive

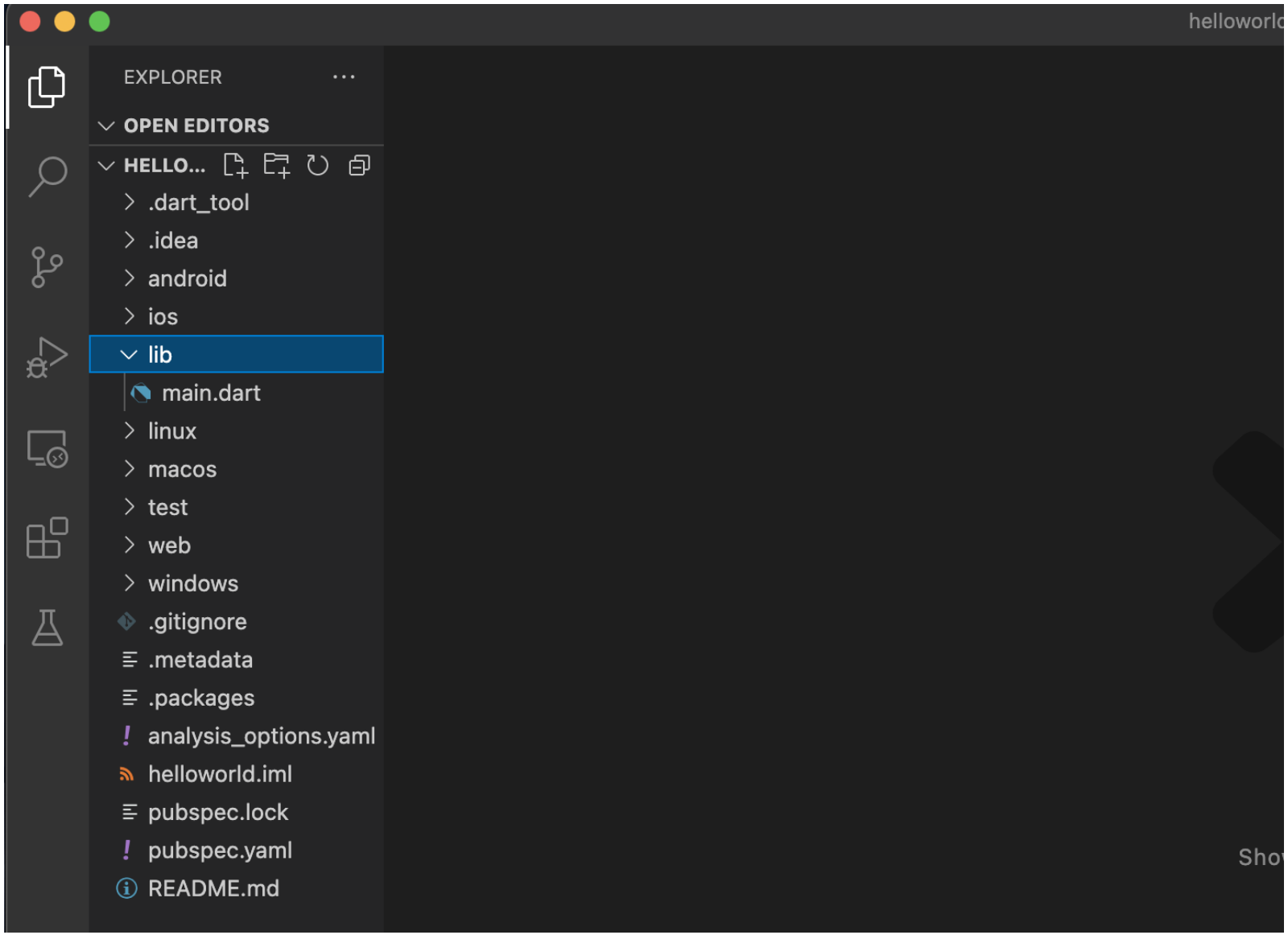


# Create Flutter Project

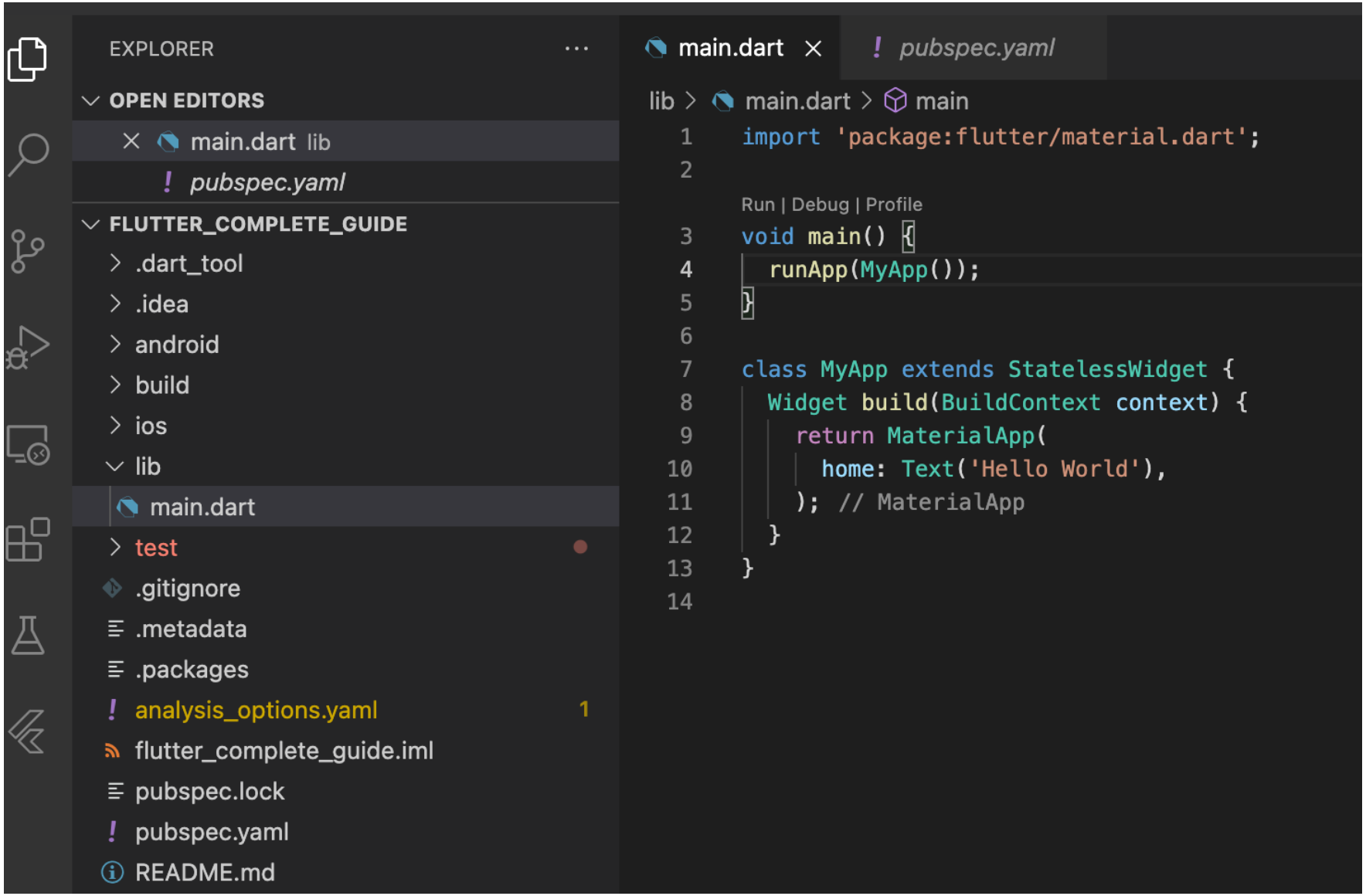
- **Create and run a simple Flutter app**

1. Create a new Flutter app by running the following from the command line:
   - `flutter create my_app`
2. A **`my_app`** directory is created, containing Flutter's starter app. Enter this directory:
   - `cd my_app`
3. To launch the app in the Simulator, ensure that the Simulator is running and enter:
   - `flutter run`

# Flutter Project



# Creating basic App



# Basic Design

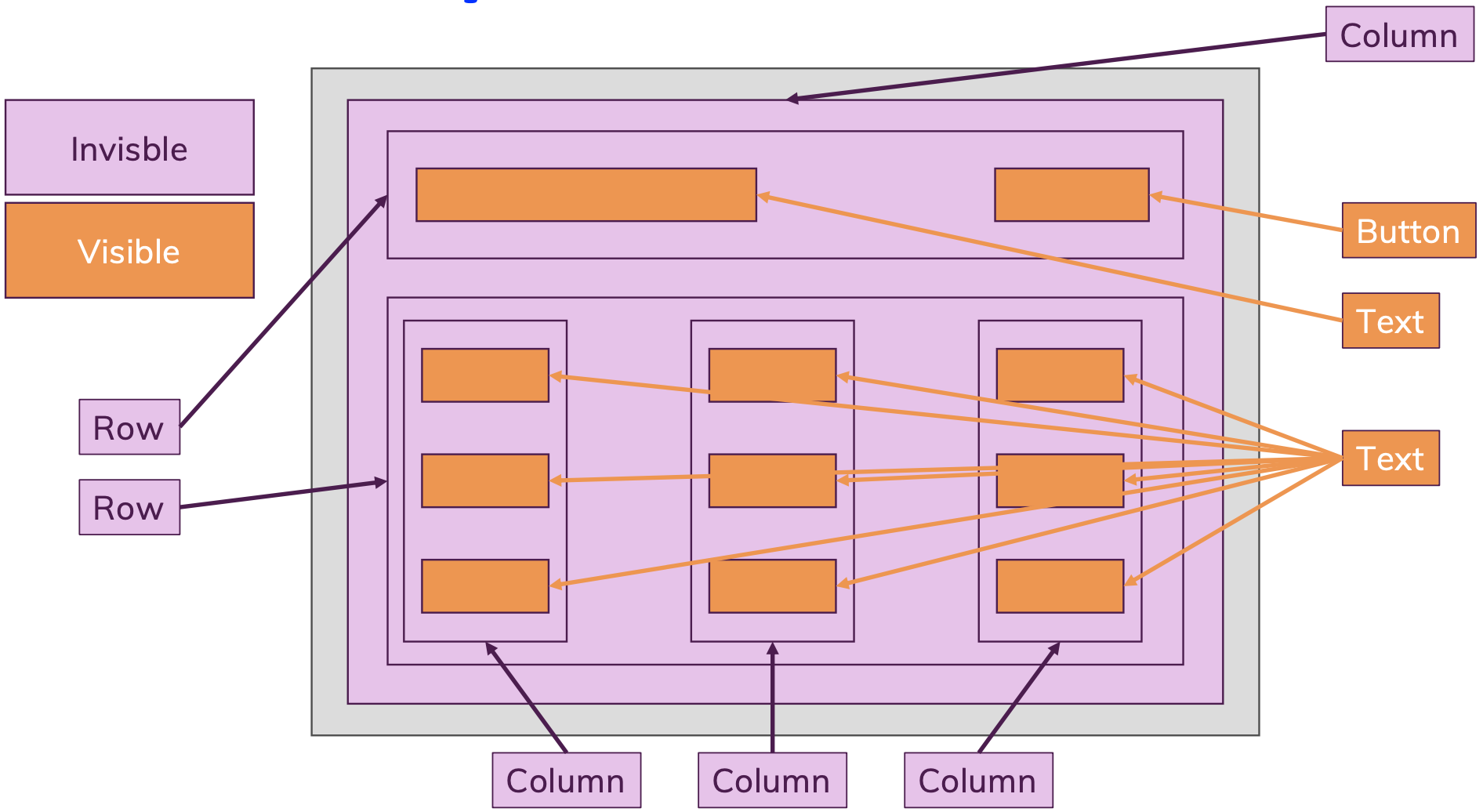
```
Run | Debug | Profile
void main() {
  runApp(MyApp());
}

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('First App'),
        ), // AppBar
        body: Text('Hello, this is my deafualt'),
      ), // Scaffold
    ); // MaterialApp
  }
}
```

# WIDGETS

| Output & Input (Visible) | Layout & Control (Invisible) |
| --- | --- |
| RaisedButton(), Text(), Card(), ... | Row(), Column(), ListView(), ... |
| Drawn onto the screen: “What the user sees” | Give your app structure and control how visible widgets are drawn onto the screen (incirectly visible) |

Container()

# VISIBLE/INVISIBLE WIDGETS



# Tutorial Link

- https://www.youtube.com/watch?v=x0uinJvhNxI
- https://dit.udemy.com/course/learn-flutter-dart-to-build-ios-android-apps/

# ADDING MORE WIDGETS

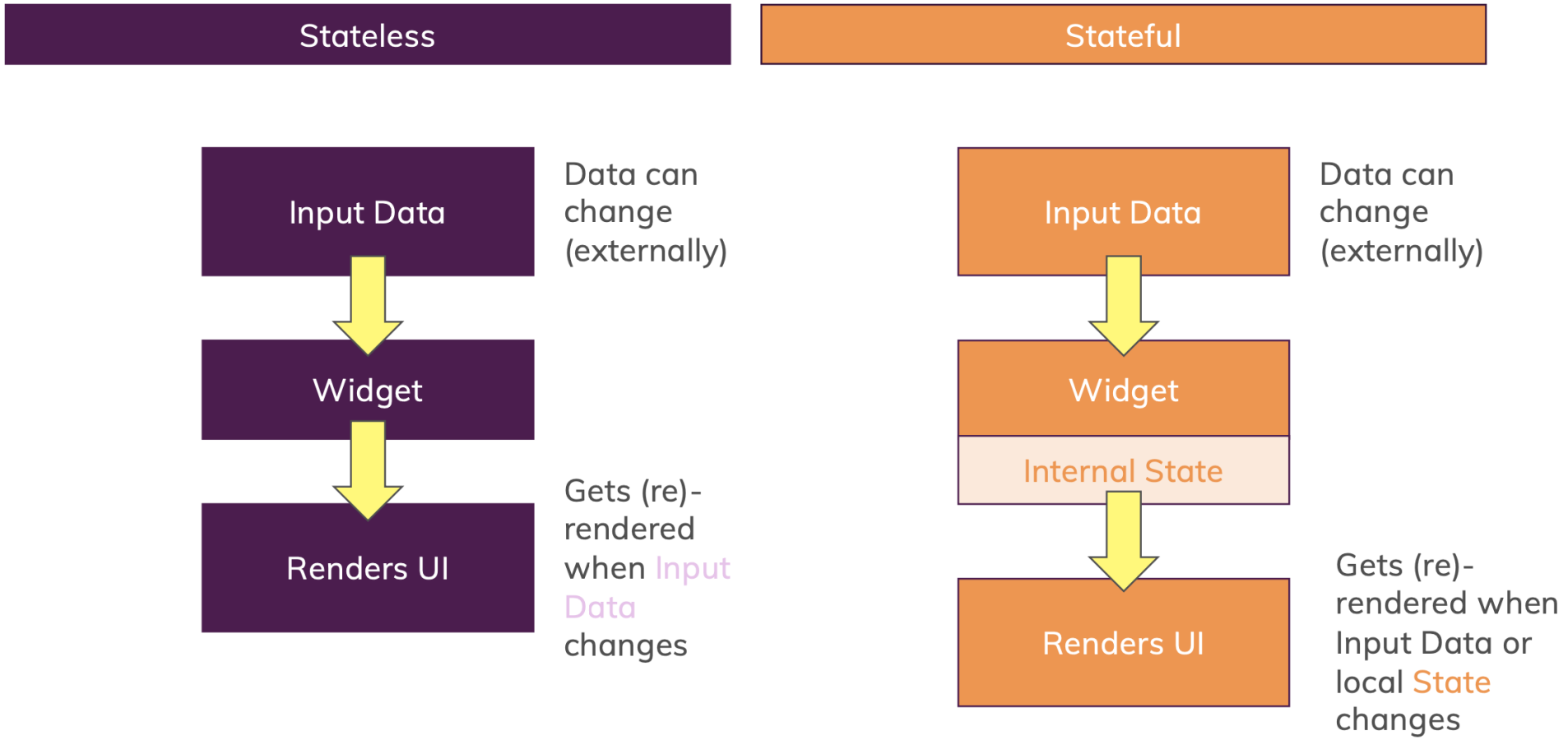
```dart
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    var questions = [
      'What is your name?',
      'Where are you from?',
    ];
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('First App'),
        ), // AppBar
        // body: Text('Hello, this is my deafualt'),
        body: Column(
          children: [Text('Questions'),
          RaisedButton(child: Text('Answer 1'),onPressed: null,),
          RaisedButton(child: Text('Answer 2'),onPressed: null,),
          RaisedButton(child: Text('Answer 3'),onPressed: null,),
          ],
        ), // Column
      ), // Scaffold
    ); // MaterialApp
  }
```

# RaisedButton deprecated

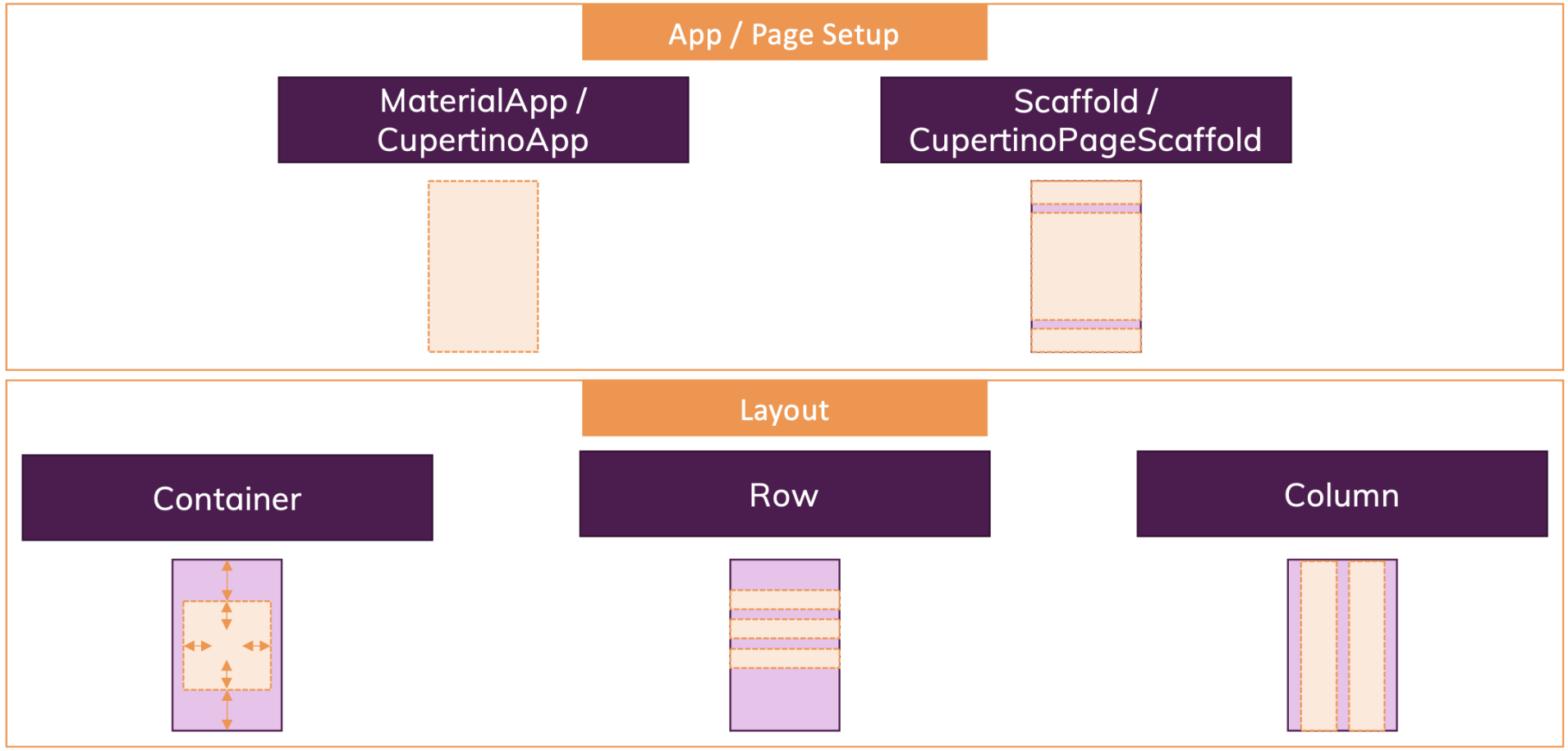
main.dart > MyApp > build

```dart
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    var questions = ['what is your name?', 'Where are you from?'];
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("Hello World"),
        ), // AppBar
        body: Column(
          children: [
            Text('Questions'),
            ElevatedButton(onPressed: null, child: Text('Answer1')),
            ElevatedButton(onPressed: null, child: Text('Answer2')),
            ElevatedButton(onPressed: null, child: Text('Answer3')),
          ],
        ), // Column
      ), // Scaffold
    ); // MaterialApp
  }
}
```

# STATELESS & STATEFUL



# Widgets


App / Page Setup
MaterialApp / CupertinoApp
Scaffold / CupertinoPageScaffold
Layout
Container
Row
Column

# Details

## Container

- Extremely versatile widget!
- Can be sized (width, height, maxWidth, maxHeight), styled (border, color, shape, ...) and more
- Can take a child (but doesn't have to) which you also can align in different ways
- You'll use this widget quite often

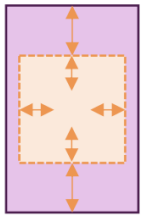
## Row / Column

- Must-use if you need multiple widgets sit next to each other horizontally or vertically
- Limited styling options => Wrap with a Container (or wrap child widgets) to apply styling
- Children can be aligned along main-axis and cross-axis (see separate cheat sheet)

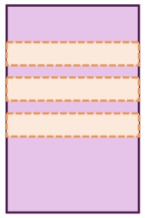
# Widgets

Layout

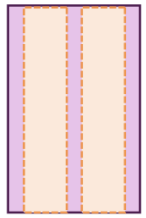
Container



Row

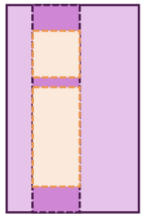


Column

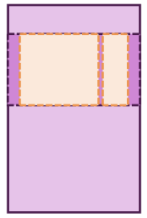


Row / Column Children

Flexible



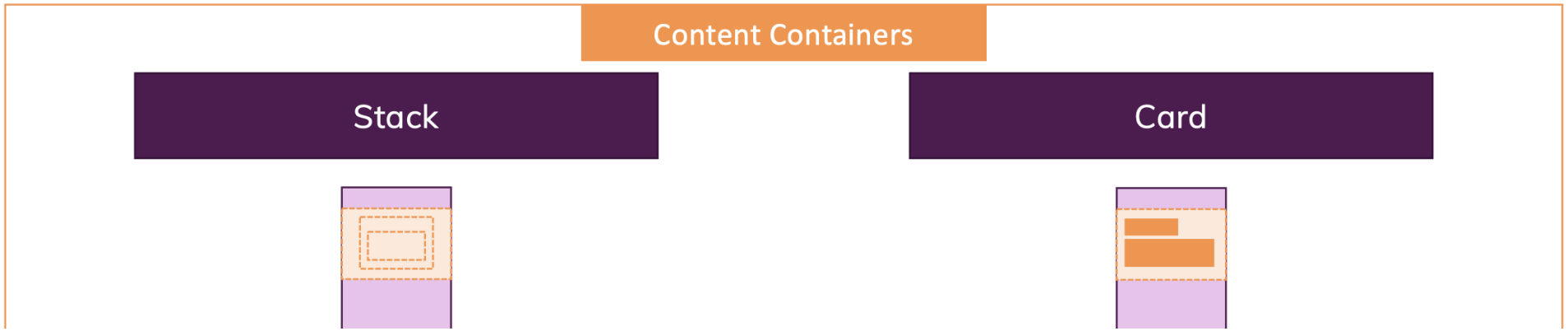
Expanded



Flexible / Expanded

- Also see separate cheat sheet attached to lecture about Row / Column / Flexible / Expanded
- Helps you size the child widgets of Row / Column

# Widgets



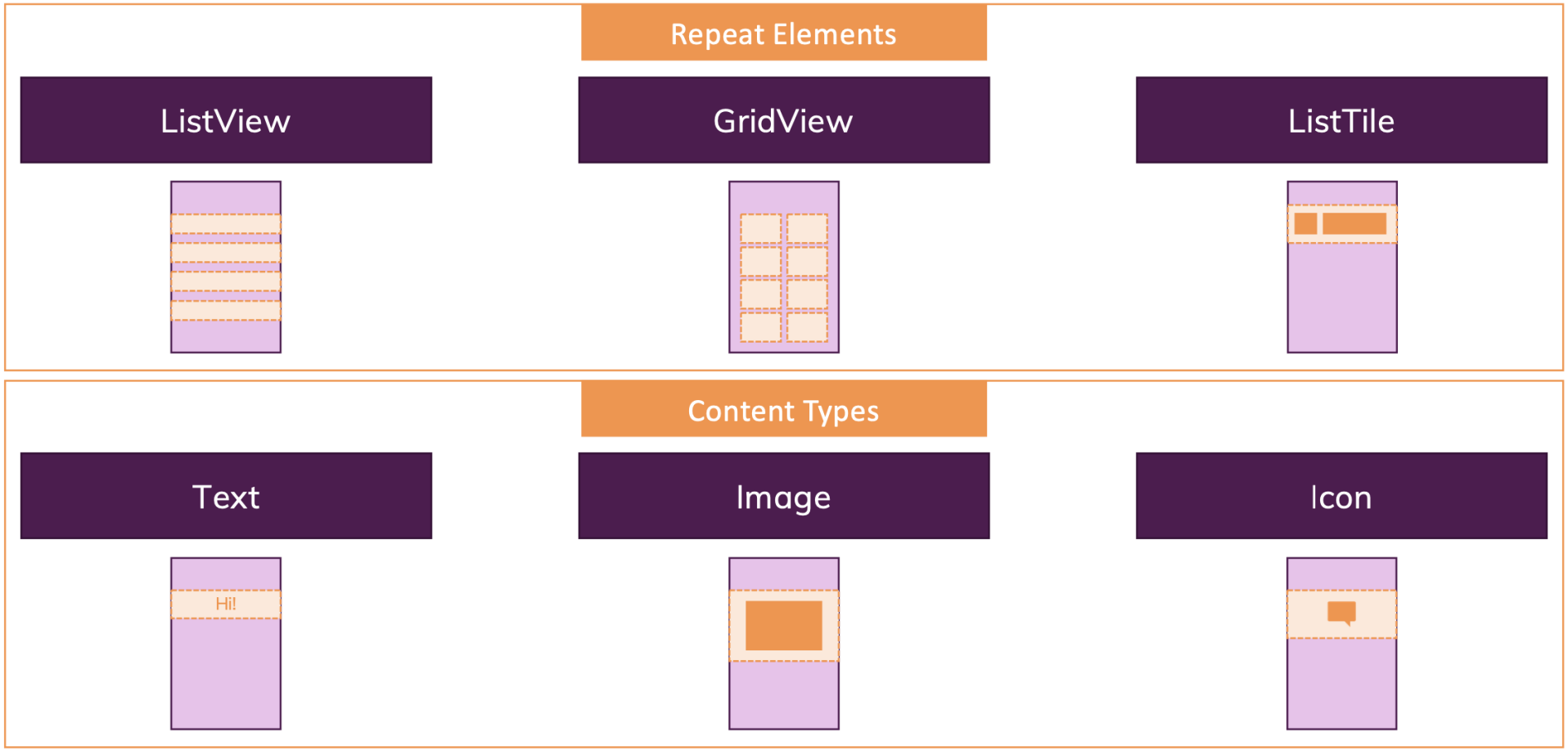

## Stack

- Used to position items on top of each other (along the Z axis)
- Widgets can overlap
- You can position items in absolute space (i.e. in a coordinate space) via the Positioned() widget

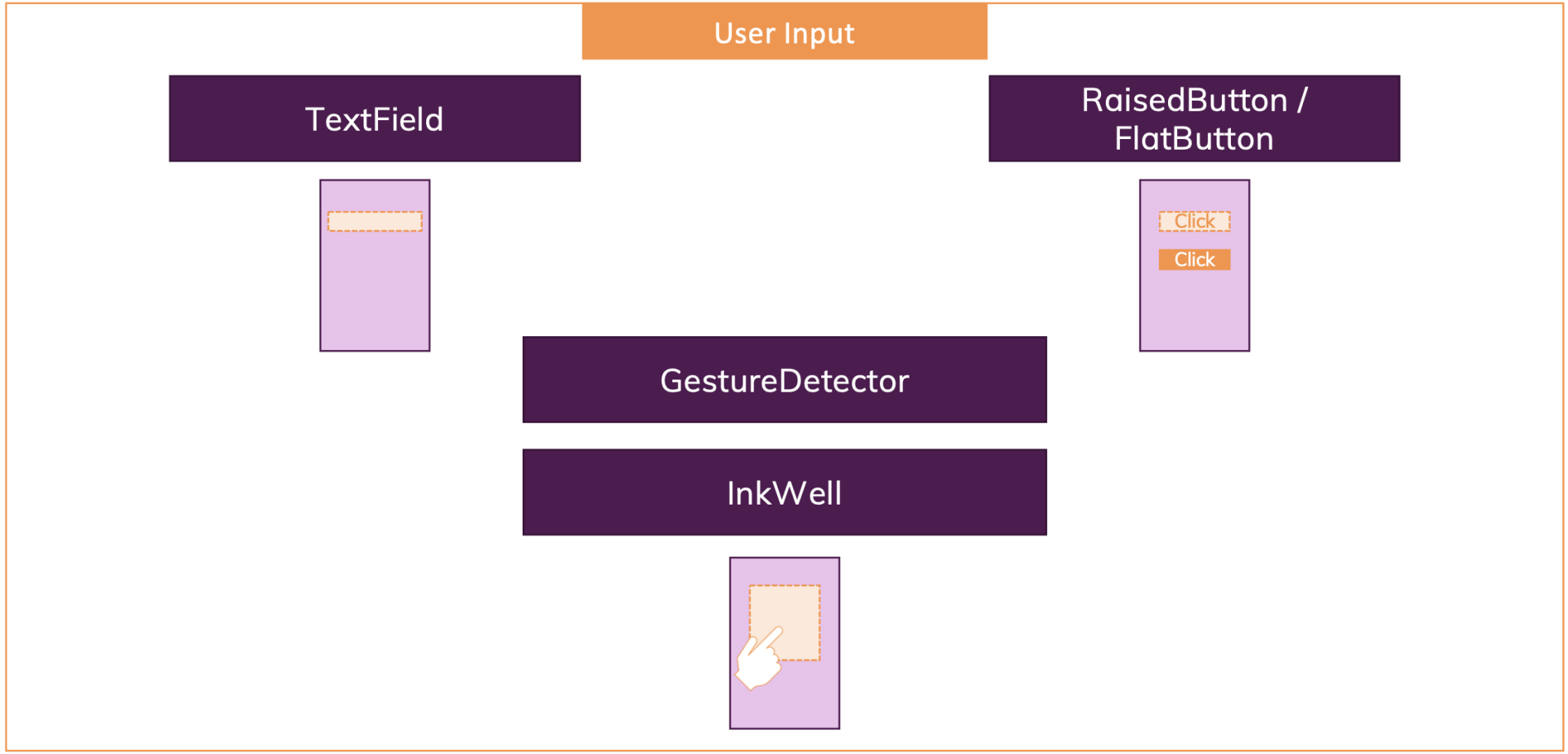
## Card

- A container with some default styling (shadow, background color, rounded corners)
- Can take one child (can be anything)
- Typically used to output a single piece / group of information

# WIDGETS


Repeat Elements
ListView
GridView
ListTile
Content Types
Text
Hi!
Image
Icon

# WIDGETS


User Input
TextField
RaisedButton /
FlatButton
Click
Click
GestureDetector
InkWell

- Data – Persistence
- Read-Write File
- SQLite

# Data - Persistence

## ❑ Store Key-Value on disk

- If you have a relatively small collection of key-values to save, you can use the shared_preferences plugin.
- The shared preferences plugin wraps NSUserDefaults on iOS and SharedPreferences on Android.
- This recipe uses the following steps:
  - Add the dependency
  - Save data
  - Read data
  - Remove data

# Data - Persistence

## ❑ Store Key-Value on disk

1. Add the dependency.

- Before starting, add the shared_preferences package as a dependency using flutter pub add:

- `flutter pub add shared_preferences`

# Data - Persistence

## ❑ Store Key-Value on disk

- **2. Save Data**
  - To persist data, use the setter methods provided by the SharedPreferences class. Setter methods are available for various primitive types, such as setInt, setBool, and setString.
  - synchronously update the key-value pair in-memory.

```
// obtain shared preferences
final prefs = await SharedPreferences.getInstance();

// set value
await prefs.setInt('counter', counter);
```

# Data - Persistence

## ❑ Store Key-Value on disk

- **3. Read Data**
  - To read data, use the appropriate getter method provided by the SharedPreferences class. For each setter there is a corresponding getter. For example, you can use the getInt, getBool, and getString methods.

```
final prefs = await SharedPreferences.getInstance();

// Try reading data from the counter key.
// If it doesn't exist, return 0.
final counter = prefs.getInt('counter') ?? 0;
```

# Data - Persistence

## ❑ Store Key-Value on disk

- **4. Remove Data**
  - To delete data, use the remove() method..

```
final prefs = await SharedPreferences.getInstance();
```

```
await prefs.remove('counter');
```

# Data - Persistence

- Read-Write Files
- Persist data with SQLite

# Beyond UI

- Data and Backend
- Platform Integration
- Packages and Plugins
- Testing and Debugging
- Performance and Optimization
- Deployment
- Add to an existing app

# Data and Backend

- ✓ The state management – level of app development stages
- ✓ Networking and HTTP:
  - ✓ Fetch data from the internet
  - ✓ Make authenticated requests
  - ✓ Send data to the internet
  - ✓ Perform CRUD of data over the internet
- ✓ Data Serialization
  - ✓ JSON Serialization

# Data and Backend

- ✓ Persistence – Date storage using key-value pair
  - ✓ How to add, read and remove data from the local disk using Shared Preferences concept.
- ✓ Read and Write data to files
- ✓ Persist data with SQLite
  - ✓ If you have large amounts of data on the local device, consider using a database. Databases provide faster CRUD functions.
- ✓ Firebase - Backend-as-a-Service (BaaS)
- ✓ Google APIs

# Platform Integration

- ✓ The Support Tiers
  - ✓ **Supported** - Google-tested platforms that are automatically tested on every commit by continuous integration testing.
  - ✓ **Best effort -** Platforms that we intend to support through coding practices, but are only tested on an ad-hoc basis.
  - ✓ **Unsupported** - Platforms that we don't test or support.
- ✓ How to write platform-specific code? There is an automatic platform adaptation as well.
- ✓ Apps for Android, iOS, Linux, macOS, Web and Windows

# Packages and Plugins

- ✓ Flutter supports using shared packages contributed by other developers to the Flutter and Dart ecosystems.
- ✓ This allows quickly building an app without having to develop everything from scratch.
- ✓ A plugin is a type of package—the full designation is plugin package, which is generally shortened to plugin.

# Testing and Debugging

- **Testing**:
  - ✓ The more features your app has, the harder it is to test manually.
  - ✓ Automated tests help ensure that your app performs correctly before you publish it, while retaining your feature and bug fix velocity.
  - ✓ Automated testing falls into a few categories:
    1. A ***unit test*** tests a single function, method, or class.
    2. A ***widget test*** (in other UI frameworks referred to as component test) tests a single widget.
    3. An ***integration test*** tests a complete app or a large part of an app.

# Testing and Debugging

- **Debugging**:
  - ✓ There's a wide variety of tools and features to help debug Flutter applications. Here are some of the available tools:
    - ✓ **DevTools**, a suite of performance and profiling tools that run in a browser.
    - ✓ **Android Studio/IntelliJ, and VS Code** (enabled with the Flutter and Dart plugins) support a built-in source-level debugger with the ability to set breakpoints, step through code, and examine values.
    - ✓ **Flutter inspector**, a widget inspector available in DevTools, and also directly from Android Studio and IntelliJ

# Performance and Optimization

- ✓ What is performance? Why is performance important? How do I improve performance?
- ✓ To improve performance, you first need metrics: some measurable numbers to verify the problems and improvements.
  - ✓ Startup time to the first frame
  - ✓ Statistics of frame
  - ✓ CPU/GPU usage (a good approximation for energy use)
- ✓ Speed of your app - Are your animations janky (not smooth)?
- ✓ App Size - How to measure your app's size. The smaller the size, the quicker it is to download.
- ✓ Rendering animations in your app is one of the most cited topics of interest when it comes to measuring performance.

# Thank you!