

Flutter Kya Hai — Pehli App Samjho

Flutter is Google's UI toolkit: one codebase, real apps on Android and iOS — and everything on screen is a widget.

Introduction

The Dart series taught you the language. Now the language gets a face. **Flutter** is Google's UI toolkit: a complete box of ready-made screen pieces — buttons, text, images, layouts — that you assemble with Dart code to build real apps for Android *and* iOS from one codebase. In this opener you learn Flutter's one big idea — **everything is a widget** — and read your first complete app, line by line. (*Roman Urdu: Flutter mein screen par nazar aane wali har cheez aik widget hai — text bhi, button bhi, poori screen bhi.*)

Key points

- Flutter apps are **written in Dart** — classes, lists, null safety, everything you just learned. Nothing is wasted; Flutter is where the Dart series pays off.
- One codebase, two app stores:** the same project builds a native-speed Android app and an iOS app. Flutter does not make a slow "web page in a frame" — it *draws* every pixel itself, which is why Flutter apps feel smooth (60–120 frames per second).
(*Roman Urdu: Aik dafa code likho, Android aur iPhone dono ki app tayyar.*)
- Everything is a widget.** A `Text` is a widget. A button is a widget. The whole screen is a widget. Even the app itself is a widget. Widgets are classes (Lesson D.8!) — you build objects from them and nest them inside each other, like boxes inside boxes.
- The family tree of a basic app: `MaterialApp` (the app) → `Scaffold` (the screen frame: top bar + body) → `AppBar` (title bar) and `body` (the middle). Most beginner apps are variations of this tree.
- `runApp(...)` starts the app and puts your first widget on the phone screen. `Center` is a widget that centres its **child**. Almost every widget takes settings through named parameters — the `{name = default}` pattern from Lesson D.6, now everywhere.
- To build Flutter apps you need three things (next lesson installs them): the **Flutter SDK**, an **editor** (VS Code), and a **phone to run on** — an emulator or your real phone.

Code examples

Example 1 — The smallest real Flutter app

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text('Assalam o Alaikum!'),
        ),
      ),
    ),
  );
}
```

What it does: Read it inside-out, like nested boxes: a `Text` sits inside a `Center`, which sits inside the `body` of a `Scaffold`, which is the `home` screen of a `MaterialApp`. The first line imports Flutter's material library — the box of widgets. `runApp` puts the whole tree on the phone.

What you will see: a plain screen with "Assalam o Alaikum!" exactly in the middle. It does not look exciting — but it is a genuine app running on a real device framework, and every app in this series grows from these same seven lines.

Example 2 — Add a title bar and style the text

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('My First App')),
        body: Center(
          child: Text(
            'Main Flutter seekh raha hoon',
            style: TextStyle(fontSize: 24),
          ),
        ),
      ),
    ),
  );
}
```

What it does: Two changes only. The `Scaffold` gains an `appBar` — an `AppBar` widget holding a title `Text`. And the body text gains a `style` — a `TextStyle` object (a class again!) that sets its size to 24. Notice the pattern: widgets take objects as settings.

What you will see: a coloured title bar at the top reading "My First App", and noticeably larger text in the centre. Change the title in the code, save, and the app updates in under a second — that instant update is **hot reload**, and Lesson F.2 makes it yours.

Example 3 — The widget tree, visible: several widgets nested

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Widget Tree')),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Icon(Icons.school, size: 60),
              Text(
                'Har cheez widget hai',
                style: TextStyle(fontSize: 20),
              ),
            ],
          ),
        ),
      ),
    );
}
```

What it does: A `Column` widget stacks its **children** (a list — D.7!) top to bottom: here an `Icon` (a school icon from Flutter's built-in set) above a `Text`. `mainAxisAlignment: MainAxisAlignment.center` centres the stack vertically. `Column` gets its own deep lesson in F.4; today, just see the tree grow.

What you will see: a school icon in the middle of the screen with the line "Har cheez widget hai" under it, under a "Widget Tree" title bar. Count the widgets on this screen: `MaterialApp`, `Scaffold`, `AppBar`, `Text` × 2, `Center`, `Column`, `Icon` — eight widgets, one screen. That counting habit is Flutter thinking.

Quick recap

- Flutter = Google's UI toolkit; apps are written in Dart; one codebase → Android + iOS.
- Everything on screen is a widget, and widgets are classes nested like boxes.

- Basic tree: `MaterialApp` → `Scaffold` → `AppBar` + `body`.
- `runApp()` starts the app; settings arrive as named parameters and small objects like `TextStyle`.

Common mistakes

1. **Forgetting the import line** — without `import 'package:flutter/material.dart';`, every widget name is unknown and the file fills with errors. It must be the first line of every Flutter file.
2. **Reading the tree outside-in and getting lost** — beginners drown in brackets. Read inside-out instead: find the innermost widget (the content), then ask "what holds it?" step by step outward. The indentation is your map — keep it tidy.
3. **Expecting to design by dragging** — Flutter has preview tools, but the real skill is writing the tree in code. Coming from Dart, you already have the skill; the widgets are just new vocabulary.

Practice

1. Take Example 2 and change the app completely by editing only values: a new title, your own name as the centre text, font size 30. Run it after every single change and watch hot reload (or re-run) each time.
2. In Example 3, add a third child to the Column's list — a second `Text` with your class name — and predict the screen before you run it. Then swap the two Text widgets' order in the list and see what changes.

Watch the video lesson: <https://youtu.be/bYMTvB45Ld0>

Read online: <https://toolscope.netlify.app/notes/flutter-what-is-flutter> · More free tools and notes at <https://toolscope.netlify.app>