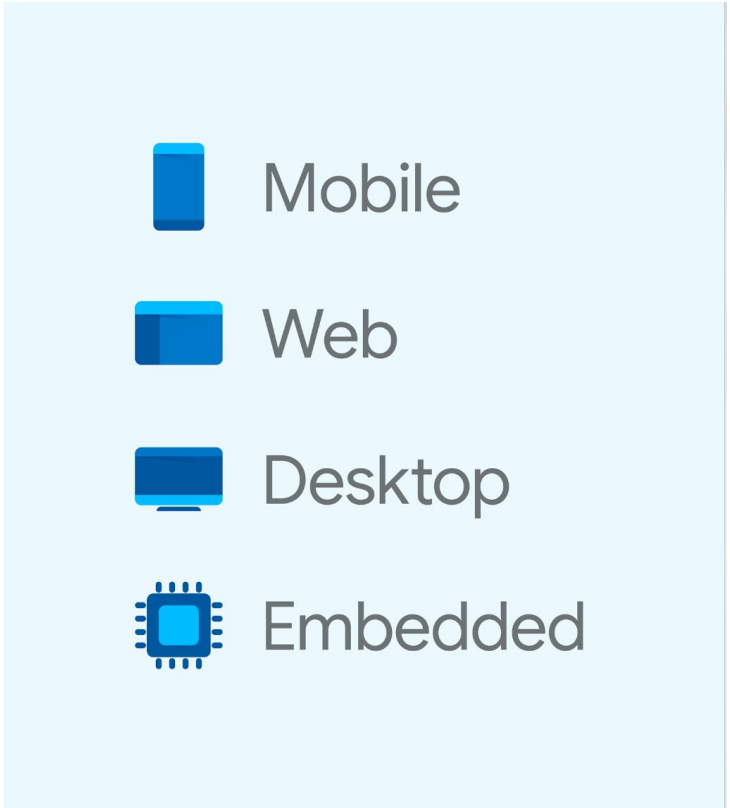

MOBILE PROGRAMMING
WITH FLUTTER - DAY2
เริ่มพัฒนาแอปพลิเคชันบนมือถือด้วย Flutter


Flutter


Mobile
Web
Desktop
Embedded

“บน macOS เราสามารถสร้างแอปทีเดียว
ให้ใช้ได้ทั้ง iOS, Android ได้เลย”

“แต่สำหรับ Windows และ Linux
จะทำได้เฉพาะฝั่ง Android เท่านั้น”

📌 สิ่งที่เราต้องเตรียมสำหรับ Flutter

- **Flutter SDK**
  - Flutter SDK สำหรับจัดการ Project Flutter
  - Git สำหรับใช้งาน Flutter SDK
- **Platform Tools**
  - Android Studio สำหรับพัฒนา บน Android
  - XCode สำหรับใช้พัฒนาบน iOS
- **Virtual Device**
  - Android สำหรับ Preview App
  - iOS สำหรับ Preview App

Install Flutter using VS Code

Learn how to install and set up Flutter in a Code OSS-based editor. This includes (but is not limited to), VS Code, Cursor, and Windsurf.

> **Tip**
>
> If you've never set up or developed an app with Flutter before, follow Set up and test drive Flutter instead.

Choose your development platform

The instructions on this page are configured to cover installing Flutter on a **macOS** device.

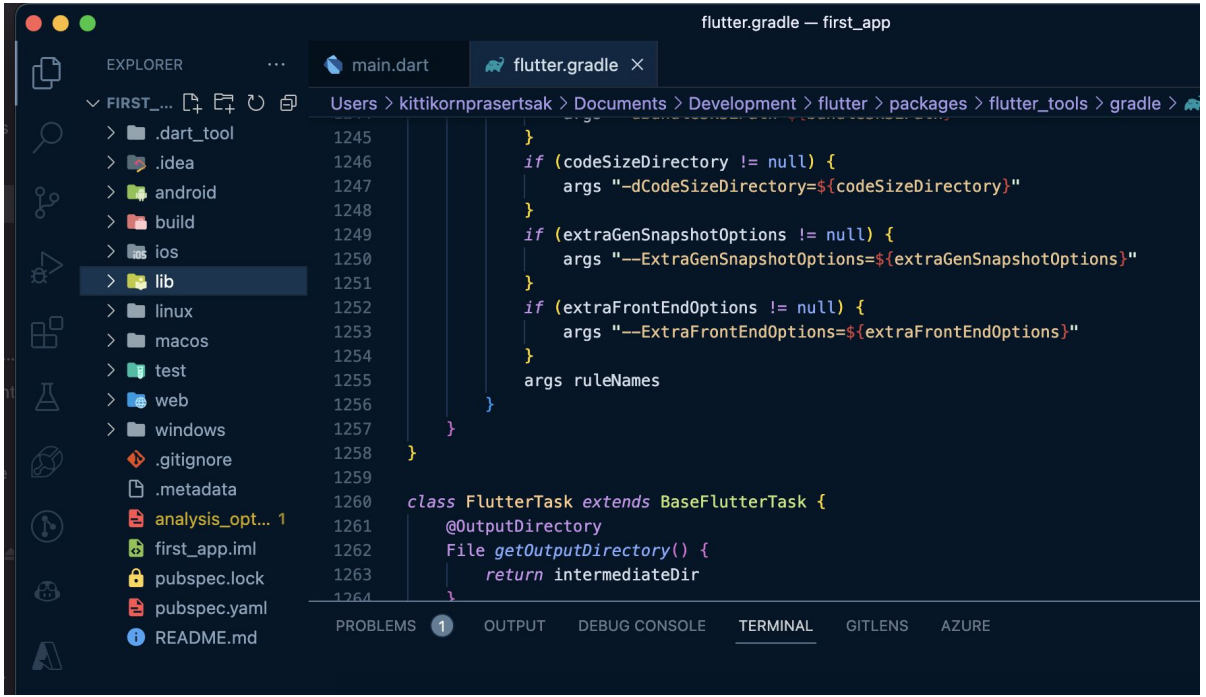
If you'd like to follow the instructions for a different OS, please select one of the following.

Windows | macOS | Linux | ChromeOS

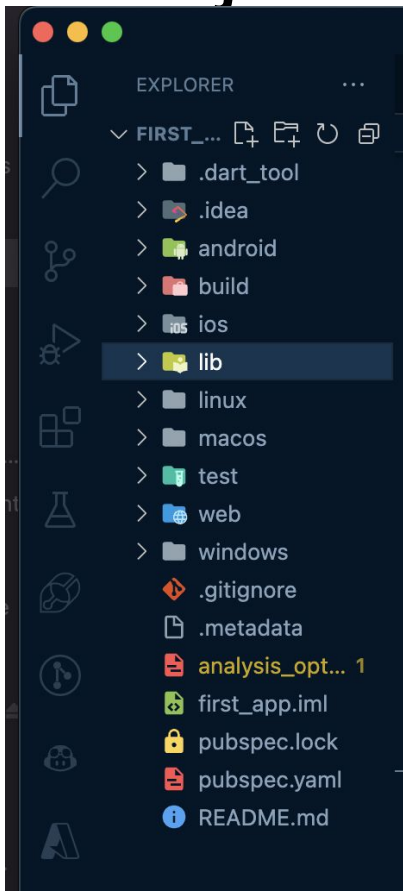
Download prerequisite software

⭐ Flutter Project Structure

⭐ Flutter Project Structure

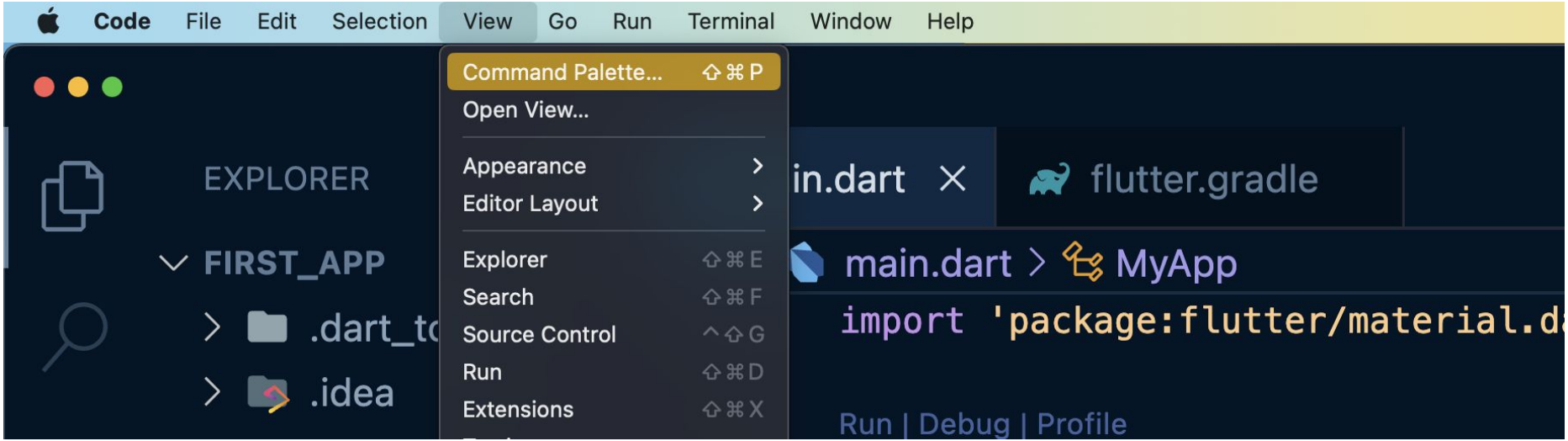


⭐ Flutter Project Structure



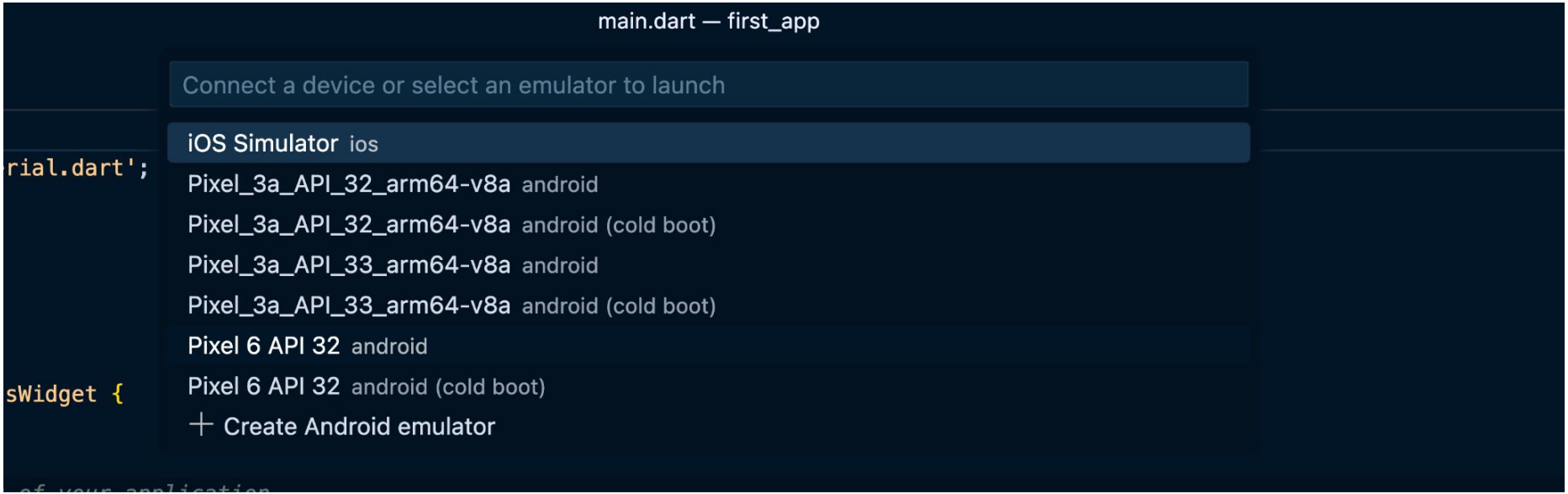
⭐ เวลาเขียน และ สร้างไฟล์หลักที่ /lib/main.dart

เมื่อต้องการเปิด Emulator



เมื่อต้องการเปิด Emulator

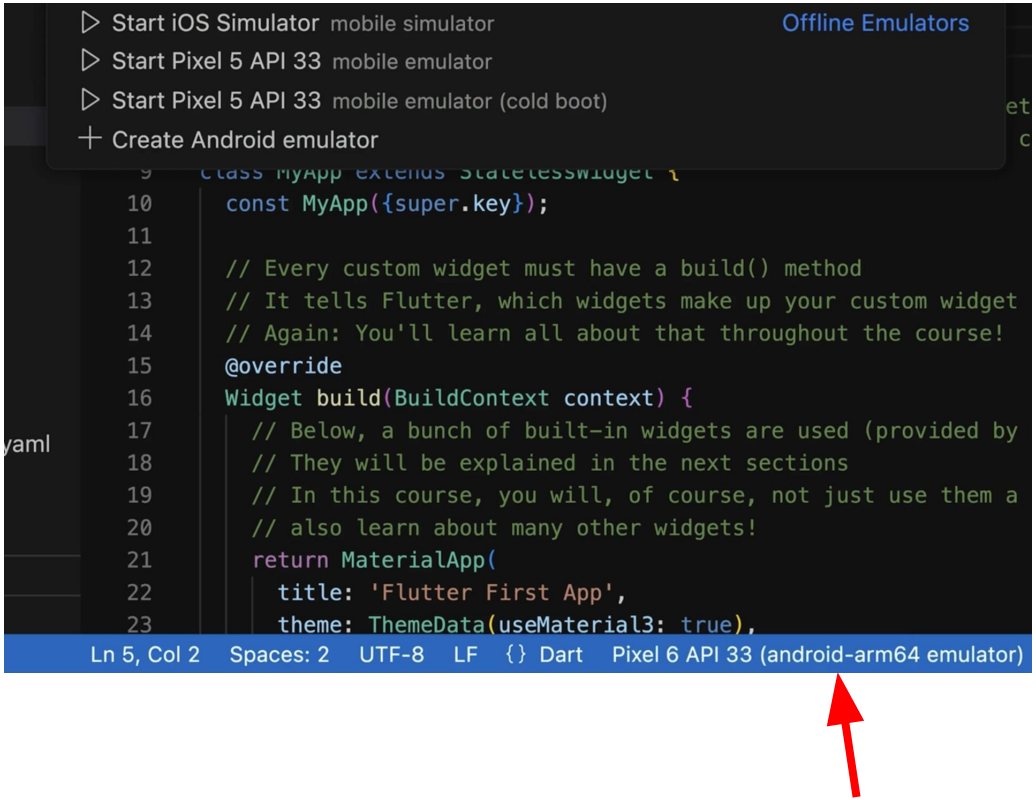
เลือก Emulator ที่ต้องการ



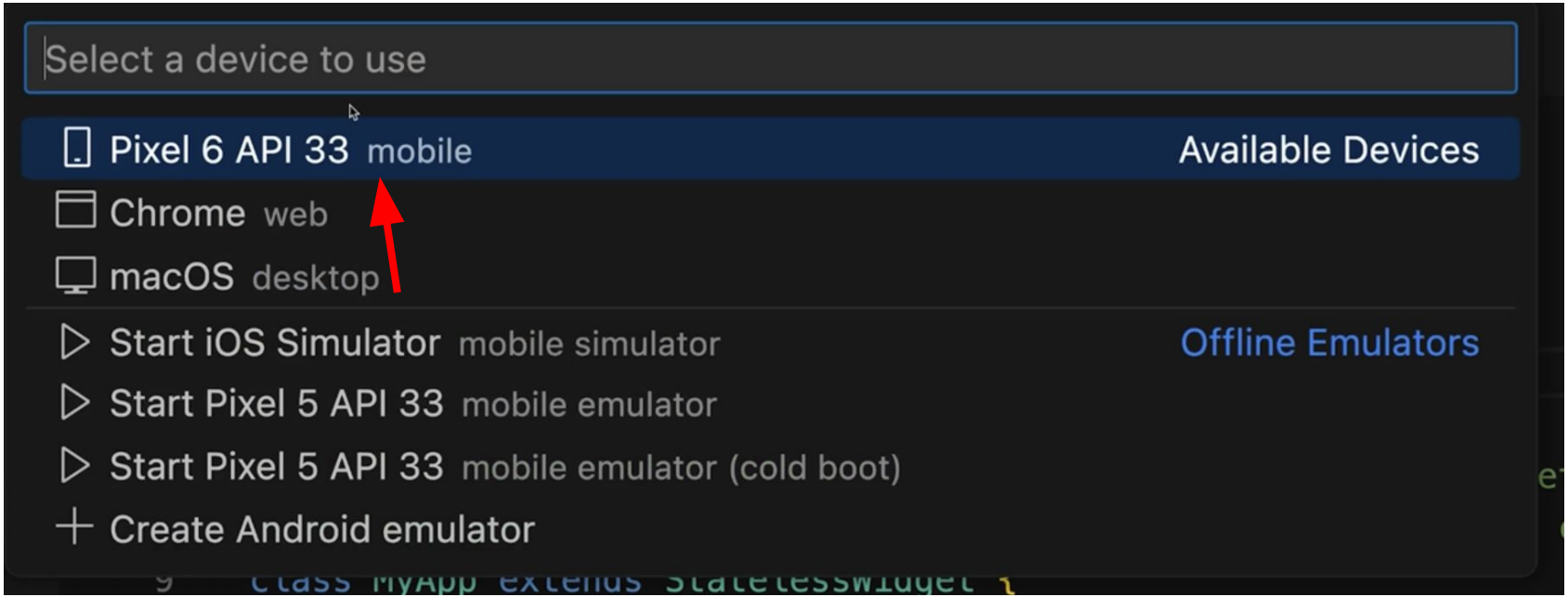
รอสักครู่ Emulator ก็จะออกมา

ตรวจสอบก่อนรัน App

1. คลิกขวาล่าง เลือกอุปกรณ์ที่ชื่ออุปกรณ์

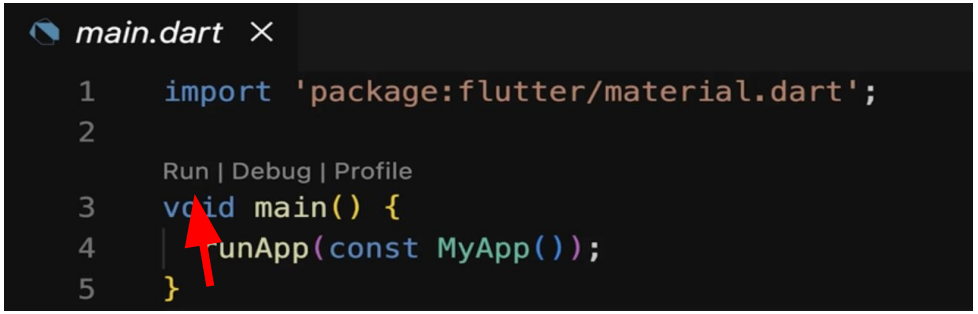


2. เลือกอุปกรณ์ที่ชื่ออุปกรณ์



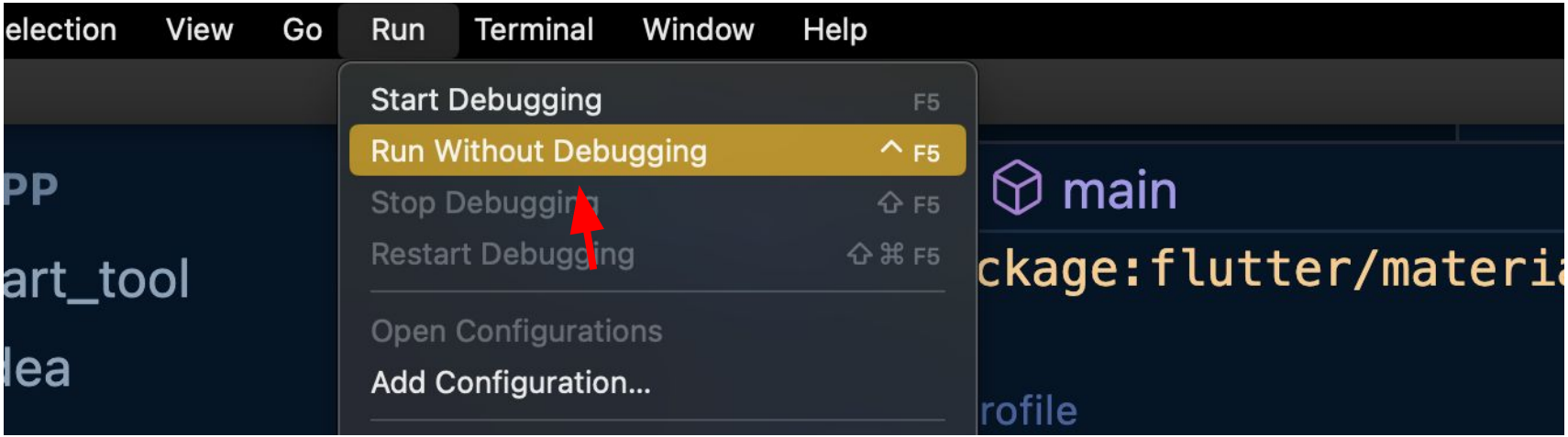
วิธีการทดสอบแอป ทำได้หลายวิธี

1. ผ่าน Run ตรง main.dart



2. ผ่าน Terminal พิมพ์ “flutter run”

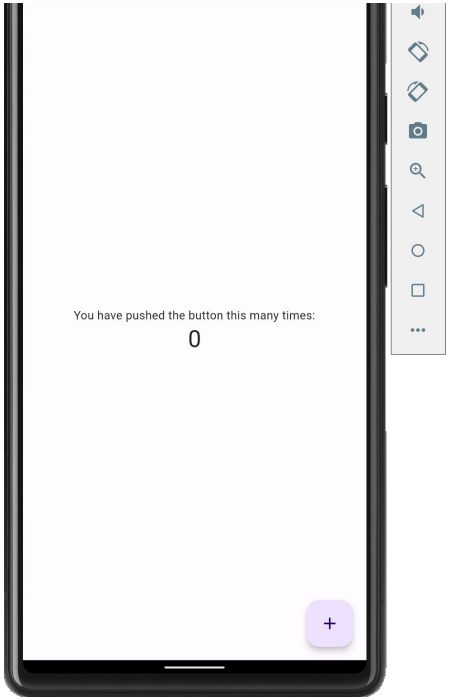
3. ผ่าน Run ตรง Run



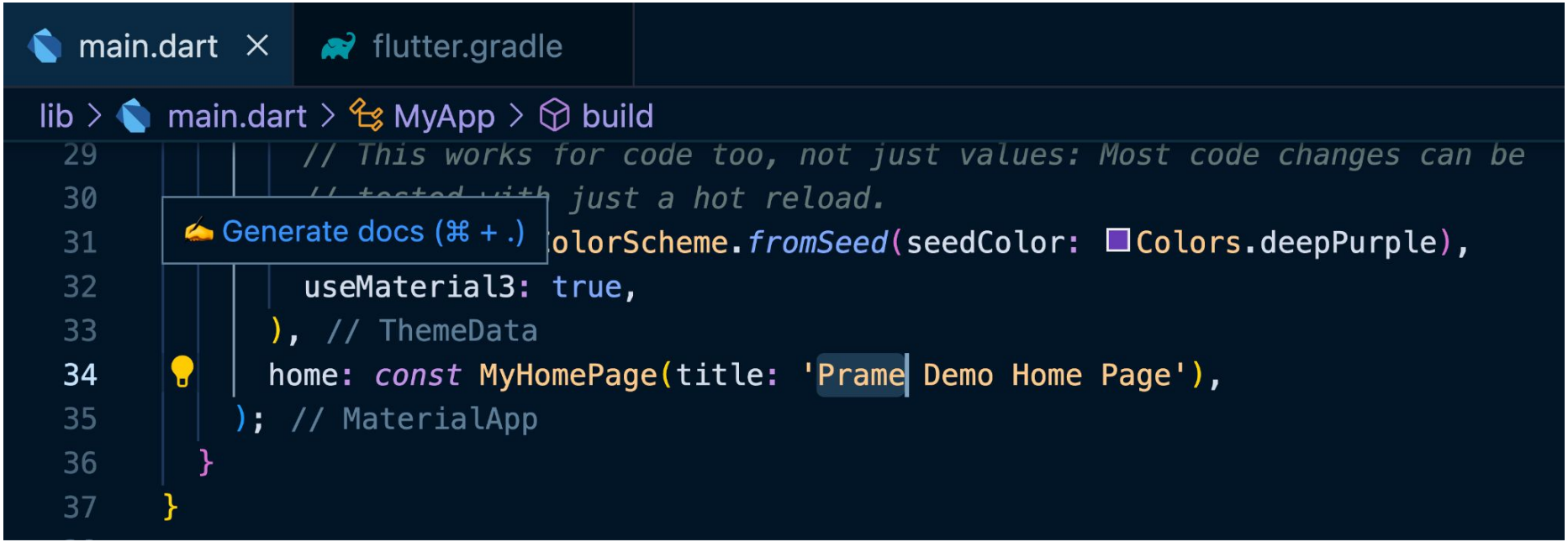
เมื่อได้เรียบร้อยจะมีหน้าข้อความประมาณนี้

```
Kittikorns-MBP:first_app kittikornprasertsak$ sudo flutter run
Password:
   Woah! You appear to be trying to run flutter as root.
   We strongly recommend running the flutter tool without superuser privileges.
  /
 /
Using hardware rendering with device sdk gphone64 arm64. If you notice graphics artifacts, consider enabling software rendering with "--enable-software-rendering".
Launching lib/main.dart on sdk gphone64 arm64 in debug mode...
Running Gradle task 'assembleDebug'...                              6.4s
✓  Built build/app/outputs/flutter-apk/app-debug.apk.
Syncing files to device sdk gphone64 arm64...                       102ms

Flutter run key commands.
r Hot reload. 🔥🔥🔥
R Hot restart.
h List all available interactive commands.
d Detach (terminate "flutter run" but leave application running).
c Clear the screen
q Quit (terminate the application on the device).

A Dart VM Service on sdk gphone64 arm64 is available at: http://127.0.0.1:58746/sJlW9VjEDT8=/
The Flutter DevTools debugger and profiler on sdk gphone64 arm64 is available at: http://127.0.0.1:9101?uri=http://127.0.0.1:58746/sJlW9VjEDT8=/
```

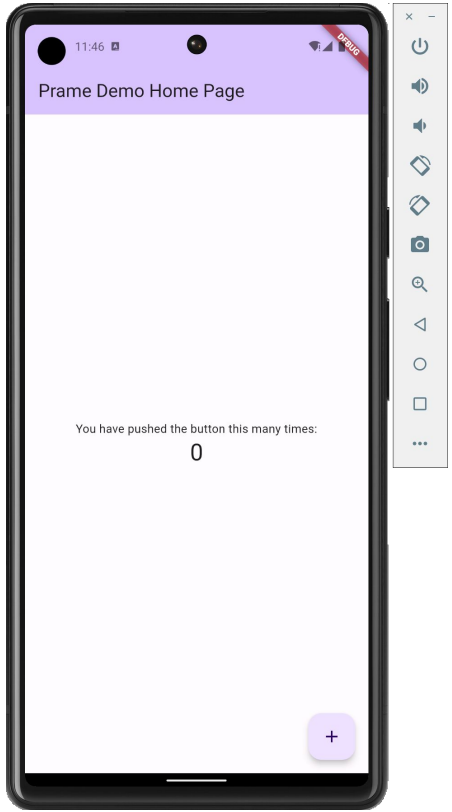
ผลลัพธ์ที่ได้



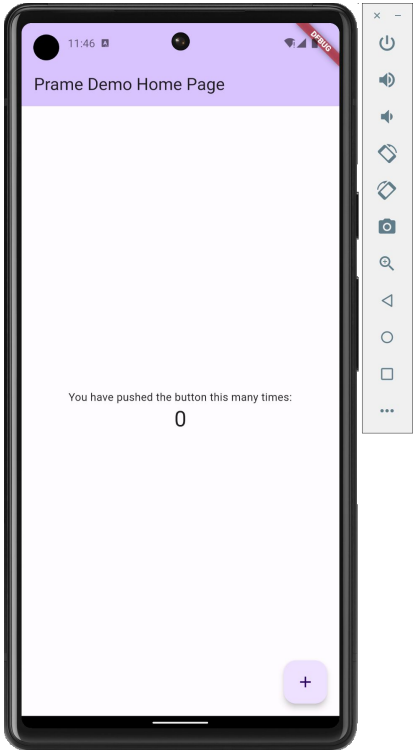
ทดสอบโดยการเปลี่ยนข้อมูลใน Title กัน

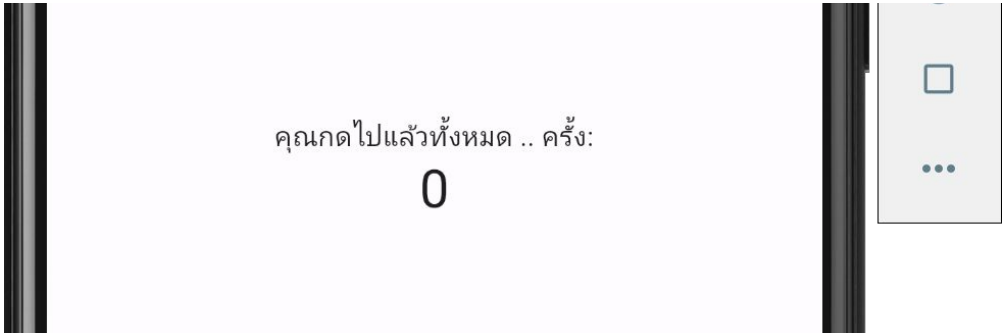


ผลลัพธ์ที่ได้


11:46
DEBUG
Prame Demo Home Page
You have pushed the button this many times:
0
+

Quiz : รบกวนแก้ข้อความเป็นแบบนี้ให้หน่อย





Quiz : แก้ภายใน Text ของ Widget

main.dart × flutter.gradle

lib > main.dart > _MyHomePageState > build

```
105            // wireframe for each widget.
106            mainAxisAlignment: MainAxisAlignment.center,
107            children: <Widget>[
108              const Text(
109                'คุณกดไปแล้วทั้งหมด .. ครั้ง:',
110              ), // Text
```

Imperative Programming vs Declarative Programming


borntoDev
หลักสูตรคุณภาพสูง
Dev Lab 3
บทความ
สำหรับภาคธุรกิจ
อีด
บัญชีของฉัน
เข้าระบบเรียน (ใหม่!)
2
Computer System
Programming Concept
Imperative VS Declarative Programming ต่างกันยังไง ?
BORNTO DEV
Developer Team
BorntoDev Co., Ltd.
Love
5
Share
Tweet
รู้มั้ยว่าที่จริงๆแล้วพื้นฐานการเขียนโค้ดมีรูปแบบหลักๆอยู่นะ ซึ่งรูปแบบเหล่านี้ไม่ว่าจะเป็นการเขียนโค้ดสไตล์ไหนก็สามารถแบ่งได้เป็น 2 อย่าง นั่นคือ Imperative และ Declarative หลายๆคนก็อาจจะเคยได้ยินผ่านหูผ่านตาเวลาเสิร์ชหาข้อมูลในอินเทอร์เน็ตมาบ้าง แต่เคยนึกสงสัยมั้ยว่า 2 แบบนี้มันคืออะไร มันง่ายมากๆเลย ลองไปเริ่มจากนิยามสั้นๆกันก่อน

คำนิยามแบบสั้นๆ ง่ายๆ

Imperative – คือ “ทำ**ยังไง**ให้ได้สิ่งนี้”

Declarative – คือ “ทำ**อะไร**”

Imperative Programming

```
let result = [];
for (let i = 0; i < numbers.length; i++) {
  if (numbers[i] > 10) {
    result.push(numbers[i]);
  }
}
```

Declarative Programming

```
let result = numbers.filter(n => n > 10);
```

Imperative กับการสร้าง UI

```
// Create a new unordered list
var ul = $("<ul>");

// Create three list items and add them to the unordered list
var li1 = $("<li>").text("Item 1");
var li2 = $("<li>").text("Item 2");
var li3 = $("<li>").text("Item 3");

ul.append(li1);
ul.append(li2);
ul.append(li3);

// Add the unordered list to the body of the document
$("body").append(ul);
```

Declarative กับการสร้าง UI

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Declarative Example',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Declarative Example'),
        ),
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

Introduction to widgets

Flutter widgets are built using a modern framework that takes inspiration from React. The central idea is that you build your UI out of widgets. Widgets describe what their view should look like given their current configuration and state. When a widget's state changes, the widget rebuilds its description, which the framework diffs against the previous description in order to determine the minimal changes needed in the underlying render tree to transition from one state to the next.

Note: If you would like to become better acquainted with Flutter by diving into some code, check out basic layout codelab, building layouts, and adding interactivity to your Flutter app.

Widgets

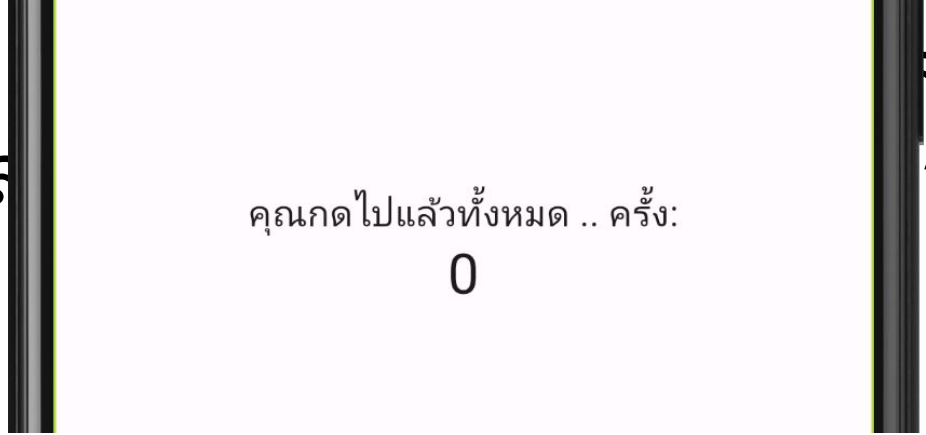
In regard to Flutter, you'll often hear "everything is a widget". Widgets are the building blocks of a Flutter app's user interface, and each widget is an immutable declaration of part of the user interface. Widgets are used to describe all aspects of a user interface, including physical aspects such as text and buttons to lay out effects like padding and alignment.

Widgets form a hierarchy based on composition. Each widget nests inside its parent and can receive context from the parent. This structure carries all the way up to the root widget, as this trivial example shows:

จากการเขียนรูปแบบดังกล่าว เราสามารถกำหนดข้อมูลอื่น ๆ ลงไปได้

```
106          mainAxisAlignment: MainAxisAlignment.center,
107          children: <Widget>[
108            const Text(
109              'คุณกดไปแล้วทั้งหมด .. ครั้ง:',
110              style: TextStyle(
111                fontSize: 18,
112              ), // TextStyle
```

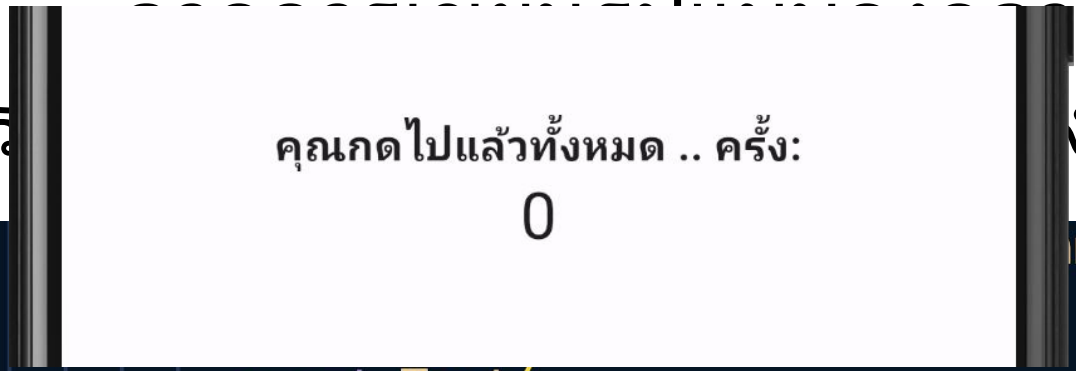
เราส ปได้




```
106          mainAxisAlignment: MainAxisAlignment.center,
107          children: <Widget>[
108            const Text(
109              'คุณกดไปแล้วทั้งหมด .. ครั้ง:',
110              style: TextStyle(
111                fontSize: 18,
112              ), // TextStyle
```

จากการเขียนรูปแบบดังกล่าว เราสามารถกำหนดข้อมูลอื่น ๆ ลงไปได้

```
106            mainAxisAlignment: MainAxisAlignment.center,
107            children: <Widget>[
108              const Text(
109                'คุณกดไปแล้วทั้งหมด .. ครั้ง:',
110                style: TextStyle(
111                  fontSize: 18,
112                  fontWeight: FontWeight.bold,
113                ), // TextStyle
114              ), // Text
```

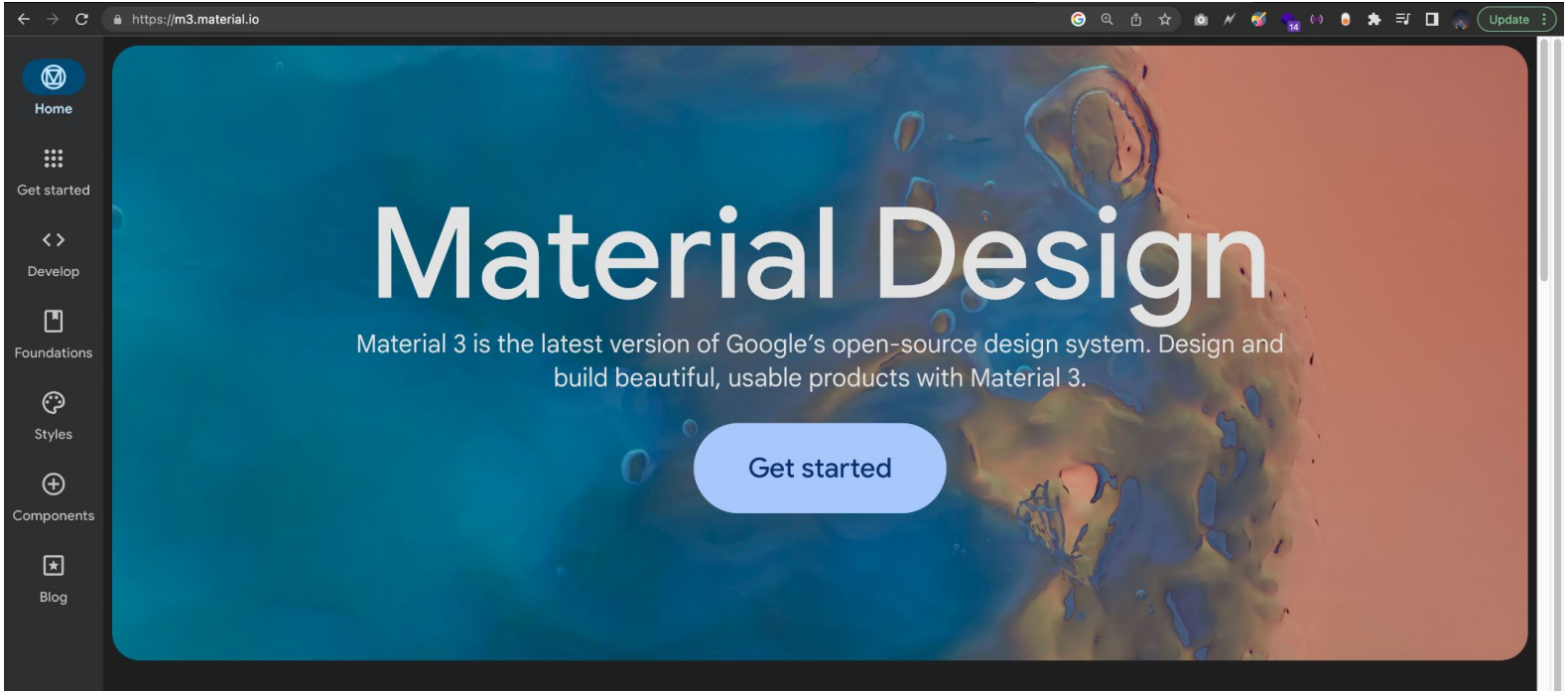



```
106                                                    enter,
107
108             const Text(
109               'คุณกดไปแล้วทั้งหมด .. ครั้ง:',
110               style: TextStyle(
111                 fontSize: 18,
112                 fontWeight: FontWeight.bold,
113               ), // TextStyle
114             ), // Text
```

การออกแบบสำหรับ Flutter

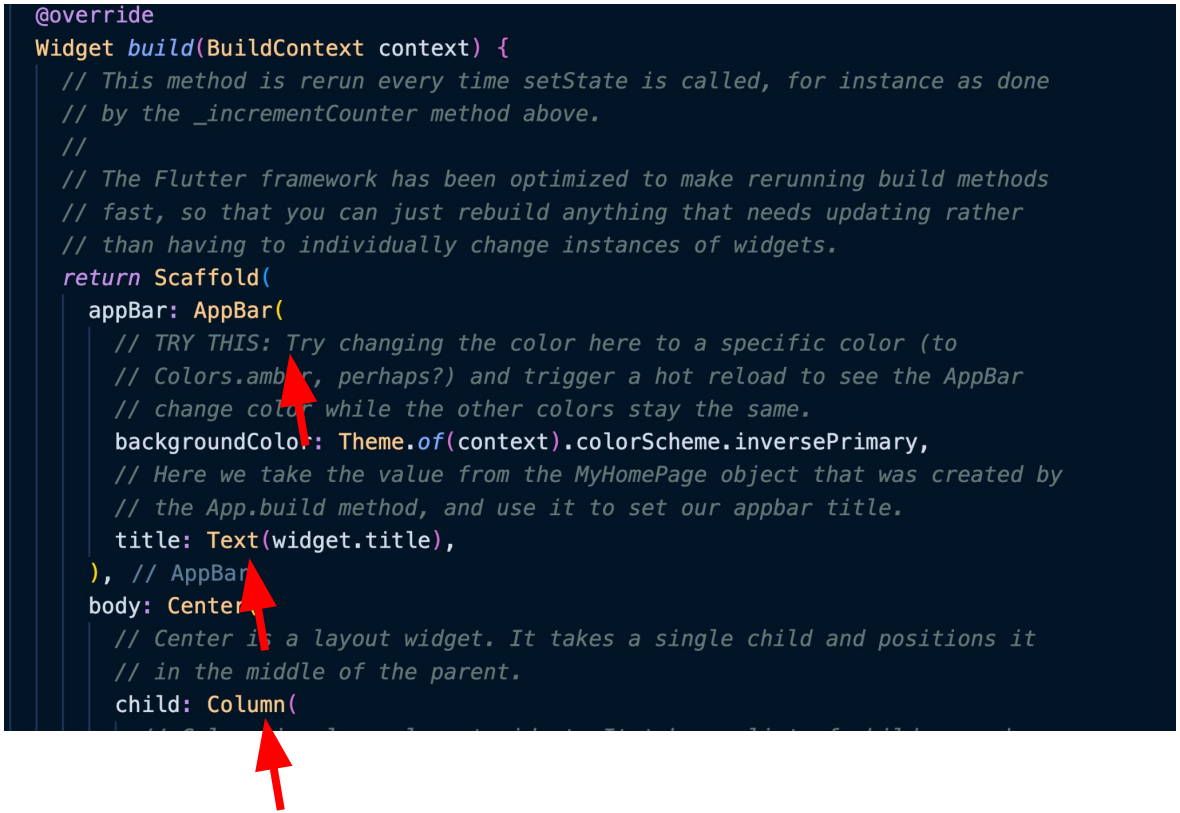
การออกแบบสำหรับ Flutter

การออกแบบสำหรับ Flutter



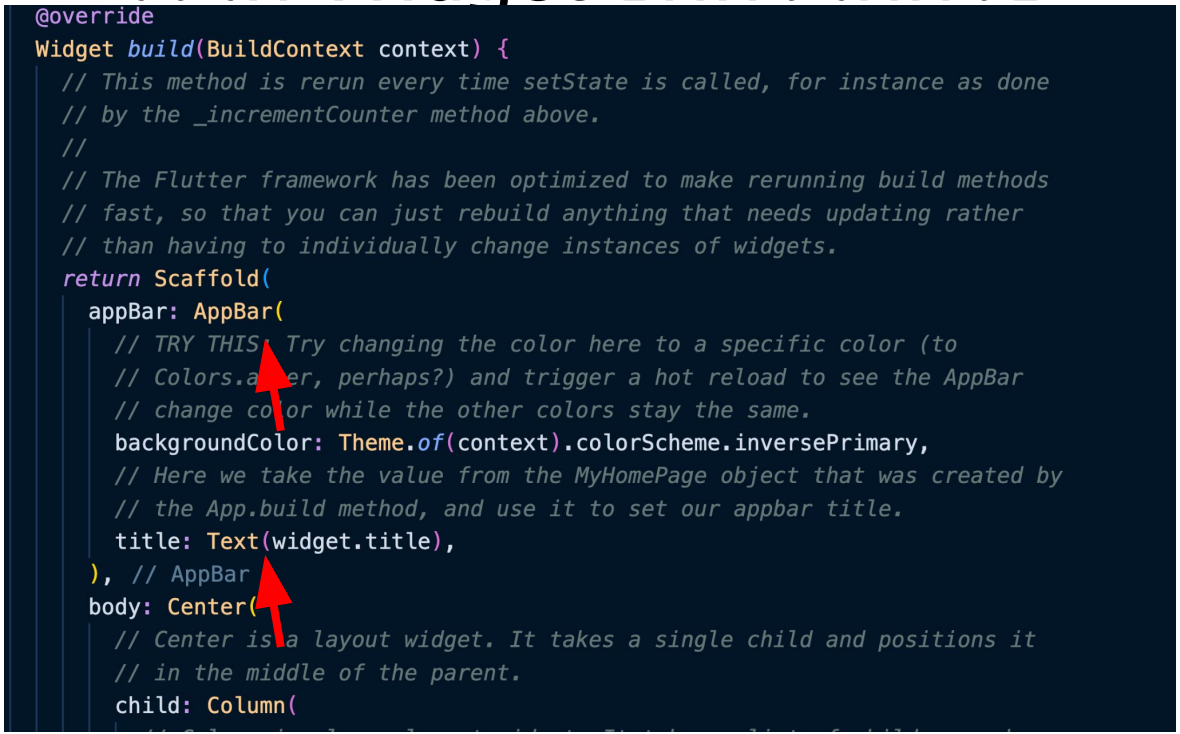
Flutter UI with Widget

Flutter UI with Widget



```dart
@override
Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build methods
  // fast, so that you can just rebuild anything that needs updating rather
  // than having to individually change instances of widgets.
  return Scaffold(
    appBar: AppBar(
      // TRY THIS: Try changing the color here to a specific color (to
      // Colors.amber, perhaps?) and trigger a hot reload to see the AppBar
      // change color while the other colors stay the same.
      backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      // Here we take the value from the MyHomePage object that was created by
      // the App.build method, and use it to set our appbar title.
      title: Text(widget.title),
    ), // AppBar
    body: Center(
      // Center is a layout widget. It takes a single child and positions it
      // in the middle of the parent.
      child: Column(
```

สังเกตได้ว่า เราสามารถรวม Widget ไว้ใต้ Widget อีกตัวได้ด้วย



```
@override
Widget build(BuildContext context) {
  // This method is rerun every time setState is called, for instance as done
  // by the _incrementCounter method above.
  //
  // The Flutter framework has been optimized to make rerunning build methods
  // fast, so that you can just rebuild anything that needs updating rather
  // than having to individually change instances of widgets.
  return Scaffold(
    appBar: AppBar(
      // TRY THIS: Try changing the color here to a specific color (to
      // Colors.amber, perhaps?) and trigger a hot reload to see the AppBar
      // change color while the other colors stay the same.
      backgroundColor: Theme.of(context).colorScheme.inversePrimary,
      // Here we take the value from the MyHomePage object that was created by
      // the App.build method, and use it to set our appbar title.
      title: Text(widget.title),
    ), // AppBar
    body: Center(
      // Center is a layout widget. It takes a single child and positions it
      // in the middle of the parent.
      child: Column(
```

สังเกตได้ว่า เราสามารถรวม Widget ไว้ใต้ Widget อีกตัวได้ด้วย

Hello world

The minimal Flutter app simply calls the `runApp()` function with a widget:

สังเกตได้ว่า เราสามารถรวม Widget ไว้ใต้ Widget อีกตัวได้ด้วย

Basic widgets

Flutter comes with a suite of powerful basic widgets, of which the following are commonly used:

`Text`
The `Text` widget lets you create a run of styled text within your application.

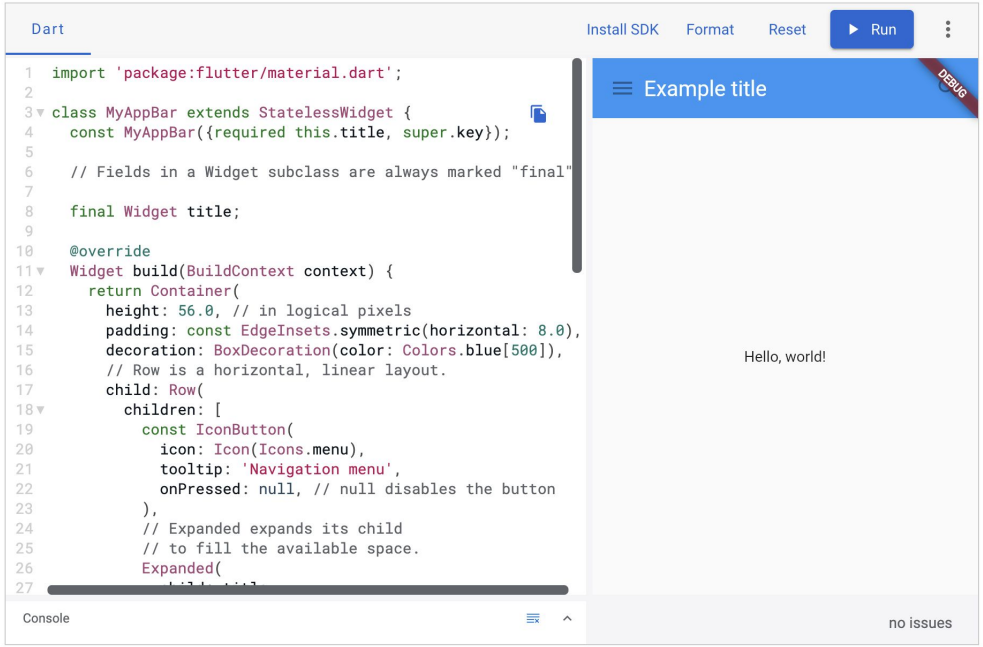
`Row`, **`Column`**
These flex widgets let you create flexible layouts in both the horizontal (`Row`) and vertical (`Column`) directions. The design of these objects is based on the web's flexbox layout model.

`Stack`
Instead of being linearly oriented (either horizontally or vertically), a `Stack` widget lets you place widgets on top of each other in paint order. You can then use the `Positioned` widget on children of a `Stack` to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web's absolute positioning layout model.

`Container`
The `Container` widget lets you create a rectangular visual element. A container can be decorated with a `BoxDecoration`, such as a background, a border, or a shadow. A `Container` can also have margins, padding, and constraints applied to its size. In addition, a `Container` can be transformed in three dimensional space using a matrix.

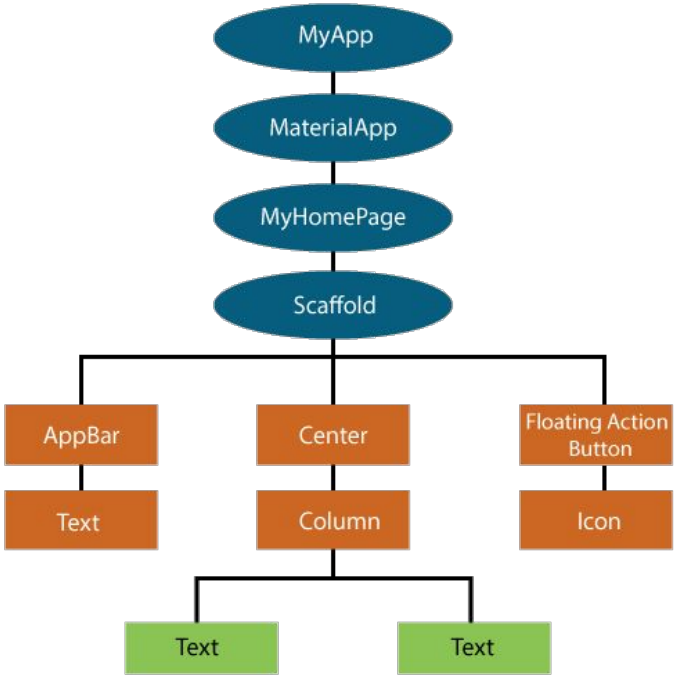
Below are some simple widgets that combine these and other widgets:

สังเกตได้ว่า เราสามารถรวม Widget ไว้ใต้ Widget อีกตัวได้ด้วย



เราจะรู้ได้อย่างไร ว่าอะไรอยู่ใต้อะไรบ้าง ?

Widget Tree


MyApp
MaterialApp
MyHomePage
Scaffold
AppBar
Center
Floating Action Button
Text
Column
Icon
Text
Text

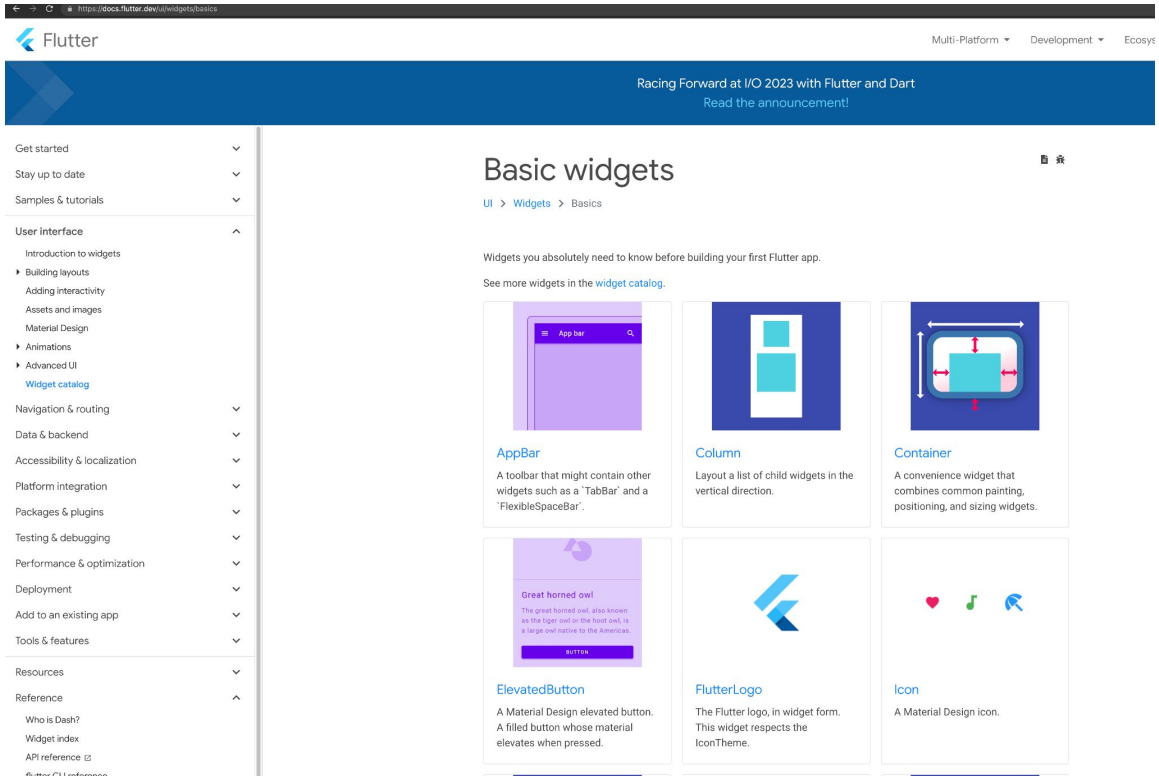
มาดูภายใน Widget กัน

OutlinedButton

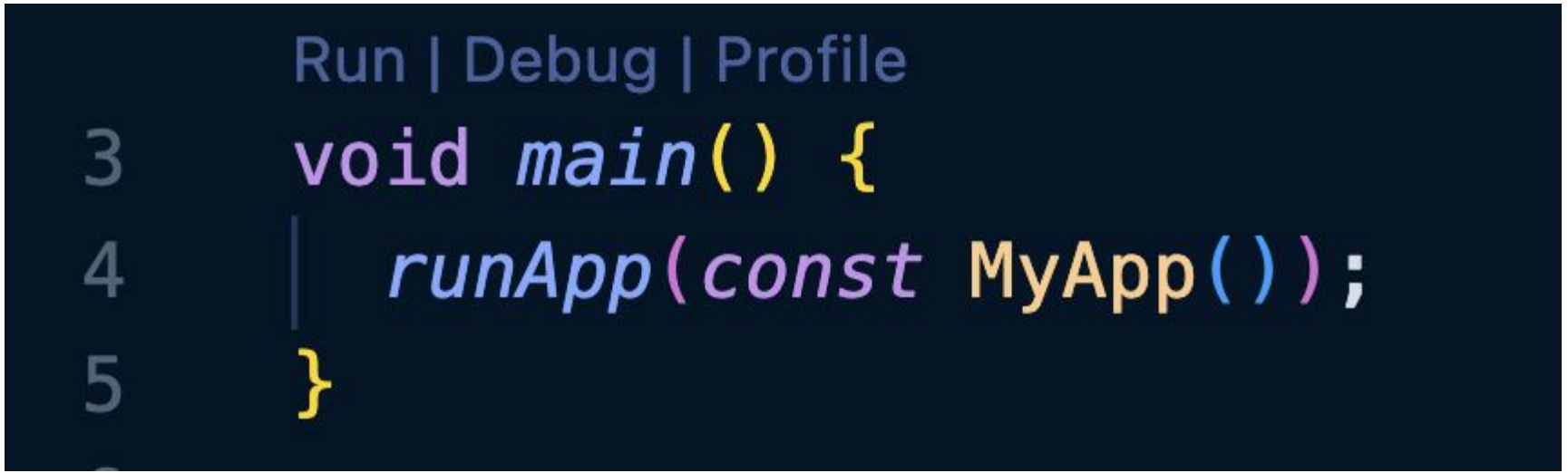
Text

Our Text Show :)

แล้วมี Widget อะไรให้เล่นบ้าง ?



คำถาม .. แล้ว MyApp ตัวนี้คือ Widget ไหมนะ ?



ลองดูการทำงานด้านในจะพบว่า

```
1078  void runApp(Widget app) {      Hixie, 8 years ago • fn3: Binding to RenderView …
1079    final WidgetsBinding binding = WidgetsFlutterBinding.ensureInitialized();
1080    assert(binding.debugCheckZone('runApp'));
1081    binding
1082      ..scheduleAttachRootWidget(binding.wrapWithDefaultView(app))
1083      ..scheduleWarmUpFrame();
1084  }
```

“Everything is a Widget”

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp( // Root widget
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My Home Page'),
        ),
        body: Center(
          child: Builder(
            builder: (context) {
              return Column(
                children: [
                  const Text('Hello, World!'),
                  const SizedBox(height: 20),
                  ElevatedButton(
                    onPressed: () {
                      print('Click!');
                    },
                    child: const Text('A button'),
                  ),
```

```dart
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp( // Root widget
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My Home Page'),
        ),
        body: Center(
          child: Builder(
            builder: (context) {
              return Column(
                children: [
                  const Text('Hello, World!'),
                  const SizedBox(height: 20),
                  ElevatedButton(
                    onPressed: () {
                      print('Click!');
                    },
                    child: const Text('A button'),
```

```dart
class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp( // Root widget
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My Home Page'),
        ),
        body: Center(
          child: Builder(
            builder: (context) {
              return Column(
                children: [
                  const Text('Hello, World!'),
                  const SizedBox(height: 20),
                  ElevatedButton(
                    onPressed: () {
                      print('Click!');
                    },
                    child: const Text('A button'),
```

- widget ลูก = อยู่ข้างใน widget แม่ (parent)
- widget ลูกสามารถ “รับ context” จาก parent
  (เช่น theme, ขนาดจอ ฯลฯ)
- Tree จะไล่ขึ้นไปจนถึง **root widget** (ตัวบนสุดของแอป)

มาลองเพิ่ม Widget อื่น ๆ กัน

การเพิ่ม OutlinedButton

```
119          OutlinedButton(onPressed: () { print("Hello World"); },
120            child:
121            Text("Hello,")), // OutlinedButton
122          ], // <Widget>[]
123        ), // Column
124      ), // Center
```

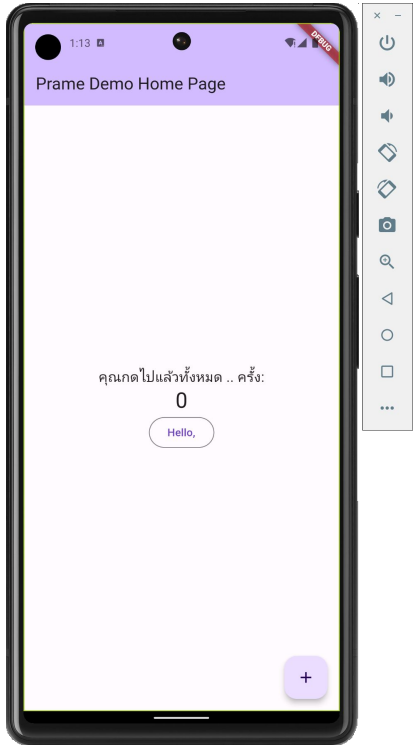
สังเกตได้ว่าเราสามารถโยนฟังก์ชันลงไป หรือเขียนแยกก็ได้

```
119          OutlinedButton(onPressed: () { print("Hello World"); },
120            child:
121            Text("Hello,")), // OutlinedButton
122          ], // <Widget>[]
123        ), // Column
124      ), // Center
```

เรียบร้อยลองกด r (hot reload)

```
119            OutlinedButton(onPressed: () { print("Hello World"); },
120              child:
121              Text("Hello,")), // OutlinedButton
122          ], // <Widget>[]
123        ), // Column
124      ), // Center
```

ลองคลิกปุ่มที่งอกมาใหม่



ผลลัพธ์ที่ได้

```
119          OutlinedButton(onPressed: () { print("Hello World"); },
120            child:
121              Text("Hello,")), // OutlinedButton
122          ] // <Widget>[]
123
124
```

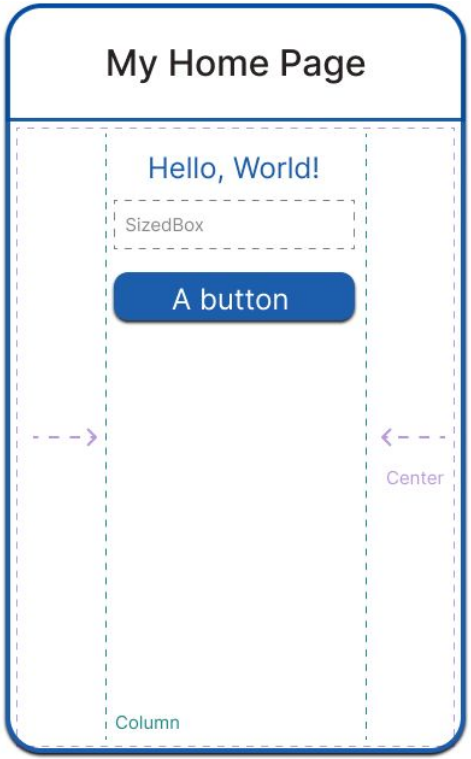
```
Performing hot reload...
Reloaded 1 of 665 libraries in 426ms (compile: 59 ms, reload: 235 ms, reassemble: 77 ms).
D/EGL_emulation( 8451): app_time_stats: avg=1945.56ms min=1.83ms max=67952.20ms count=35
D/EGL_emulation( 8451): app_time_stats: avg=1218.16ms min=1218.16ms max=1218.16ms count=1
D/EGL_emulation( 8451): app_time_stats: avg=1375.24ms min=1375.24ms max=1375.24ms count=1
I/flutter ( 8451): Hello World
D/EGL_emulation( 8451): app_time_stats: avg=111.26ms min=8.16ms max=3495.72ms count=37
I/flutter ( 8451): Hello World
I/flutter ( 8451): Hello World
```

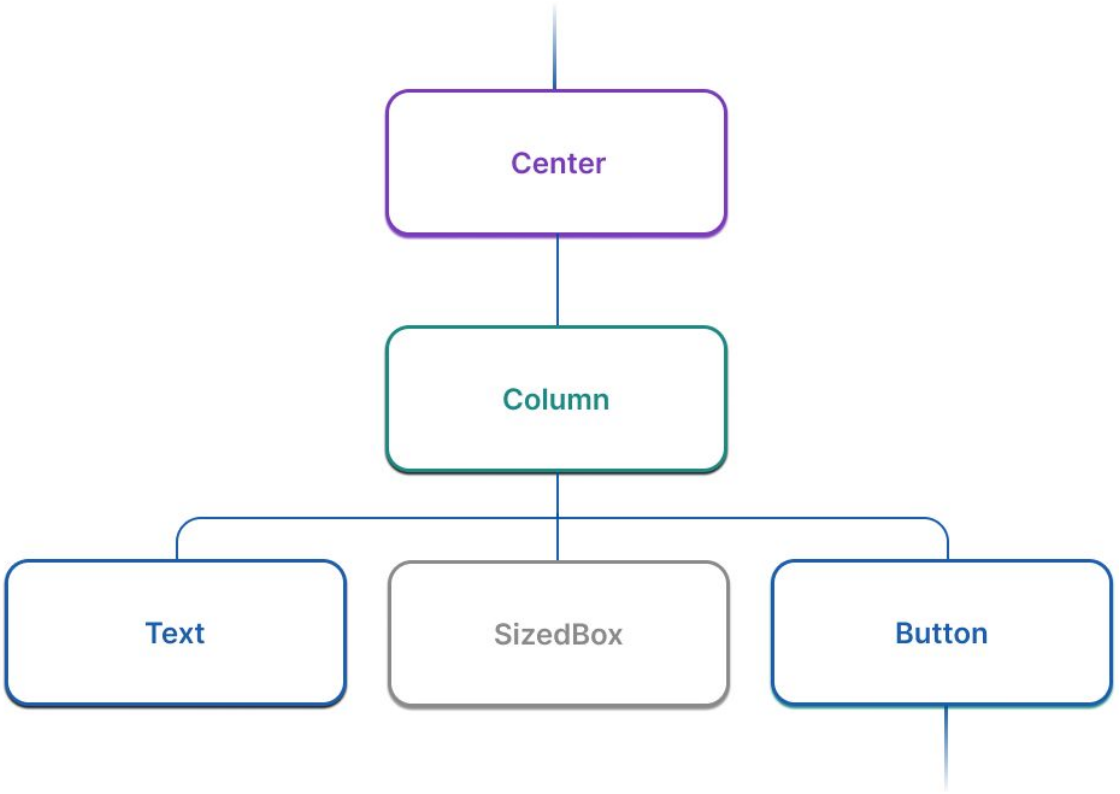
```
    style: Theme.of(context).textTheme.headlineMedium,
  ), // Text
  OutlinedButton(onPressed: () {print("Hello World2");}, child: const Text("Press Me"))
],
```

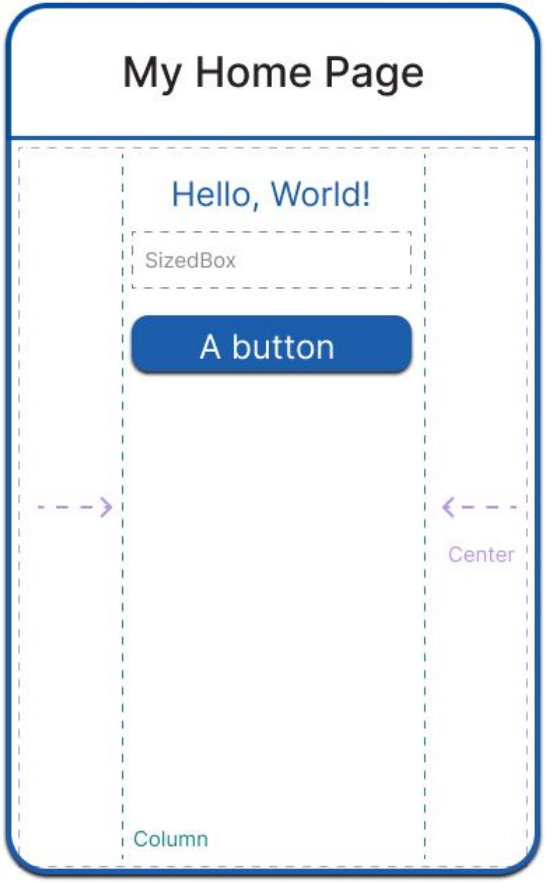

Don't invoke 'print' in production code.
Try using a logging framework. dart(avoid_print)
void print(Object? object)
Type: void Function(Object?)
Declared in dart:core.
Prints an object to the console.
On the web, object is converted to a string and that string is output to the web console using console.log.
On native (non-Web) platforms, object is converted to a string and that string is terminated by a line feed ('\n' , U+000A) and written to stdout . On Windows, the terminating line feed, and any line feeds in the string representation of object , are output using the Windows line terminator sequence of ('\r\n' , U+000D + U+000A).
OutlinedButton(onPressed: () {print("Hello World2");}, child: const Text("Press Me"))

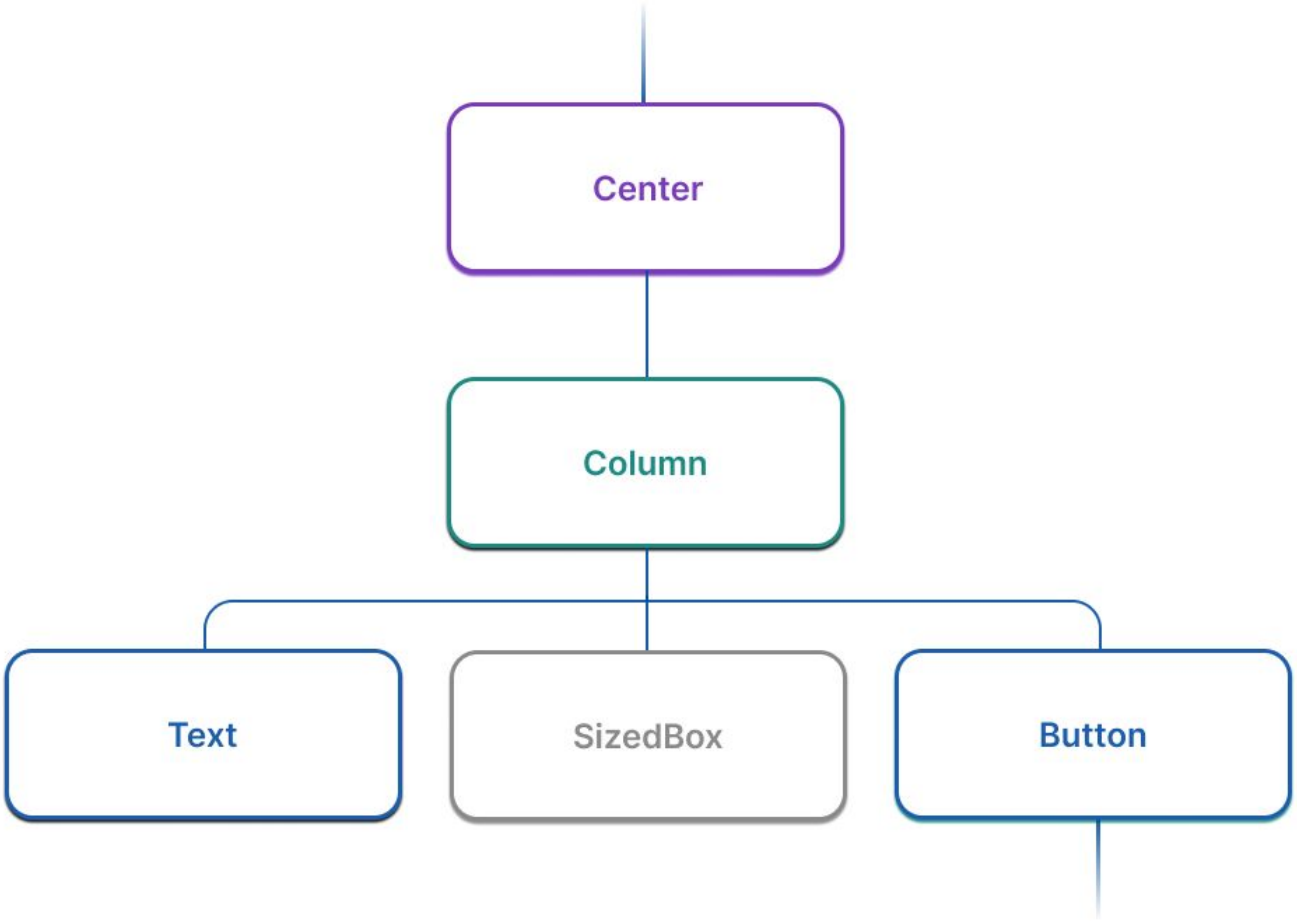
```
      style: Theme.of(context).textTheme.headlineMedium,
    ), // Text
    OutlinedButton(onPressed: () {debugPrint("Hello World2");}, child: const Text("Press Me"))
  ],
), // Column
```

Widget composition






My Home Page
Hello, World!
SizedBox
A button
Center
Column


Center
Column
Text
SizedBox
Button

Flutter Stateless and Stateful Widgets

“เราจะพบ ประเภทของ Widget หลัก ๆ ใน Flutter ได้แก่ Stateless และ Stateful Widgets”

“การใช้งาน StatelessWidget
เป็น widget ที่ไม่จำเป็นที่ต้อง
มีการเปลี่ยนแปลง state เกิดขึ้น ”

State less Widget

main.dart | ourPage.dart ● | pubspec.yaml

lib > ourPage.dart > ...

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Stateless Widget Example'),
        ), // AppBar
        body: Center(
          child: Text('Hello, World!'),
        ), // Center
      ), // Scaffold
    ); // MaterialApp
  }
}

```

“ การใช้งาน StatefulWidget
เป็น Widget ที่รอรับการเปลี่ยนแปลงของข้อมูล
ถ้ามีข้อมูลอะไรเปลี่ยนก็จะสั่ง rerun / build() ใหม่ ”

State ful Widget

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

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

class MyApp extends StatefulWidget {
  @override
  _MyAppState createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Stateful Widget Example'),
        ), // AppBar
        body: Center(
          child: Text('Button pressed $_counter times.'),
        ), // Center
        floatingActionButton: FloatingActionButton(
          onPressed: _incrementCounter,
          tooltip: 'Increment',
          child: Icon(Icons.add),
        ), // FloatingActionButton
      ), // Scaffold
    ); // MaterialApp
  }
}

```

แล้วถ้าเราต้องการเปลี่ยนการแสดงผล Text หรือ Icon ที่เป็น Stateless หละ ? มันแตกต่างยังไง

แล้วถ้าเราต้องการเปลี่ยนการแสดงผล Text หรือ Icon ที่เป็น Stateless หละ ? มันแตกต่างยังไง

Stateless ไม่ได้แปลว่า “UI เปลี่ยนไม่ได้ ”

คำว่า Stateless ใน Flutter หมายถึง
ตัว widget ไม่มี state ที่เปลี่ยนเองภายในตัวมัน

มันแค่ “อ่านค่า” จาก constructor
แล้ว render ตามนั้น

```dart
class CounterScreen extends StatefulWidget {
  const CounterScreen({super.key});

  @override
  State<CounterScreen> createState() => _CounterScreenState();
}

class _CounterScreenState extends State<CounterScreen> {
  int _counter = 0;

  void _increment() {
    setState(() {
      _counter++;   // เปลี่ยน state ตรงนี้
    });
  }
```

เปลี่ยนข้อความใน `Text` ที่เป็น Stateless

```dart
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        // Text เป็น StatelessWidget แต่ค่าที่ส่งเข้าไปเปลี่ยนได้
        Text('$_counter'),

        ElevatedButton(
          onPressed: _increment,
          child: const Text('เพิ่ม'),
        ),
      ],
    );
  }
}
```

```dart
class ToggleIcon extends StatefulWidget {
  const ToggleIcon({super.key});

  @override
  State<ToggleIcon> createState() => _ToggleIconState();
}

class _ToggleIconState extends State<ToggleIcon> {
  bool _isFavorite = false;

  void _toggle() {
    setState(() {
      _isFavorite = !_isFavorite;
    });
  }
```

แล้ว Icon ล่ะ?
เปลี่ยนได้เหมือนกันเลย

```dart
class ToggleIcon extends StatefulWidget {
  const ToggleIcon({super.key});

  @override
  State<ToggleIcon> createState() => _ToggleIconState();
}

class _ToggleIconState extends State<ToggleIcon> {
  bool _isFavorite = false;

  void _toggle() {
    setState(() {
      _isFavorite = !_isFavorite;
    });
  }

  @override
  Widget build(BuildContext context) {
    return IconButton(
      icon: Icon(
        _isFavorite ? Icons.favorite : Icons.favorite_border,
      ),
      onPressed: _toggle,
    );
  }
}
```

แล้ว Icon ล่ะ?
เปลี่ยนได้เหมือนกันเลย

สรุปแล้วมันต่างจาก Stateful ยังไงแน่ ๆ ?

กรณี Stateless (เช่น Text, Icon)

ไม่มีตัวแปรภายในที่เปลี่ยนไปตามเวลา

ทุกอย่างที่มันต้องใช้ ถูกส่งเข้ามาทาง constructor

สรุปแล้วมันต่างจาก Stateful ยังไงแน่ ๆ ?

```
Text(_isOn ? 'ON' : 'OFF'); // เปลี่ยนเพราะ _isOn เปลี่ยน
```

กรณี Stateful

ตัว widget (StatefulWidget) ยัง immutable อยู่เหมือนเดิม
แต่มี class State แยกออกมาไว้เก็บตัวแปรที่เปลี่ยนได้

```dart
class SettingsScreen extends StatefulWidget {
  const SettingsScreen({super.key});

  @override
  State<SettingsScreen> createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  bool _isDarkMode = false; // state ในหน้าจอเรา

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Built-in Stateful: Switch')),
      body: Column(
        children: [
          SwitchListTile(
            title: const Text('ใช้ Dark Mode'),
            value: _isDarkMode,
            onChanged: (value) {
              setState(() {
                _isDarkMode = value;  // อัปเดต state ของเรา
              });
            },
          ),
          const SizedBox(height: 16),
          Text(
            _isDarkMode ? 'ตอนนี้: Dark Mode' : 'ตอนนี้: Light Mode',
            style: const TextStyle(fontSize: 20),
          ),
        ],
      ),
    );
  }
}
```

SwitchListTile

```dart
class SettingsScreen extends StatefulWidget {
  const SettingsScreen({super.key});

  @override
  State<SettingsScreen> createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  bool _isDarkMode = false; // state ในหน้าจอเรา
```

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('Built-in Stateful: Switch')),
    body: Column(
      children: [
        SwitchListTile(
          title: const Text('ใช้ Dark Mode'),
          value: _isDarkMode,
          onChanged: (value) {
```

```
            setState(() {
              _isDarkMode = value;  // อัปเดต state ของเรา
            });
          },
        ),
        const SizedBox(height: 16),
        Text(
          _isDarkMode ? 'ตอนนี้: Dark Mode' : 'ตอนนี้: Light Mode',
          style: const TextStyle(fontSize: 20),
        ),
      ],
```

```dart
class SettingsScreen extends StatefulWidget {
  const SettingsScreen({super.key});

  @override
  State<SettingsScreen> createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
  bool _isDarkMode = false; // state ในหน้าจอเรา

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Built-in Stateful: Switch')),
      body: Column(
        children: [
          SwitchListTile(
            title: const Text('ใช้ Dark Mode'),
            value: _isDarkMode,
            onChanged: (value) {
              setState(() {
                _isDarkMode = value;  // อัปเดต state ของเรา
              });
            },
          ),
          const SizedBox(height: 16),
          Text(
            _isDarkMode ? 'ตอนนี้: Dark Mode' : 'ตอนนี้: Light Mode',
            style: const TextStyle(fontSize: 20),
          ),
        ],
      ),
    );
  }
}
```

```dart
class VolumeSliderScreen extends StatefulWidget {
  const VolumeSliderScreen({super.key});

  @override
  State<VolumeSliderScreen> createState() => _VolumeSliderScreenState();
}

class _VolumeSliderScreenState extends State<VolumeSliderScreen> {
  double _volume = 0.5; // 0.0 - 1.0

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Built-in Stateful: Slider')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Slider(
              value: _volume,
              min: 0,
              max: 1,
              onChanged: (value) {
                setState(() {
                  _volume = value;
                });
              },
            ),
            Text(
              'Volume: ${(_volume * 100).toStringAsFixed(0)}%',
              style: const TextStyle(fontSize: 20),
            ),
          ],
        ),
      ),
    );
  }
}
```

Slider

```dart
class VolumeSliderScreen extends StatefulWidget {
  const VolumeSliderScreen({super.key});

  @override
  State<VolumeSliderScreen> createState() => _VolumeSliderScreenState();
}

class _VolumeSliderScreenState extends State<VolumeSliderScreen> {
  double _volume = 0.5; // 0.0 - 1.0

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Built-in Stateful: Slider')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Slider(
              value: _volume,
              min: 0,
              max: 1,
              onChanged: (value) {
                setState(() {
                  _volume = value;
                });
              },
            ),
            Text(
              'Volume: ${(_volume * 100).toStringAsFixed(0)}%',
              style: const TextStyle(fontSize: 20),
            ),
          ],
        ),
      ),
    );
  }
}
```

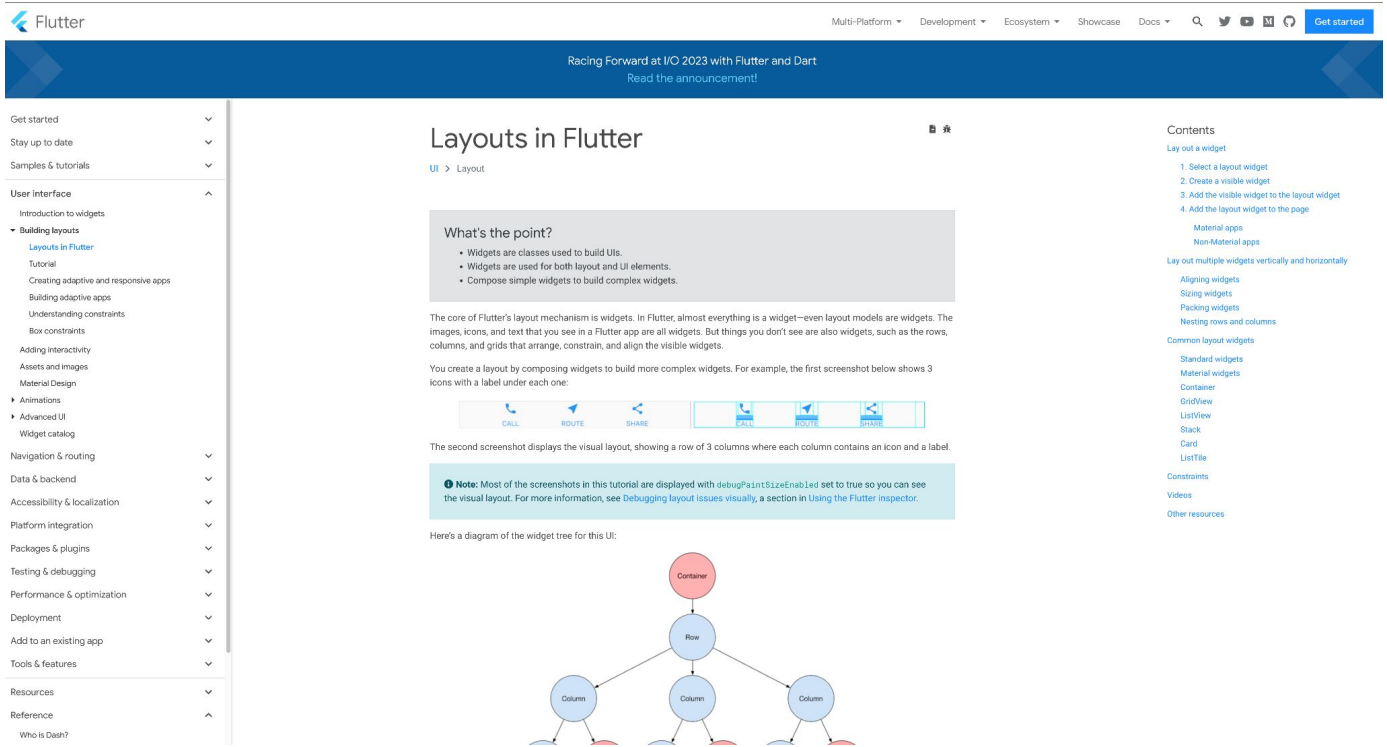
```dart
mainAxisSize: MainAxisSize.min,
children: [
  Slider(
    value: _volume,
    min: 0,
    max: 1,
    onChanged: (value) {
      setState(() {
        _volume = value;
      });
    },
  ),
  Text(
```

Stateless widget = “ป้ายบอกค่าปัจจุบันอย่างเดียว”
ข้อมูลเปลี่ยน = เราสร้างป้ายใหม่ด้วยข้อความใหม่

Stateful widget = “ป้ายที่มีสมุดโน้ตในตัวเอง”
มันจำค่าเก่าได้ใน `State` และเวลาเปลี่ยนค่าตัวเอง
ก็เรียก `setState()` เพื่อวาดใหม่

Flutter Building Layout and Widgets

การใช้งาน Layout ใน Flutter



การใช้งาน Layout ใน Flutter

The core of Flutter's layout mechanism is widgets. In Flutter, almost everything is a widget—even layout models are widgets. The images, icons, and text that you see in a Flutter app are all widgets. But things you don't see are also widgets, such as the rows, columns, and grids that arrange, constrain, and align the visible widgets.

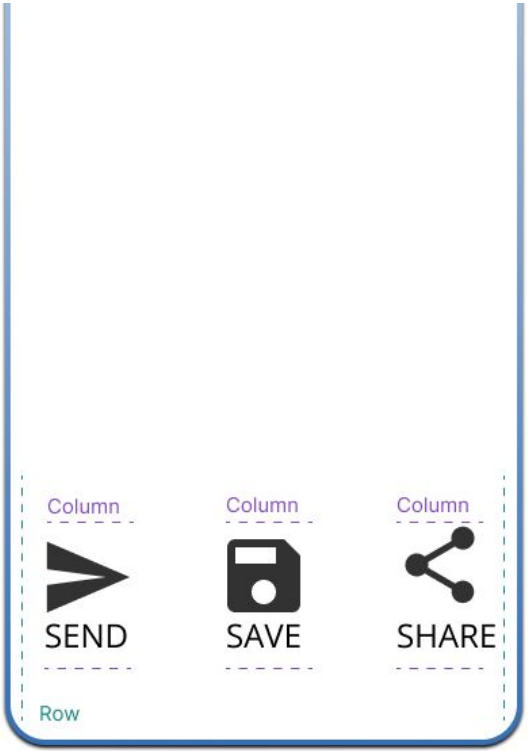
You create a layout by composing widgets to build more complex widgets. For example, the first screenshot below shows 3 icons with a label under each one:

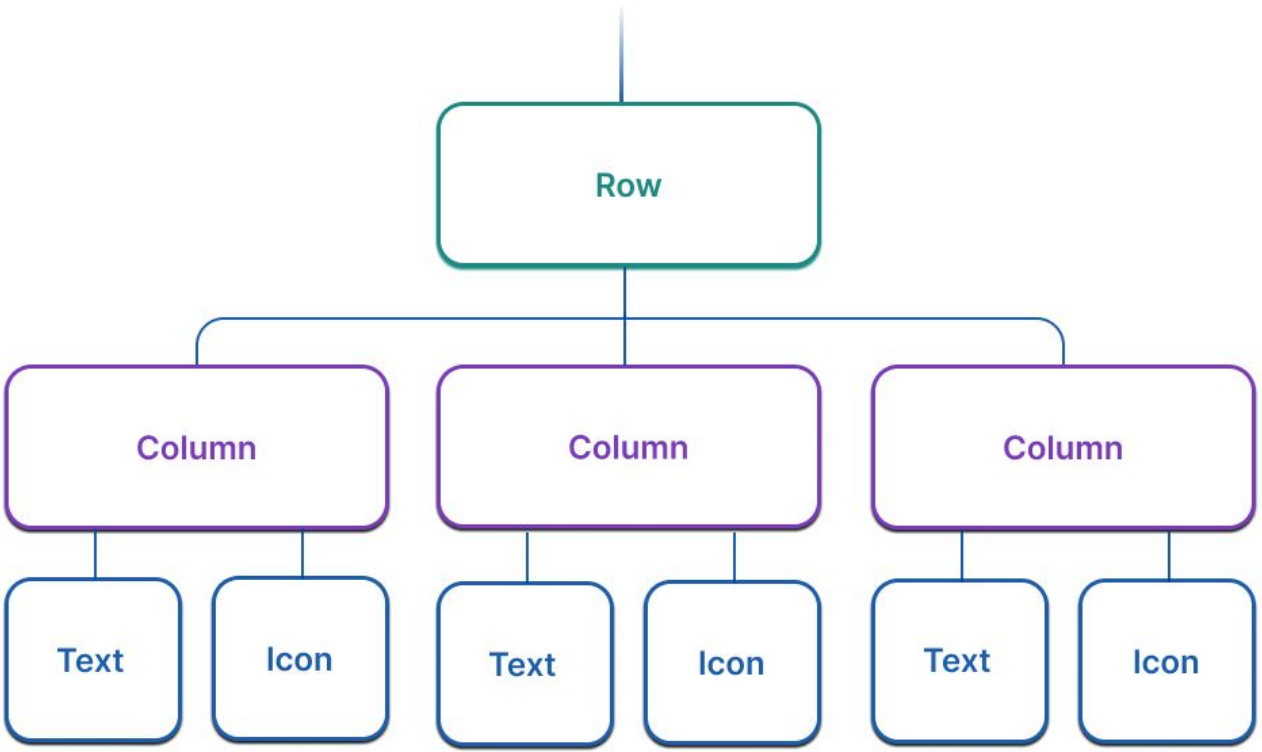
The second screenshot displays the visual layout, showing a row of 3 columns where each column contains an icon and a label.

Understanding layout in Flutter

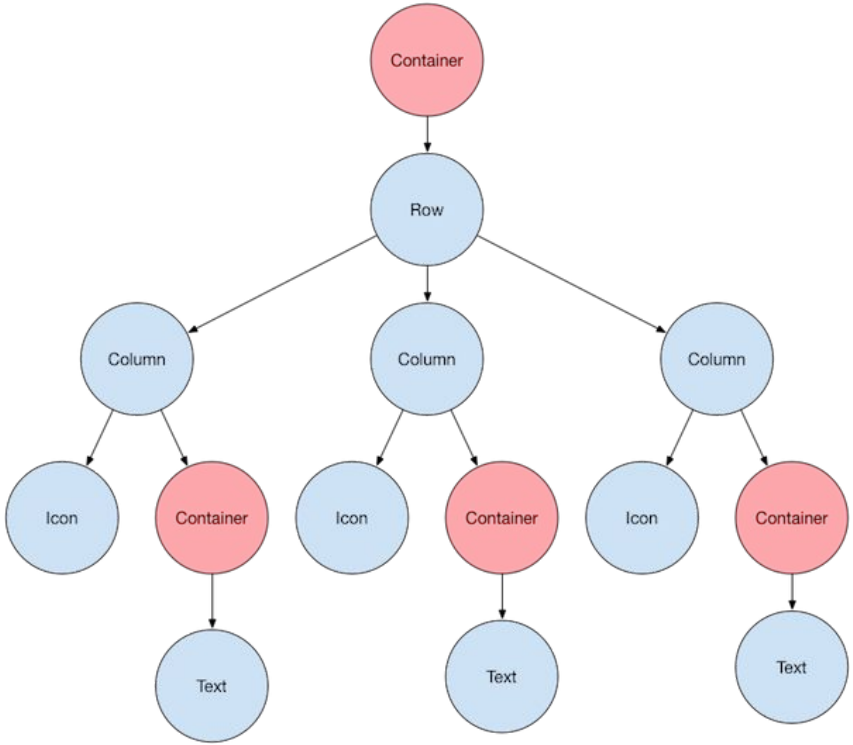
The core of Flutter's layout mechanism is widgets. In Flutter, almost everything is a widget — even layout models are widgets. The images, icons, and text that you see in a Flutter app are all widgets. Things you don't see are also widgets, such as the rows, columns, and grids that arrange, constrain, and align the visible widgets.

You create a layout by composing widgets to build more complex widgets. For example, the diagram below shows 3 icons with a label under each one, and the corresponding widget tree:

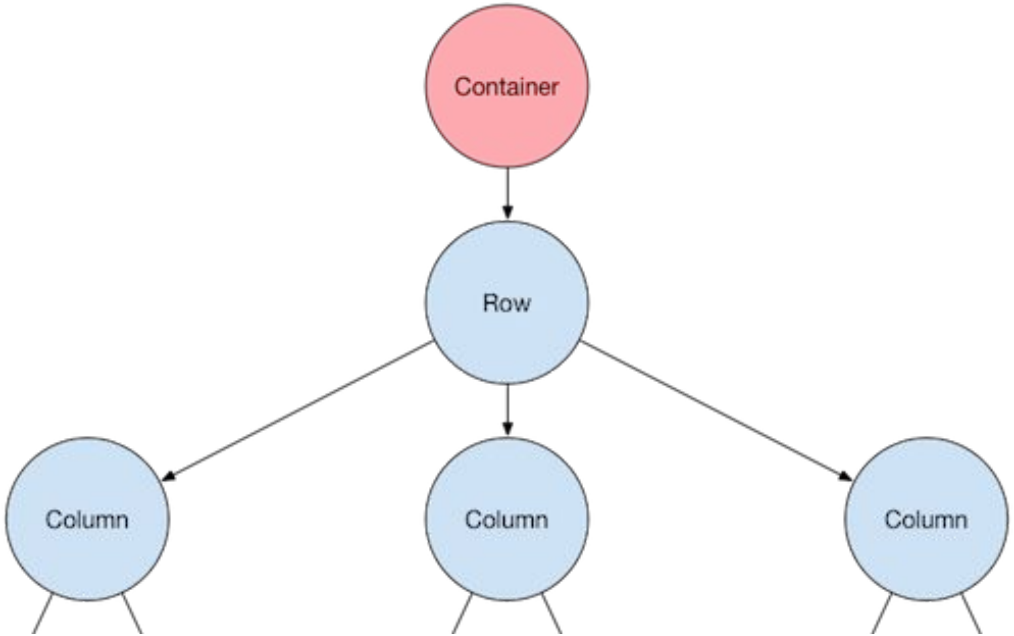

Column
Column
Column
SEND
SAVE
SHARE
Row


Row
Column
Column
Column
Text
Icon
Text
Icon
Text
Icon

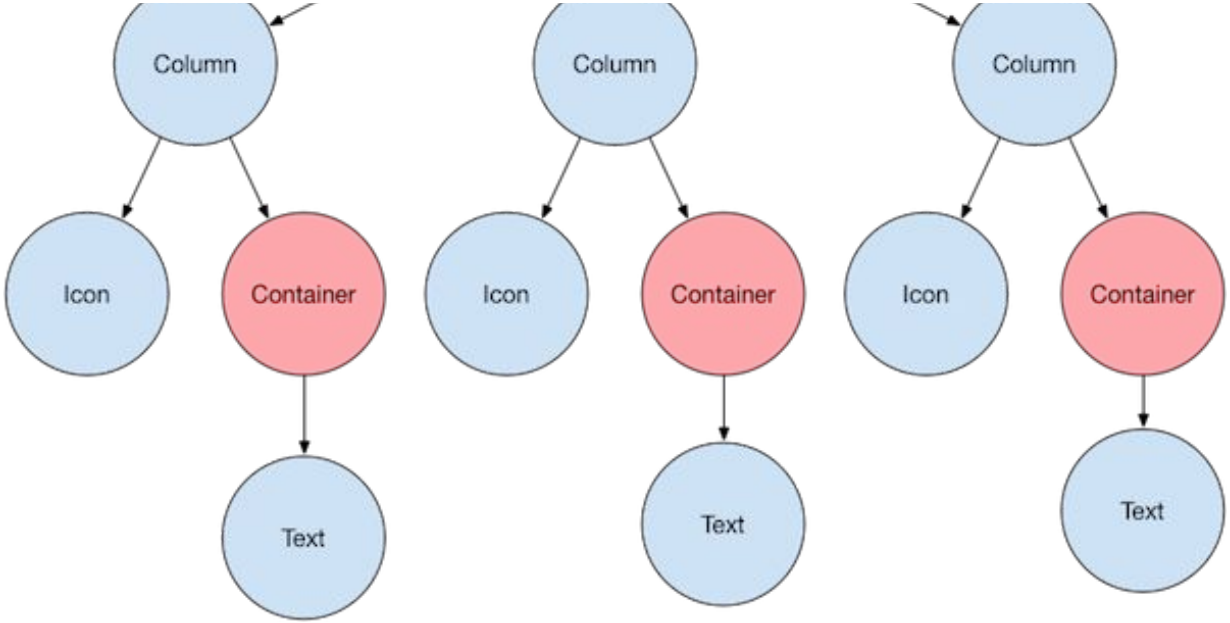
Widget Tree (อีกรอบ)



Widget Tree (อีกรอบ)



Widget Tree (อีกรอบ)



Constraints

Understanding constraints in Flutter is an important part of understanding how layout works in Flutter.

Layout, in a general sense, refers to the size of the widgets and their positions on the screen. The size and position of any given widget is constrained by its parent; it can't have any size it wants, and it doesn't decide its own place on the screen. Instead, size and position are determined by a conversation between a widget and its parent.

In the simplest example, the layout conversation looks like this:

1. A widget receives its constraints from its parent.
2. A constraint is just a set of 4 doubles: a minimum and maximum width, and a minimum and maximum height.
3. The widget determines what size it should be within those constraints, and passes its width and height back to the parent.
4. The parent looks at the size it wants to be and how it should be aligned, and sets the widget's position accordingly. Alignment can be set explicitly, using a variety of widgets like `Center`, and the alignment properties on `Row` and `Column`.

In Flutter, this layout conversation is often expressed with the simplified phrase, "Constraints go down. Sizes go up. Parent sets the position."

Box types

In Flutter, widgets are rendered by their underlying `RenderBox` objects. These objects determine how to handle the constraints they're passed.

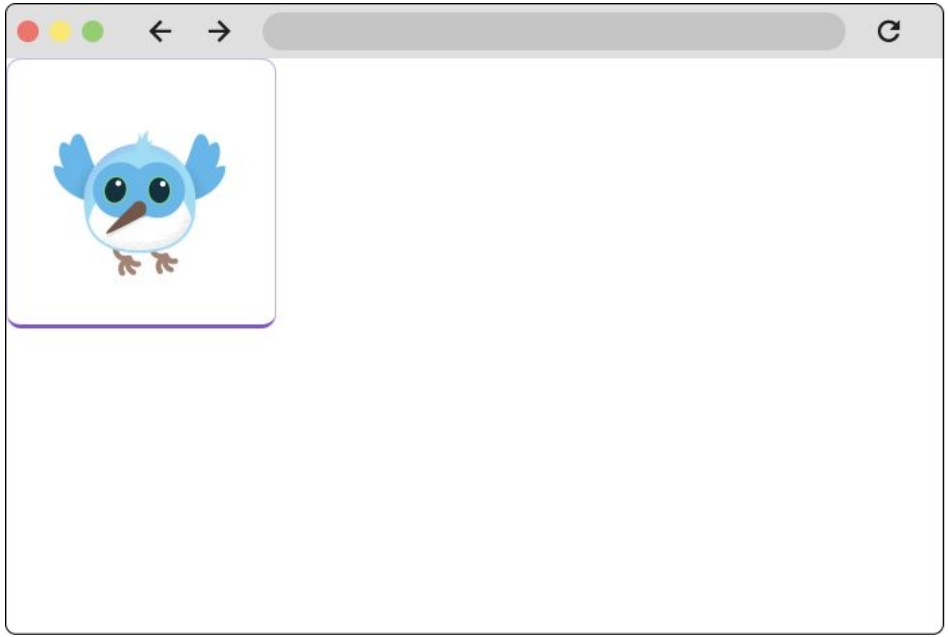
Generally, there are three kinds of boxes:

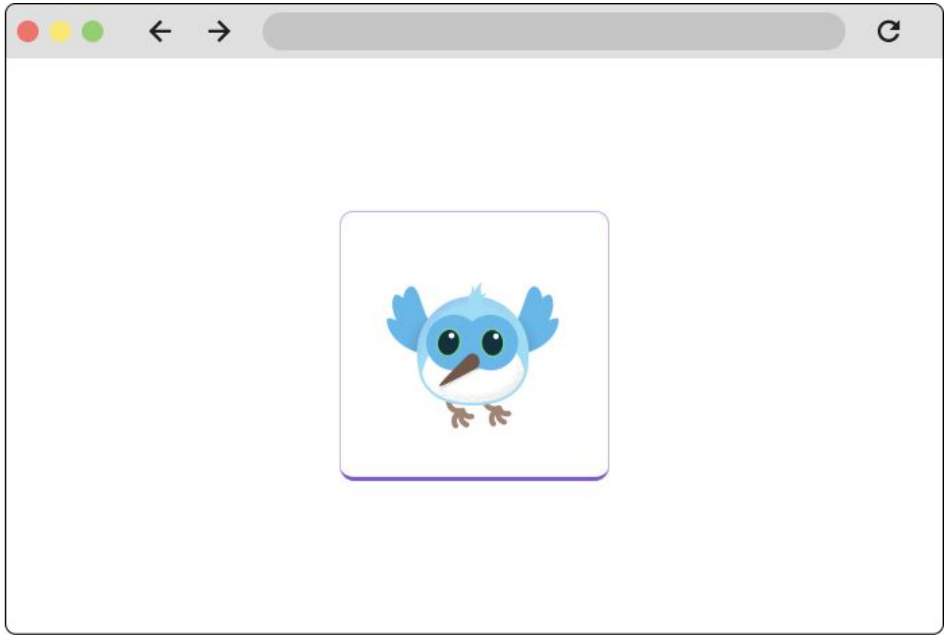
- Those that try to be as big as possible. For example, the boxes used by `Center` and `ListView`.
- Those that try to be the same size as their children. For example, the boxes used by `Transform` and `Opacity`
- Those that try to be a particular size. For example, the boxes used by `Image` and `Text`.

Some widgets, for example `Container`, vary from type to type based on their constructor arguments. The `Container` constructor defaults to trying to be as big as possible, but if you give it a width, for instance, it tries to honor that and be that particular size.

Others, for example `Row` and `Column` (flex boxes) vary based on the constraints they are given. Read more about flex boxes and constraints in the Understanding Constraints article.

Lay out a single widget

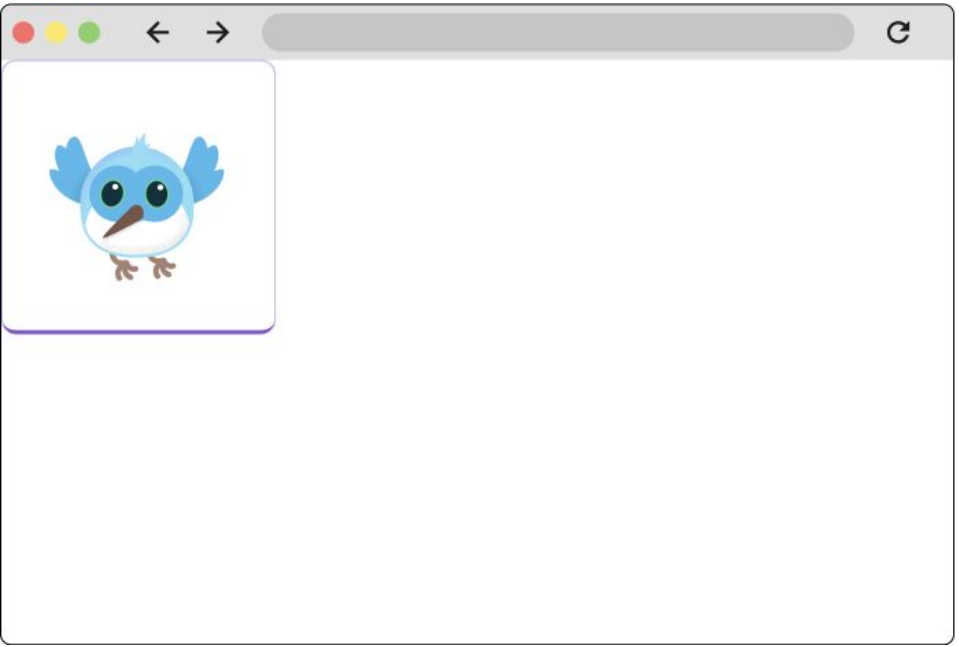


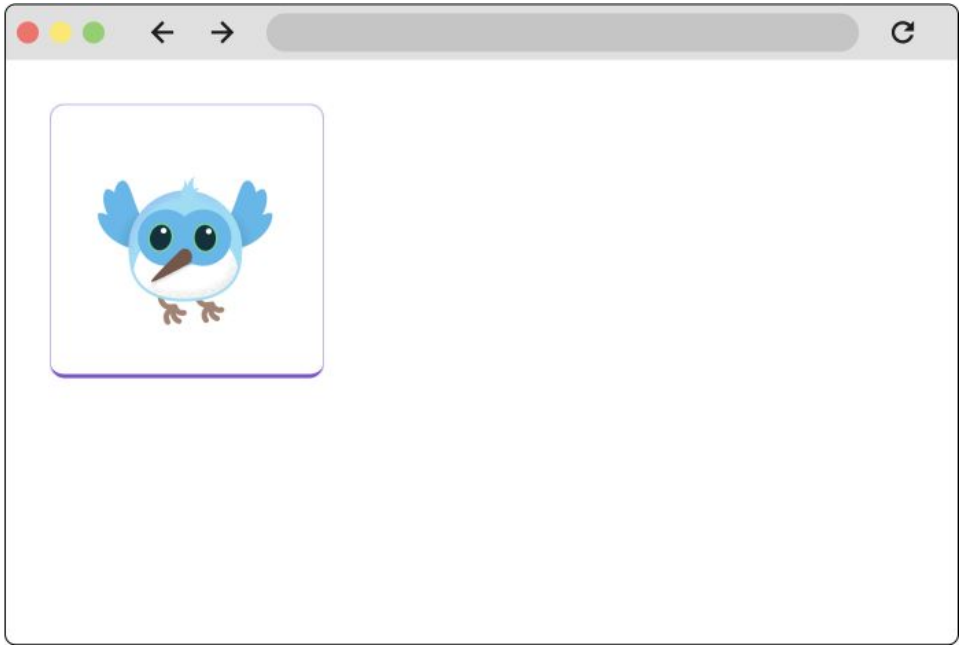


Lay out a single widget

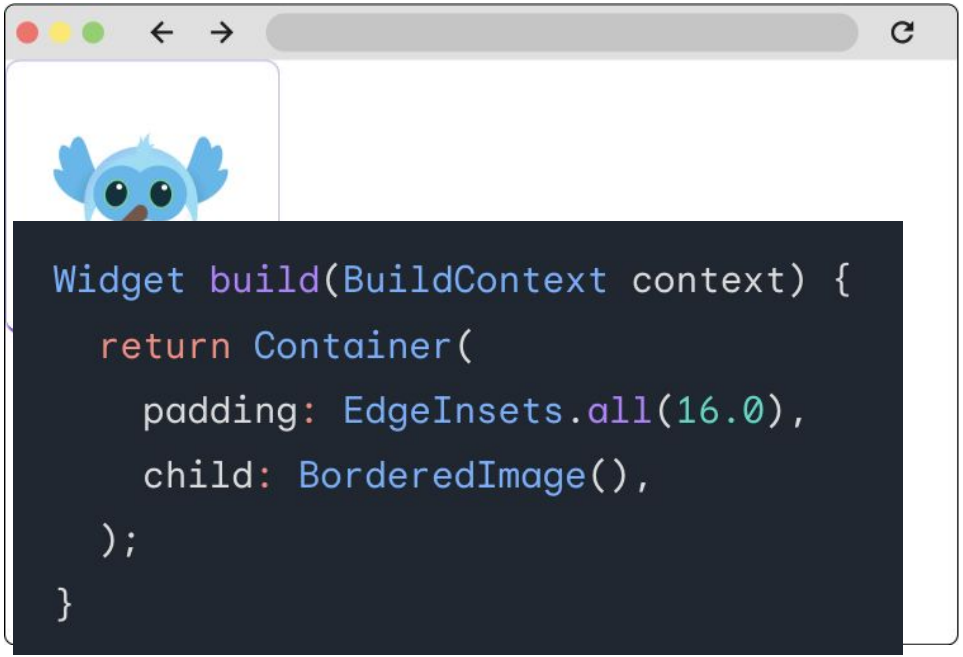
```
Widget build(BuildContext context) {
  return Center(
    child: BorderedImage(),
  );
}
```

Container

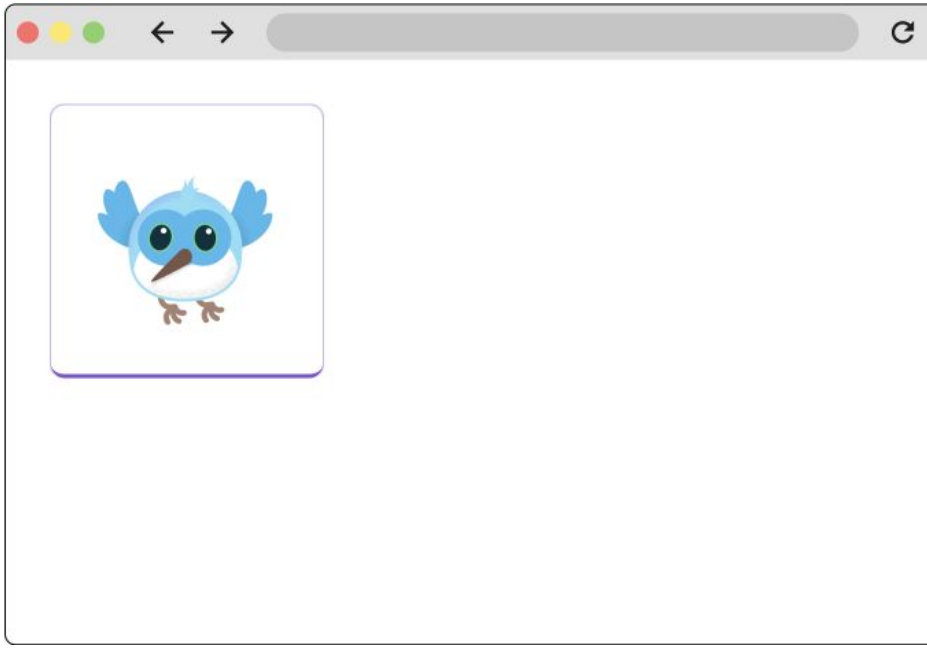




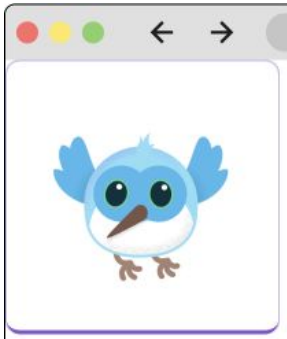
Container



```
Widget build(BuildContext context) {
  return Container(
    padding: EdgeInsets.all(16.0),
    child: BorderedImage(),
  );
}
```



Container



```
Widget build(BuildContext context) {
  return Center(
    Container(
      padding: EdgeInsets.all(16.0),
      child: BorderedImage(),
    ),
  );
}
```

อยากรู้ว่าใน Flutter มี Layout อะไรบ้าง ?

Layout widgets

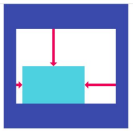
UI > Widgets > Layout

Arrange other widgets columns, rows, grids, and many other layouts.

- Single-child layout widgets
- Multi-child layout widgets
- Sliver widgets

See more widgets in the widget catalog.

Single-child layout widgets



Align

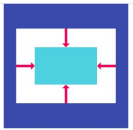
A widget that aligns its child within itself and optionally sizes itself based on the child's size.

AspectRatio

A widget that attempts to size the child to a specific aspect ratio.

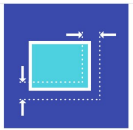
Baseline

Container that positions its child according to the child's baseline.



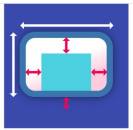
Center

Alignment block that centers its child within itself.



ConstrainedBox

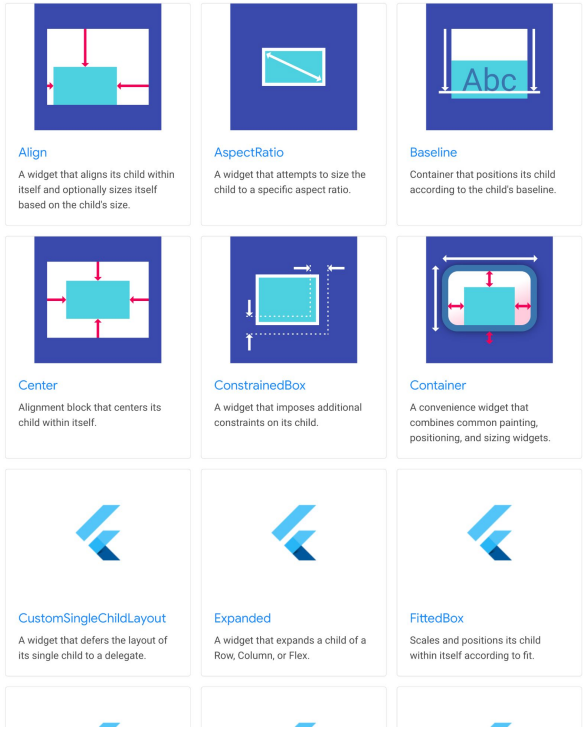
A widget that imposes additional constraints on its child.



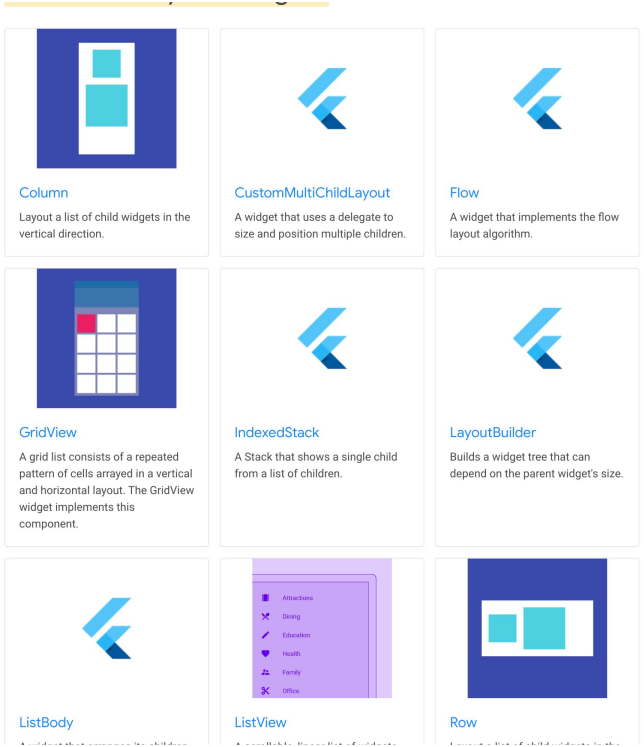
Container

A convenience widget that combines common painting, positioning, and sizing widgets.

Single-child layout widgets
คือ มีสมาชิกได้ตัวเดียว



Multi-child layout widgets
คือ มีสมาชิกได้หลายตัว



Lay out a widget

How do you lay out a single widget in Flutter? This section shows you how to create and display a simple widget. It also shows the entire code for a simple Hello World app.

In Flutter, it takes only a few steps to put text, an icon, or an image on the screen.

1. Select a layout widget

Choose from a variety of layout widgets based on how you want to align or constrain the visible widget, as these characteristics are typically passed on to the contained widget.

This example uses `Center` which centers its content horizontally and vertically.

2. Create a visible widget

For example, create a `Text` widget:

```
Text('Hello World'),
```

Create an `Image` widget:

```
Image.asset(
  'images/lake.jpg',
  fit: BoxFit.cover,
),
```

Create an `Icon` widget:

```
Icon(
  Icons.star,
  color: Colors.red[500],
),
```

3. Add the visible widget to the layout widget

All layout widgets have either of the following:

- A `child` property if they take a single child—for example, `Center` or `Container`
- A `children` property if they take a list of widgets—for example, `Row`, `Column`, `ListView`, or `Stack`.

Add the `Text` widget to the `Center` widget:

```
const Center(
  child: Text('Hello World'),
),
```

4. Add the layout widget to the page

A Flutter app is itself a widget, and most widgets have a `build()` method. Instantiating and returning a widget in the app's `build()` method displays the widget.

Material apps

For a `Material` app, you can use a `Scaffold` widget; it provides a default banner, background color, and has API for adding drawers, snack bars, and bottom sheets. Then you can add the `Center` widget directly to the `body` property for the home page.

lib/main.dart (MyApp)

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter layout demo',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Flutter layout demo'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

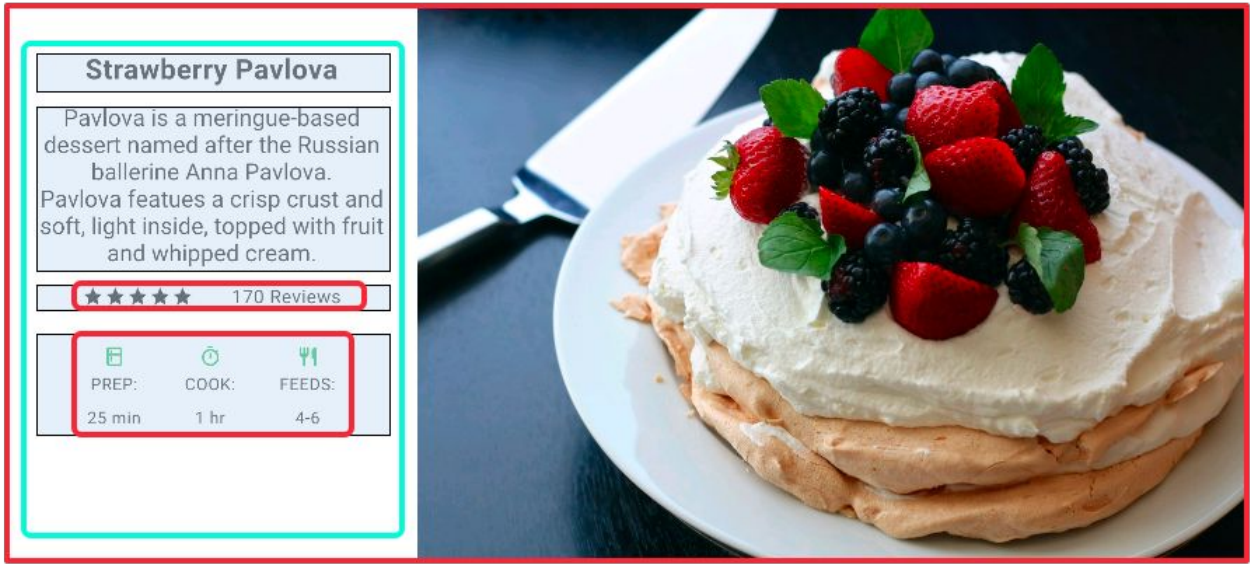
Non-Material apps

For a non-Material app, you can add the `Center` widget to the app's `build()` method:

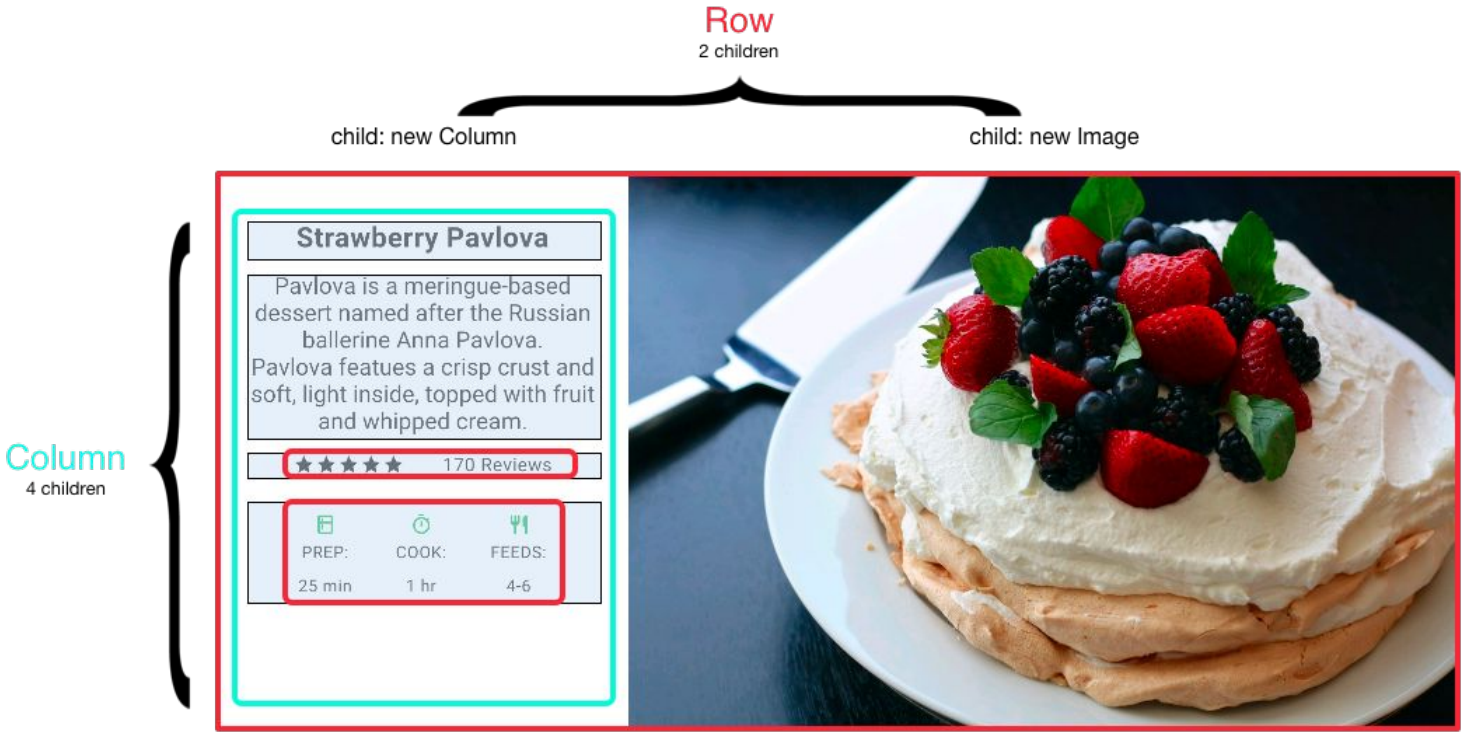
lib/main.dart (MyApp)

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

  @override
  Widget build(BuildContext context) {
    return Container(
      decoration: const BoxDecoration(color: Colors.white),
      child: const Center(
        child: Text(
          'Hello World',
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 32,
            color: Colors.black87,
          ),
        ),
      ),
    );
  }
}
```

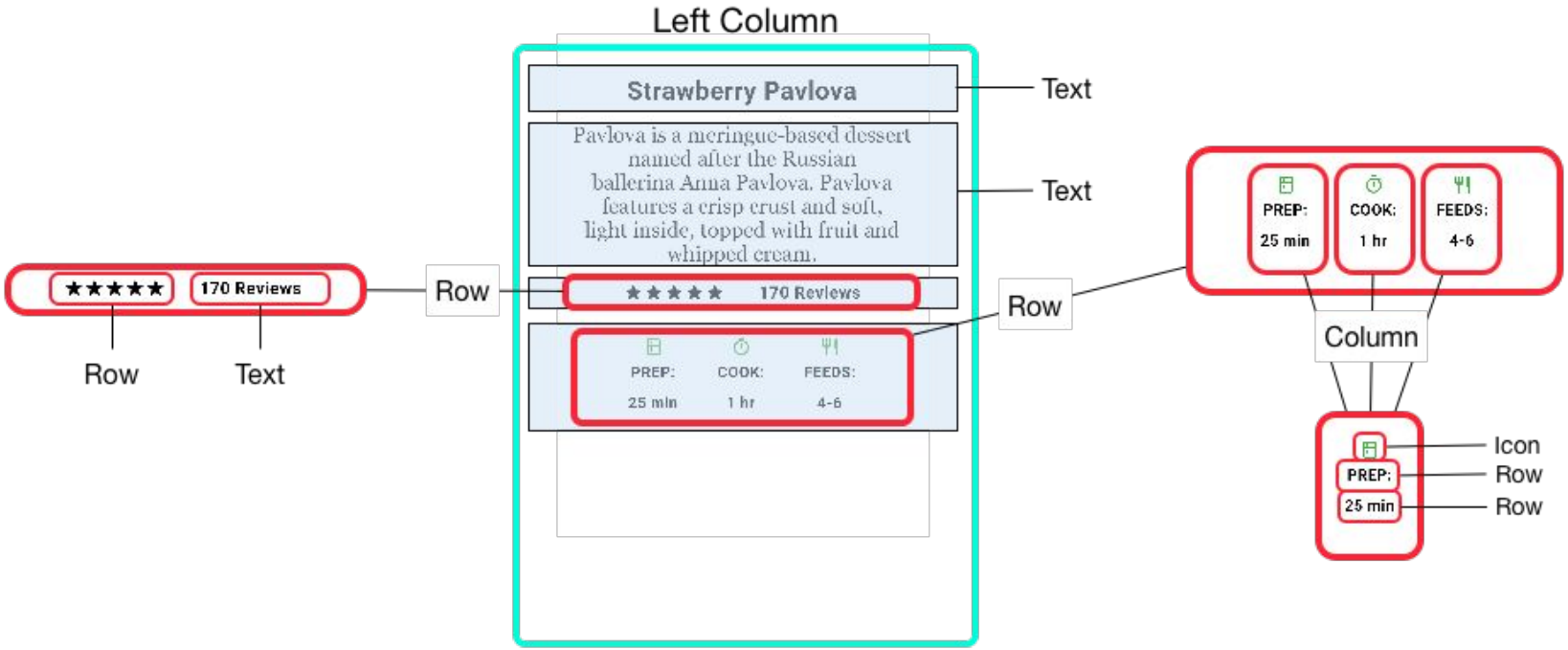
อยากได้แอปแบบนี้ทำยังไง ?



สังเกตได้ว่าเราจะใช้ Concept Row และ Column



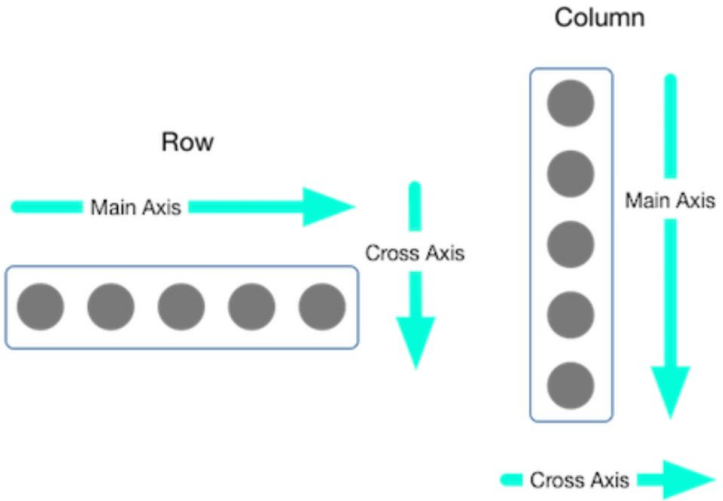
สังเกตได้ว่าเราจะใช้ Concept Row และ Column



ว่าแต่จะใช้ Row / Column แบบนี้ยังไง ?

Aligning widgets

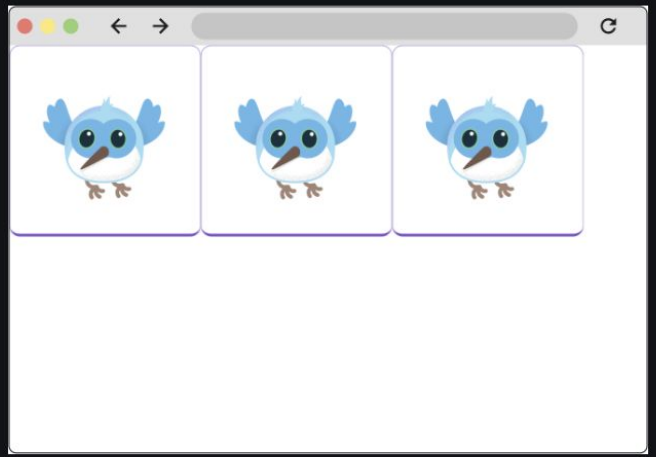
You control how a row or column aligns its children using the `mainAxisAlignment` and `crossAxisAlignment` properties. For a row, the main axis runs horizontally and the cross axis runs vertically. For a column, the main axis runs vertically and the cross axis runs horizontally.



The `MainAxisAlignment` and `CrossAxisAlignment` enums offer a variety of constants for controlling alignment.

Aligning widgets

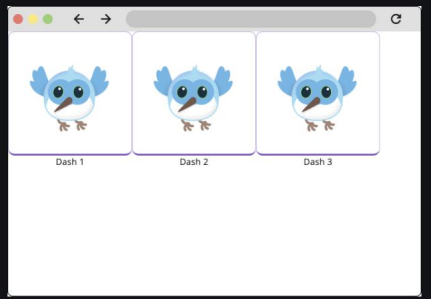
```dart
Widget build(BuildContext context) {
  return Row(
    children: [
      BorderedImage(),
      BorderedImage(),
      BorderedImage(),
    ],
  );
}
```



This figure shows a row widget with three children.

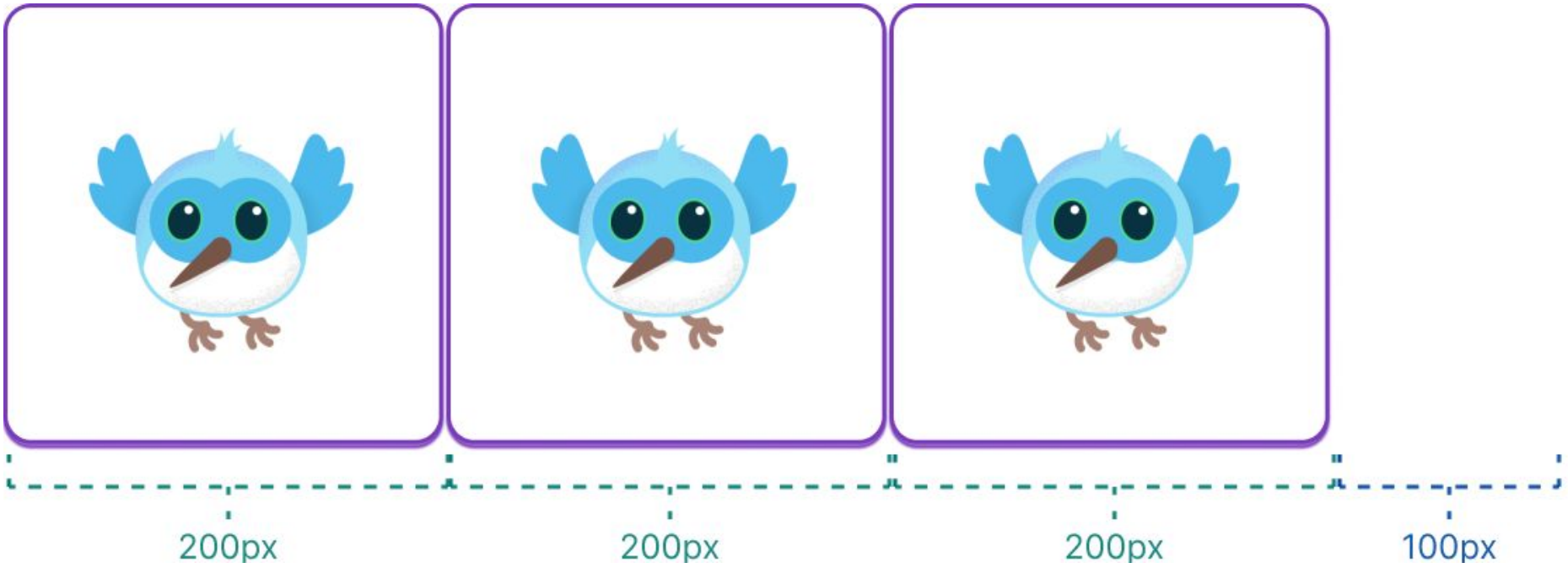
Aligning widgets

```dart
Widget build(BuildContext context) {
  return Row(
    children: [
      Column(
        children: [
          BorderedImage(),
          Text('Dash 1'),
        ],
      ),
      Column(
        children: [
          BorderedImage(),
          Text('Dash 2'),
        ],
      ),
      Column(
        children: [
          BorderedImage(),
          Text('Dash 3'),
        ],
      ),
    ],
  );
}
```

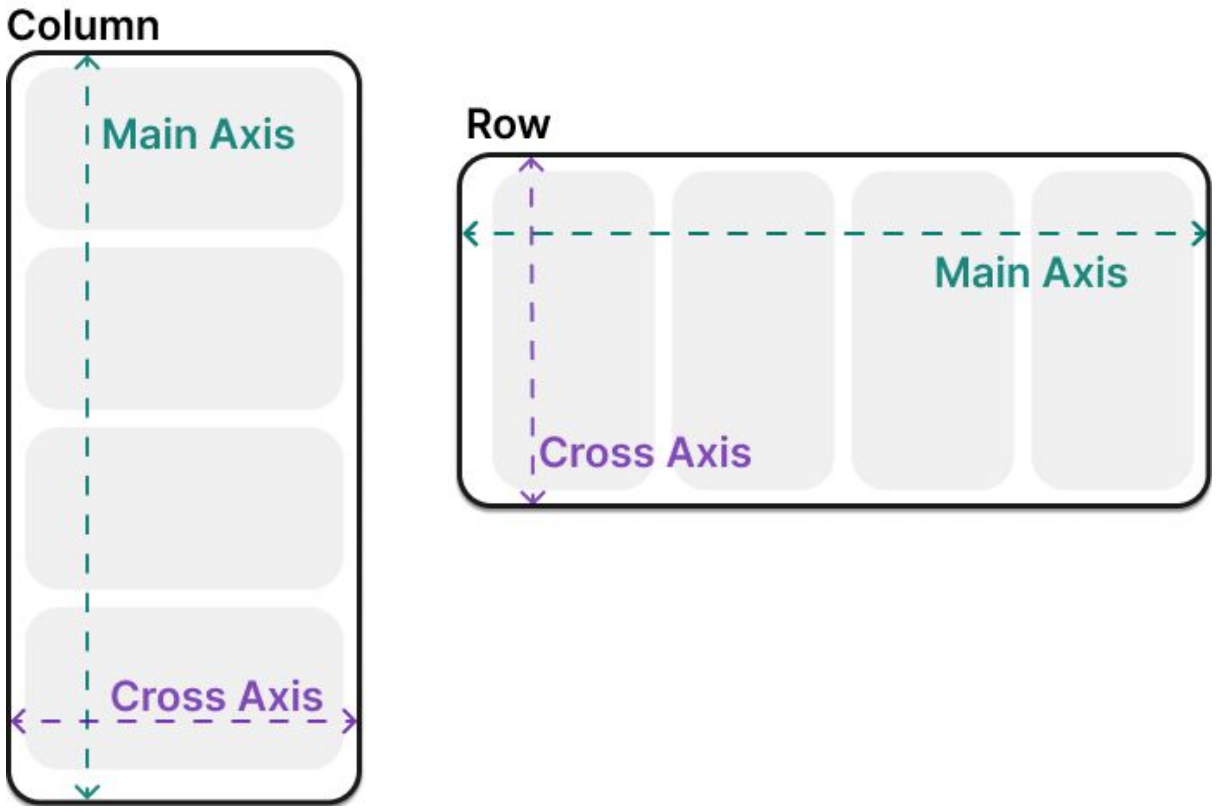


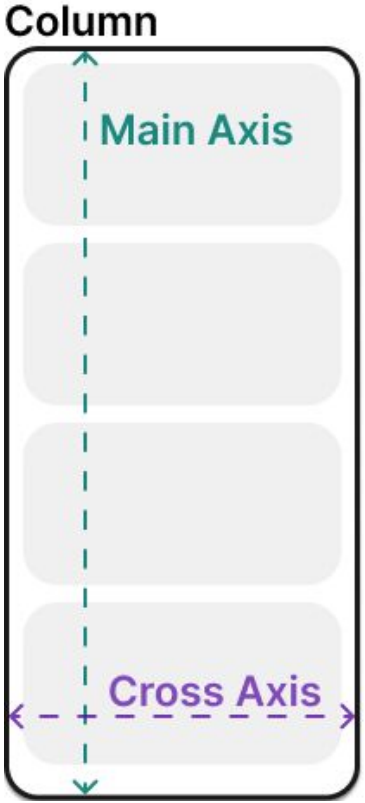
This figure shows a row widget with three children, each of which is a column.

Align widgets within rows and columns



เราคุมการจัดเรียงได้ด้วย `mainAxisAlignment` และ `crossAxisAlignment`





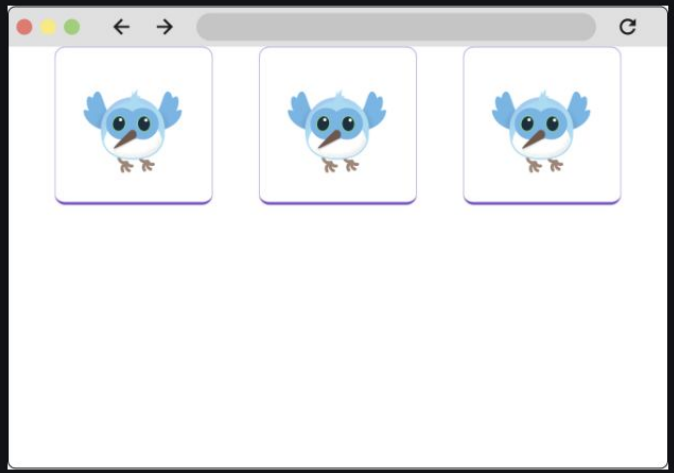
แกนหลัก (main axis) / แกนรอง (cross axis)

ใน Row : main = แนวนอน, cross = แนวตั้ง

ใน Column : main = แนวตั้ง, cross = แนวนอน

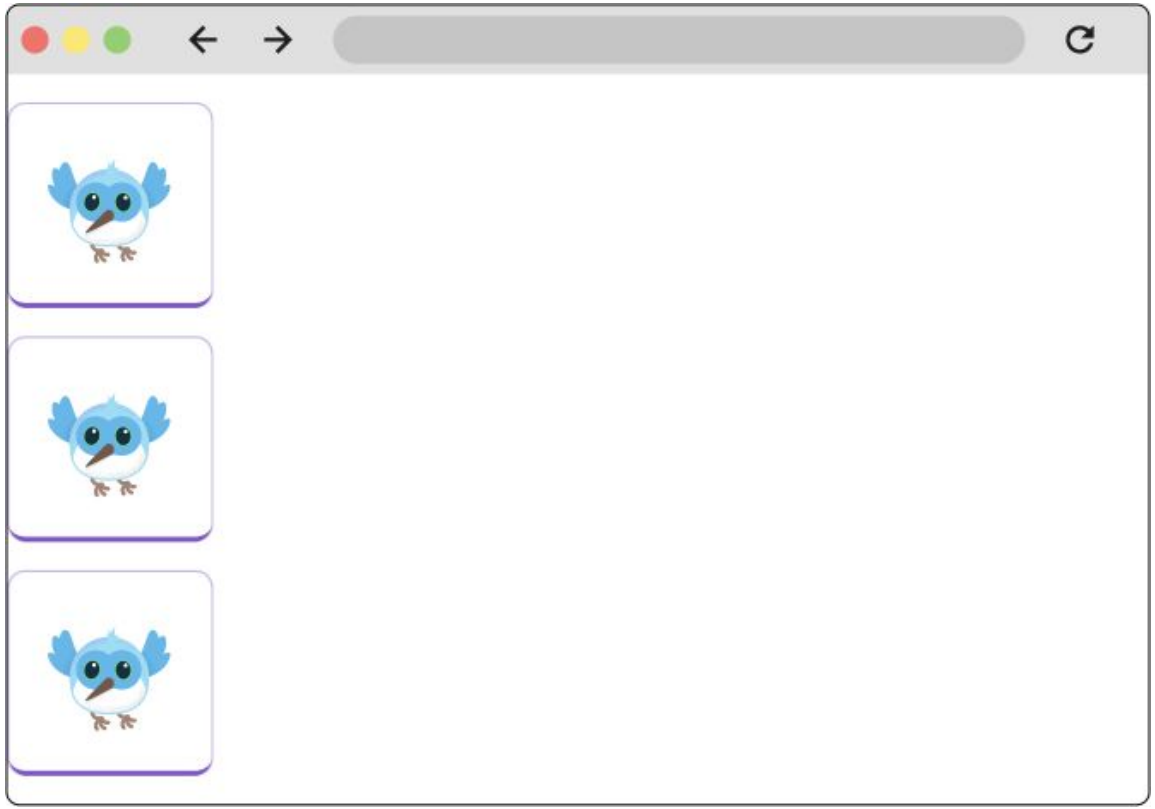
ตัวอย่าง: แบ่งพื้นที่แนวนอนให้ spaceEvenly (แบ่งเท่า ๆ กันก่อน/กลาง/หลัง)

```dart
Widget build(BuildContext context) {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      BorderedImage(),
      BorderedImage(),
      BorderedImage(),
    ],
  );
}
```



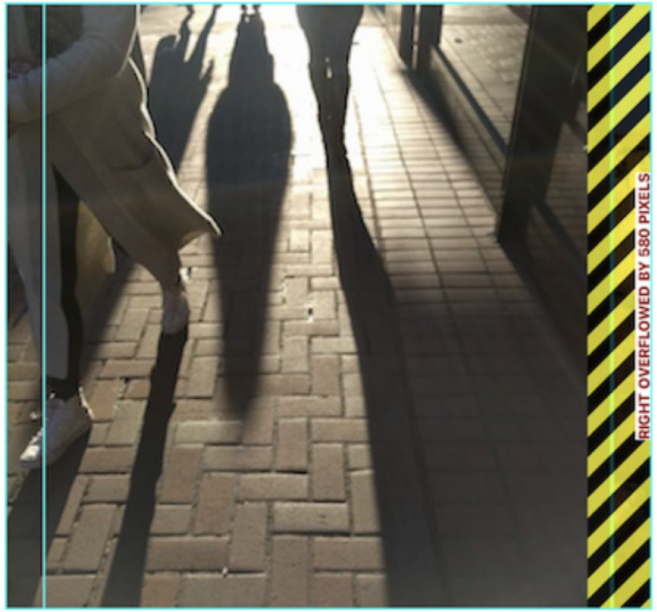
This figure shows a row widget with three children, which are aligned with the MainAxisAlignment.spaceEvenly constant.

ในกรณี Column ก็คิดเหมือนกัน แต่ mainAxis คือแนวตั้ง

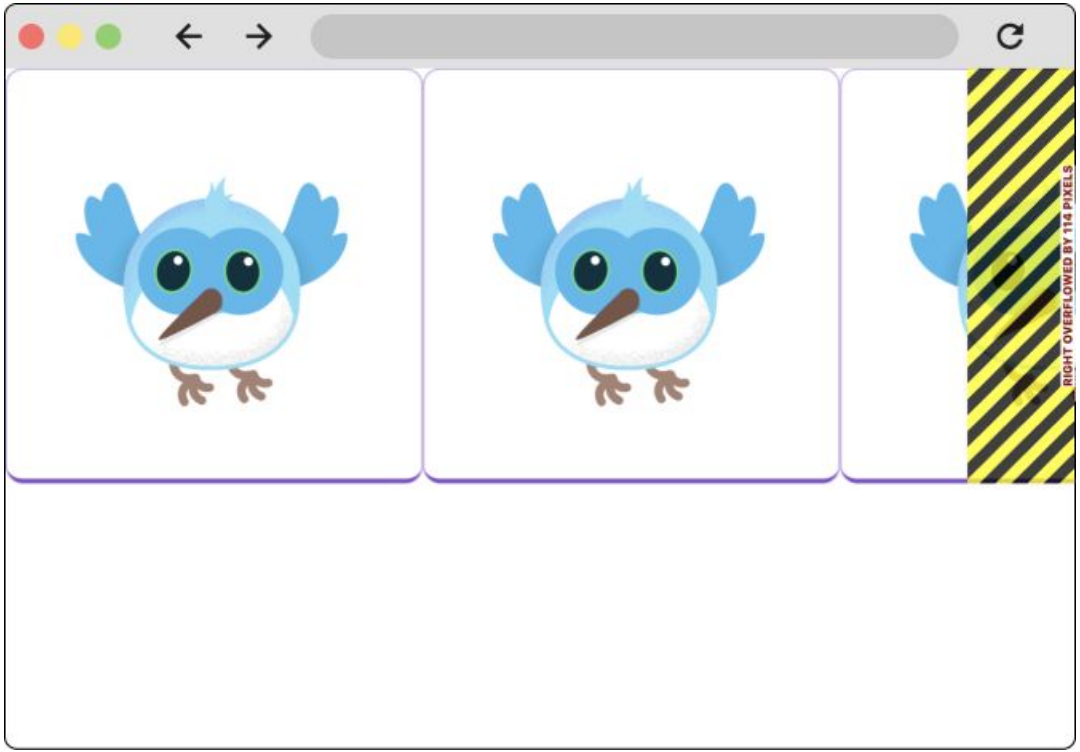


Sizing widgets

When a layout is too large to fit a device, a yellow and black striped pattern appears along the affected edge. Here is an example of a row that is too wide:

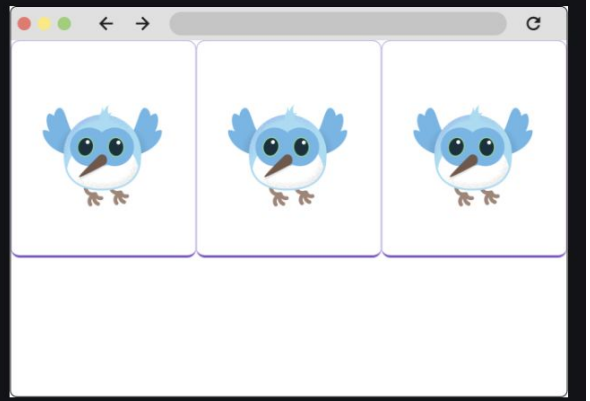


Sizing widgets within rows and columns



Sizing widgets within rows and columns

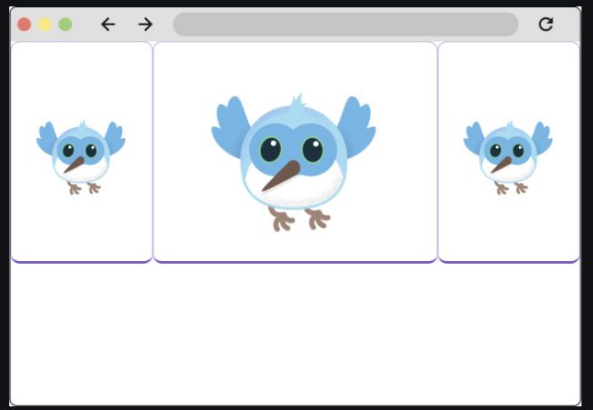
```dart
Widget build(BuildContext context) {
  return const Row(
    children: [
      Expanded(
        child: BorderedImage(width: 150, height: 150),
      ),
      Expanded(
        child: BorderedImage(width: 150, height: 150),
      ),
      Expanded(
        child: BorderedImage(width: 150, height: 150),
      ),
    ],
  );
}
```



This figure shows a row widget with three children that are wrapped with `Expanded` widgets.

Sizing widgets within rows and columns

```
Widget build(BuildContext context) {
  return const Row(
    children: [
      Expanded(
        child: BorderedImage(width: 150, height: 150),
      ),
      Expanded(
        flex: 2,
        child: BorderedImage(width: 150, height: 150),
      ),
      Expanded(
        child: BorderedImage(width: 150, height: 150),
      ),
    ],
  );
}
```



This figure shows a row widget with three children which are wrapped with `Expanded` widgets. The center child has it's `flex` property set to 2.

Sizing widgets

Widgets can be sized to fit within a row or column by using the `Expanded` widget. To fix the previous example where the row of images is too wide for its render box, wrap each image with an `Expanded` widget.

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic2.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic3.jpg'),
    ),
  ],
);
```

App source: sizing

Sizing widgets

Perhaps you want a widget to occupy twice as much space as its siblings. For this, use the `Expanded` widget `flex` property, an integer that determines the flex factor for a widget. The default flex factor is 1. The following code sets the flex factor of the middle image to 2:

```
Row(
  crossAxisAlignment: CrossAxisAlignment.center,
  children: [
    Expanded(
      child: Image.asset('images/pic1.jpg'),
    ),
    Expanded(
      flex: 2,
      child: Image.asset('images/pic2.jpg'),
    ),
    Expanded(
      child: Image.asset('images/pic3.jpg'),
    ),
  ],
);
```

App source: sizing

Packing widgets

By default, a row or column occupies as much space along its main axis as possible, but if you want to pack the children closely together, set its `mainAxisSize` to `MainAxisSize.min`. The following example uses this property to pack the star icons together.

```
Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    const Icon(Icons.star, color: Colors.black),
    const Icon(Icons.star, color: Colors.black),
  ],
)
```

App source: pavlova

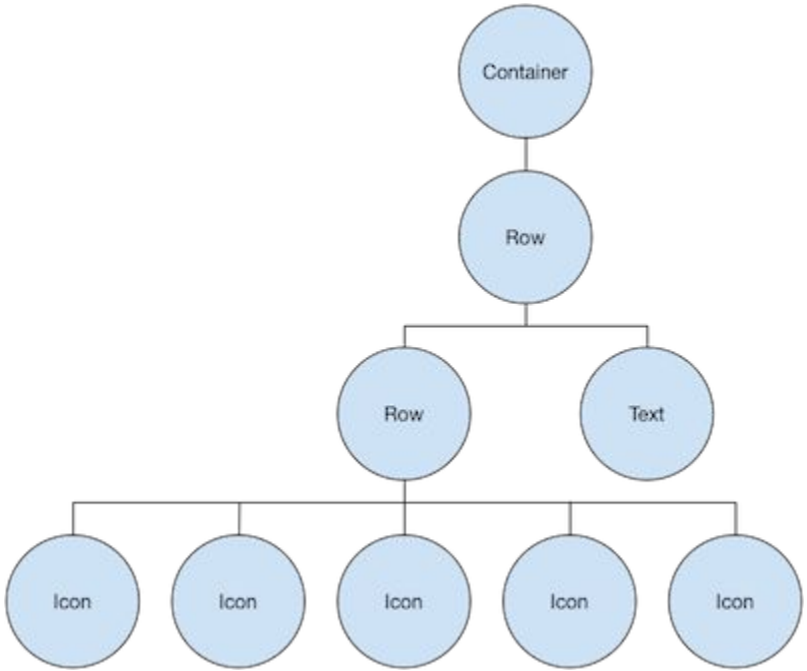
Ne s

named after the Russian ballerina Anna Pavlova. Pavlova features a crisp crust and soft, light inside, topped with fruit and whipped cream.

★★★★★ 170 Reviews

PREP:	COOK:	FEEDS:
25 min	1 hr	4-6

The widget tree for the ratings row

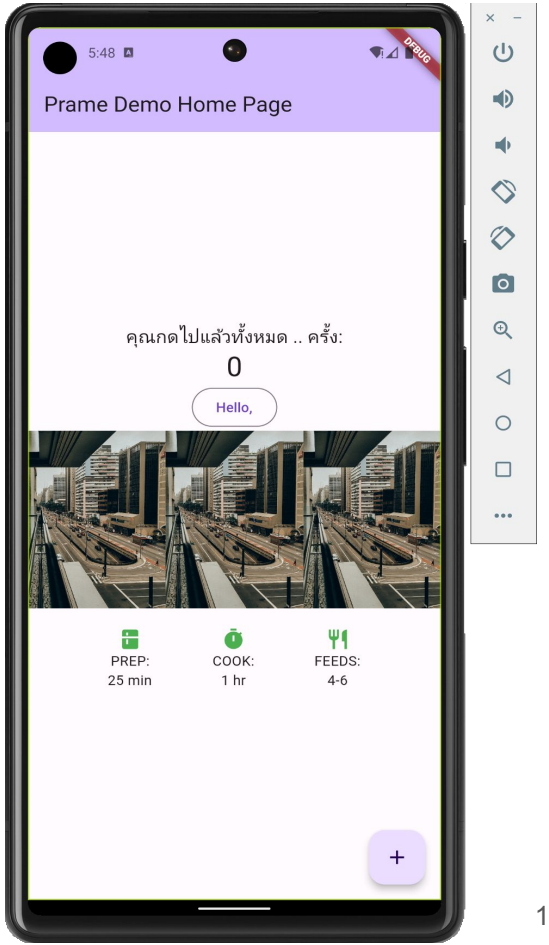


The `ratings` variable creates a row containing a smaller row of 5 star icons, and text:

```
var stars = Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    Icon(Icons.star, color: Colors.green[500]),
    const Icon(Icons.star, color: Colors.black),
    const Icon(Icons.star, color: Colors.black),
  ],
);

final ratings = Container(
  padding: const EdgeInsets.all(20),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      stars,
      const Text(
        '170 Reviews',
        style: TextStyle(
          color: Colors.black,
          fontWeight: FontWeight.w800,
          fontFamily: 'Roboto',
          letterSpacing: 0.5,
          fontSize: 20,
        ),
      ),
    ],
  ),
);
```

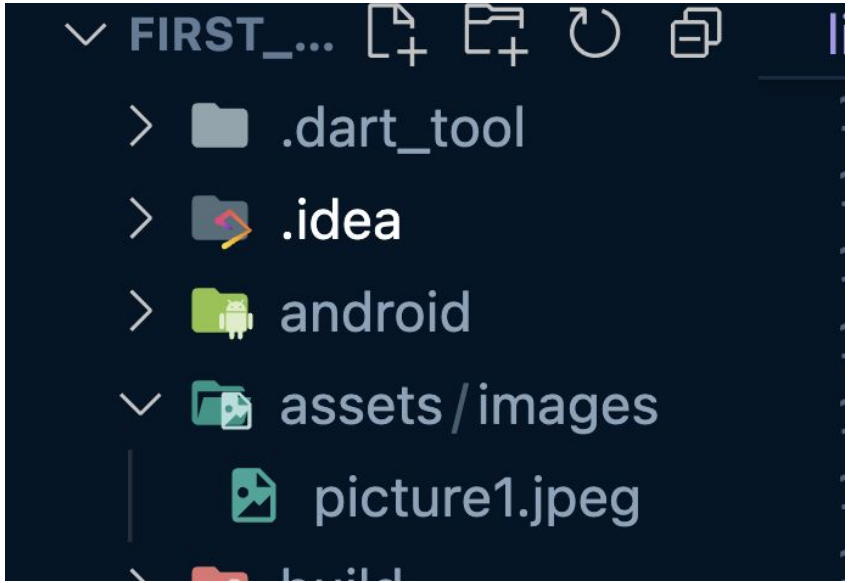
```
136          ), // Row
137          Container(
138            padding: const EdgeInsets.all(20),
139            child: Row(
140              mainAxisAlignment: MainAxisAlignment.spaceEvenly,
141              children: [
142                Column(
143                  children: [
144                    Icon(Icons.kitchen, color: Colors.green[500]),
145                    const Text('PREP:'),
146                    const Text('25 min'),
147                  ],
148                ), // Column
149                Column(
150                  children: [
151                    Icon(Icons.timer, color: Colors.green[500]),
152                    const Text('COOK:'),
153                    const Text('1 hr'),
154                  ],
155                ), // Column
156                Column(
157                  children: [
158                    Icon(Icons.restaurant, color: Colors.green[500]),
159                    const Text('FEEDS:'),
160                    const Text('4-6'),
161                  ],
162                ), // Column
163              ],
164            ), // Row
165          ), // Container
```

```
136         ), // Row
137         Container(
138           padding: const EdgeInsets.all(20),
139           child: Row(
140             mainAxisAlignment: MainAxisAlignment.spaceEvenly,
141             children: [
142               Column(
143                 children: [
144                   Icon(Icons.kitchen, color: Colors.green[500]),
145                   const Text('PREP:'),
146                   const Text('25 min'),
147                 ],
148               ), // Column
149               Column(
150                 children: [
151                   Icon(Icons.timer, color: Colors.green[500]),
152                   const Text('COOK:'),
153                   const Text('1 hr'),
154                 ],
155               ), // Column
156               Column(
157                 children: [
158                   Icon(Icons.restaurant, color: Colors.green[500]),
159                   const Text('FEEDS:'),
160                   const Text('4-6'),
161                 ],
162               ), // Column
163             ],
164           ), // Row
165         ), // Container
```

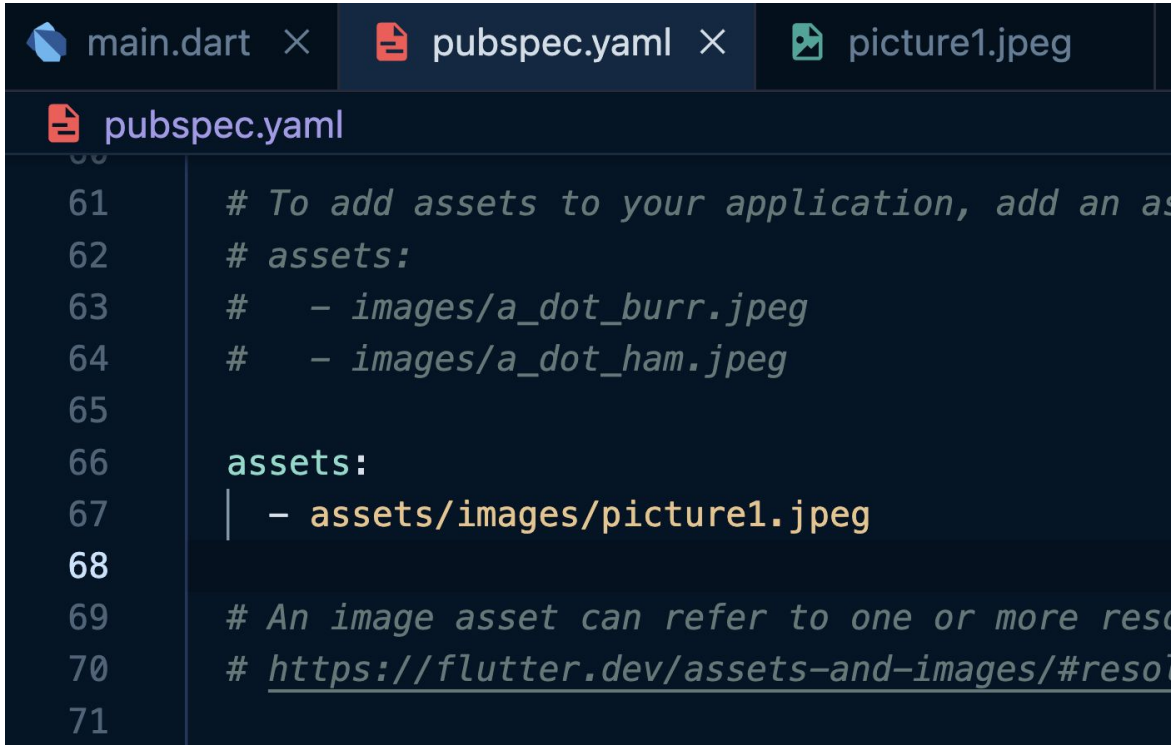


วิธีการใส่รูปภาพใน Flutter App

สร้าง Folder ใน Project ชื่อ assets/images แล้วใส่ไฟล์ภาพลงไป

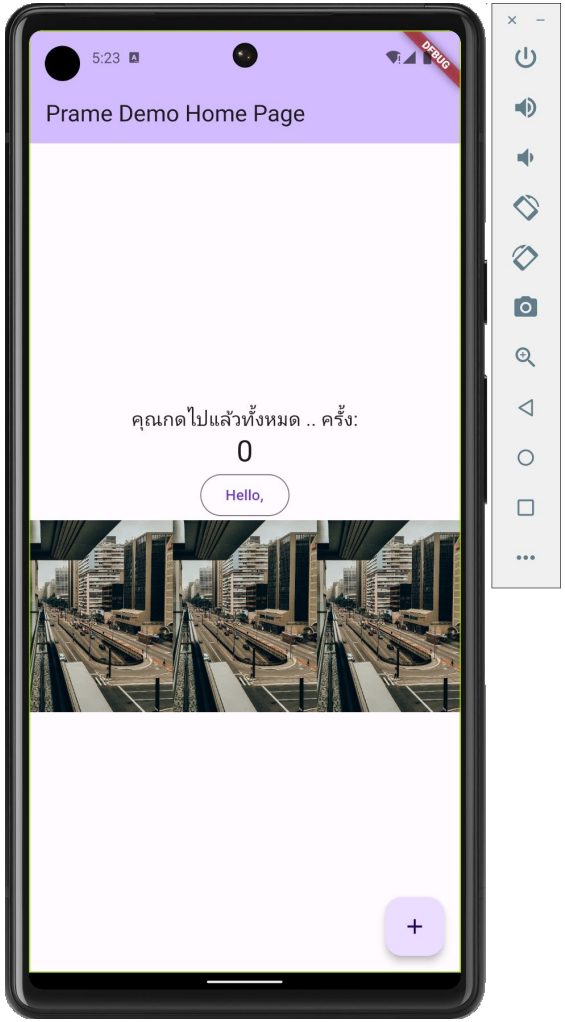


เพิ่มที่อยู่ของภาพในไฟล์ pubspec.yaml ลงไป



ชี้ตำแหน่งให้ถูกต้อง และ ทำการ reload

```
123        Row(
124          crossAxisAlignment: CrossAxisAlignment.center,
125          children: [
126            Expanded(
127              child: Image.asset('assets/images/picture1.jpeg'),
128            ), // Expanded
129            Expanded(
130              child: Image.asset('assets/images/picture1.jpeg'),
131            ), // Expanded
132            Expanded(
133              child: Image.asset('assets/images/picture1.jpeg'),
134            ), // Expanded
```


5:23
DEBUG
Prame Demo Home Page
คุณกดไปแล้วทั้งหมด .. ครั้ง:
0
Hello,

GridView

Use `GridView` to lay widgets out as a two-dimensional list. `GridView` provides two pre-fabricated lists, or you can build your own custom grid. When a `GridView` detects that its contents are too long to fit the render box, it automatically scrolls.

Summary (GridView)

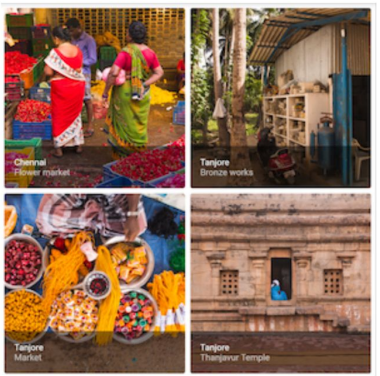
- Lays widgets out in a grid
- Detects when the column content exceeds the render box and automatically provides scrolling
- Build your own custom grid, or use one of the provided grids:
    - `GridView.count` allows you to specify the number of columns
    - `GridView.extent` allows you to specify the maximum pixel width of a tile

Note: When displaying a two-dimensional list where it's important which row and column a cell occupies (for example, it's the entry in the "calorie" column for the "avocado" row), use `Table` or `DataTable`.

GridView

Examples (GridView)

Uses `GridView.extent` to create a grid with tiles a maximum 150 pixels wide.



Uses `GridView.count` to create a grid that's 2 tiles wide in portrait mode, and 3 tiles wide in landscape mode. The titles are created by setting the `footer` property for each `GridTile`.

Dart code: grid_list_demo.dart from the Flutter Gallery

GridView

App source: grid_and_list

```
Widget _buildGrid() => GridView.extent(
    maxCrossAxisExtent: 150,
    padding: const EdgeInsets.all(4),
    mainAxisSpacing: 4,
    crossAxisSpacing: 4,
    children: _buildGridTileList(30));

// The images are saved with names pic0.jpg, pic1.jpg...pic29.jpg.
// The List.generate() constructor allows an easy way to create
// a list when objects have a predictable naming pattern.
List<Container> _buildGridTileList(int count) => List.generate(
    count, (i) => Container(child: Image.asset('images/pic$i.jpg')));
```