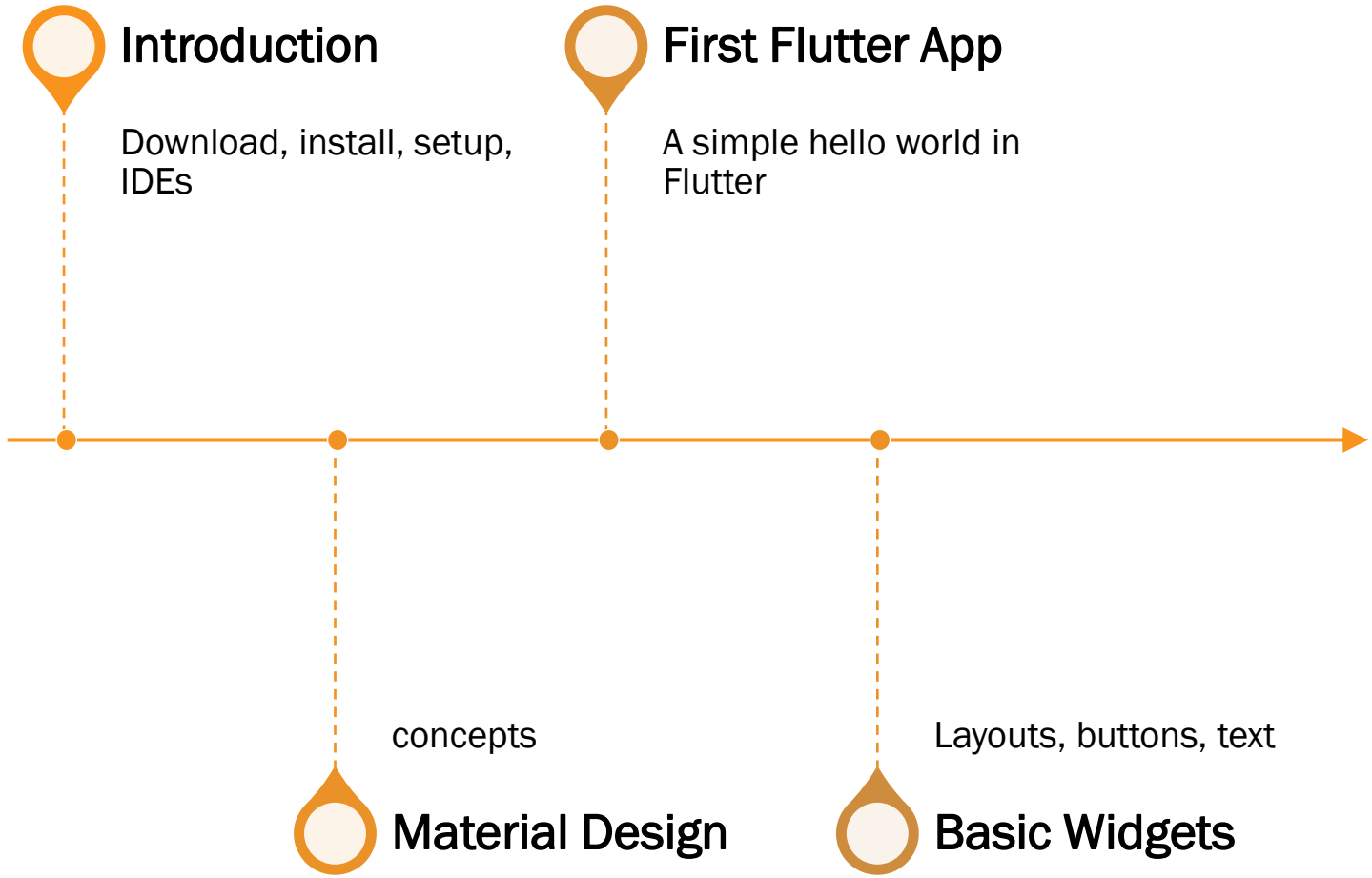
Flutter Framework

COURSE 5 (REV 5)

GAVRILUT DRAGOS

Agenda


Introduction
Download, install, setup, IDEs
First Flutter App
A simple hello world in Flutter
concepts
Material Design
Layouts, buttons, text
Basic Widgets

Introduction

Introduction

A framework build by Google design to create UX apps for:

- Mobile devices (Android and iOS)
- Desktops (Windows, Linux , MAC/OSX, ChromeOS)
- Web
- Embedded devices

Characteristics:

- Flutter designed was started in 2015 and it was available for usage in 2017
- Based on the paradigm: "**code once, deploy everywhere**"
- It is based on Dart language (meaning that usage of Flutter is easily done via Dart). However, the core engine of Flutter is written in C/C++ for high performance.
- Based on Material design concepts (https://material.io/)
- Current version: 3.24.0 (12.Feb.2025)

Introduction

Website: https://flutter.dev/ and https://flutter.dev/development

Download & Install:

- Windows development : https://docs.flutter.dev/get-started/install/windows
- Linux development: https://docs.flutter.dev/get-started/install/linux
- MAC/OSX development: https://docs.flutter.dev/get-started/install/macos

A separate setup is required for

- Android development
- Windows app development
- Web development

IDEs

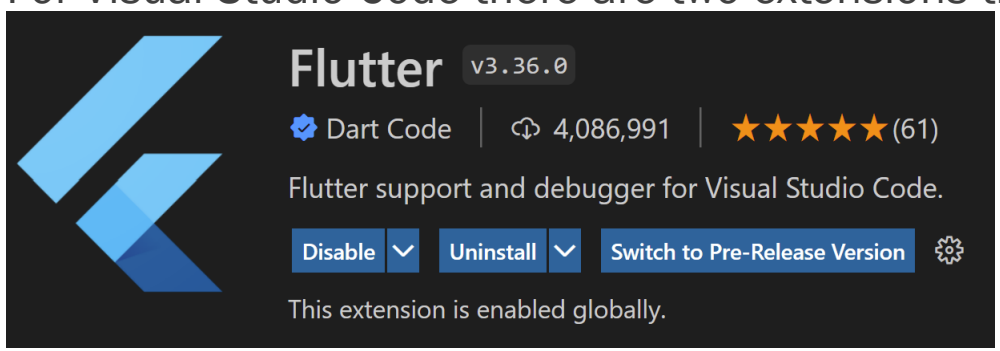
There are multiple IDEs that can be used for Flutter development:

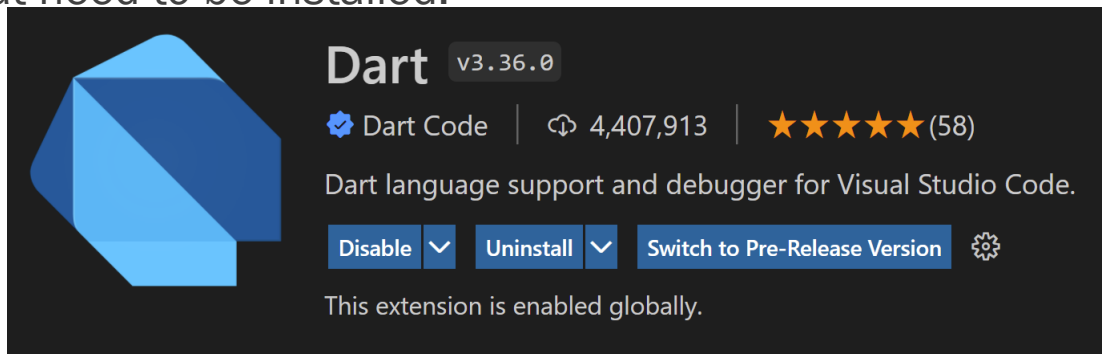
1. Visual Studio Code
2. Android Studio / IntelliJ
3. Emacs
4. DartPad (https://dartpad.dev)

For all these editors a plugin(s) will be required to allow integration with Flutter framework (the only exception to this rule being DartPad).

Visual Studio Code

For Visual Studio Code there are two extensions that need to be installed:





With this install you will have access to some new commands (accessible via Ctrl+Shift+P (Command Palette) from Visual Studio Code such as:

1. Flutter: New project
2. Flutter: Launch emulator
3. Flutter: Upgrade packages
4. ....

Flutter CLI

Once flutter is installed, you should see the following files in the <Flutter_installed_folder>/bin

```
cache
internal
mingit
dart
dart.bat
flutter
flutter.bat
```

If your using Windows OS for development use `flutter.bat` otherwise `flutter` (sh version).

It is usually best to make sure that this folder is located in the PATH environment variable and can be used directly from different locations.

Flutter CLI

Flutter CLI can be used to run several commands that are useful to automate build / checking of flutter-based applications:

Generic commands for Flutter SDK

- `channel` List or switch Flutter channels.
- `config` Configure Flutter settings.
- `doctor` Show information about the installed tooling.
- `upgrade` Upgrade your copy of Flutter.

Flutter CLI

Flutter CLI can be used to run several commands that are useful to automate build / checking of flutter-based applications:

Generic commands for Flutter projects

- `analyze` Analyze the project's Dart code.
- `assemble` Assemble and build Flutter resources.
- `build` Build an executable app or install bundle.
- `clean` Delete the build/ and .dart_tool/ directories.
- `create` Create a new Flutter project.
- `pub` Commands for managing Flutter packages.
- `run` Run your Flutter app on an attached device.

Flutter CLI

Flutter CLI can be used to run several commands that are useful to automate build / checking of flutter-based applications:

Generic commands for tools/devices

- `attach` Attach to a running app.
- `devices` List all connected devices.
- `emulators` List, launch and create emulators.
- `install` Install a Flutter app on an attached device.
- `logs` Show log output for running Flutter apps.
- `screenshot` Take a screenshot from a connected device.

Flutter CLI

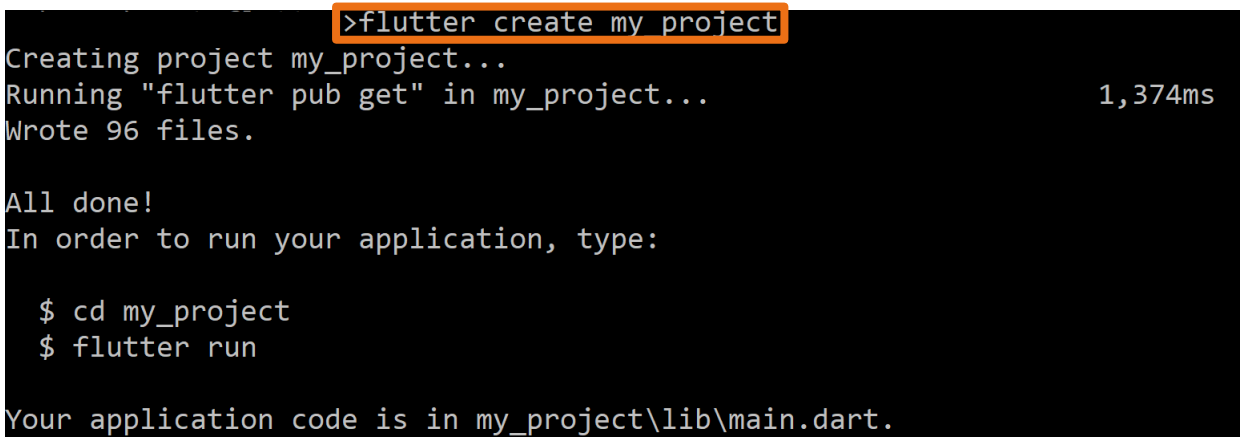
Flutter doctor: flutter.bat doctor is a very useful command to check the status of flutter framework.

```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.10.3, on Microsoft Windows [Version 10.0.22000.556], locale en-US)
[X] Android toolchain - develop for Android devices
    X Unable to locate Android SDK.
      Install Android Studio from: https://developer.android.com/studio/index.html
      On first launch it will assist you in installing the Android SDK components.
      (or visit https://flutter.dev/docs/get-started/install/windows#android-setup for detailed instructions).
      If the Android SDK has been installed to a custom location, please use
      `flutter config --android-sdk` to update to that location.

[√] Chrome - develop for the web
[√] Visual Studio - develop for Windows (Visual Studio Community 2022 17.2.0 Preview 2.1)
[!] Android Studio (not installed)
[√] Connected device (3 available)
[√] HTTP Host Availability

! Doctor found issues in 2 categories.
```

In this example, there are two problems: Android toolchain is not installed, and android studio is not installed. However, since Visual Studio is present, building a Windows app is possible. Make sure that you have the latest toolchain (latest update of Visual Studio).

Flutter CLI

To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output.



Flutter CLI

To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output. You should see a new folder “my_project” that has the following content:

```
.dart_tool
.idea
android
ios
lib
test
web
windows
.gitignore
.metadata
.packages
analysis_options.yaml
my_project.iml
pubspec.lock
pubspec.yaml
README.md
```

Flutter CLI

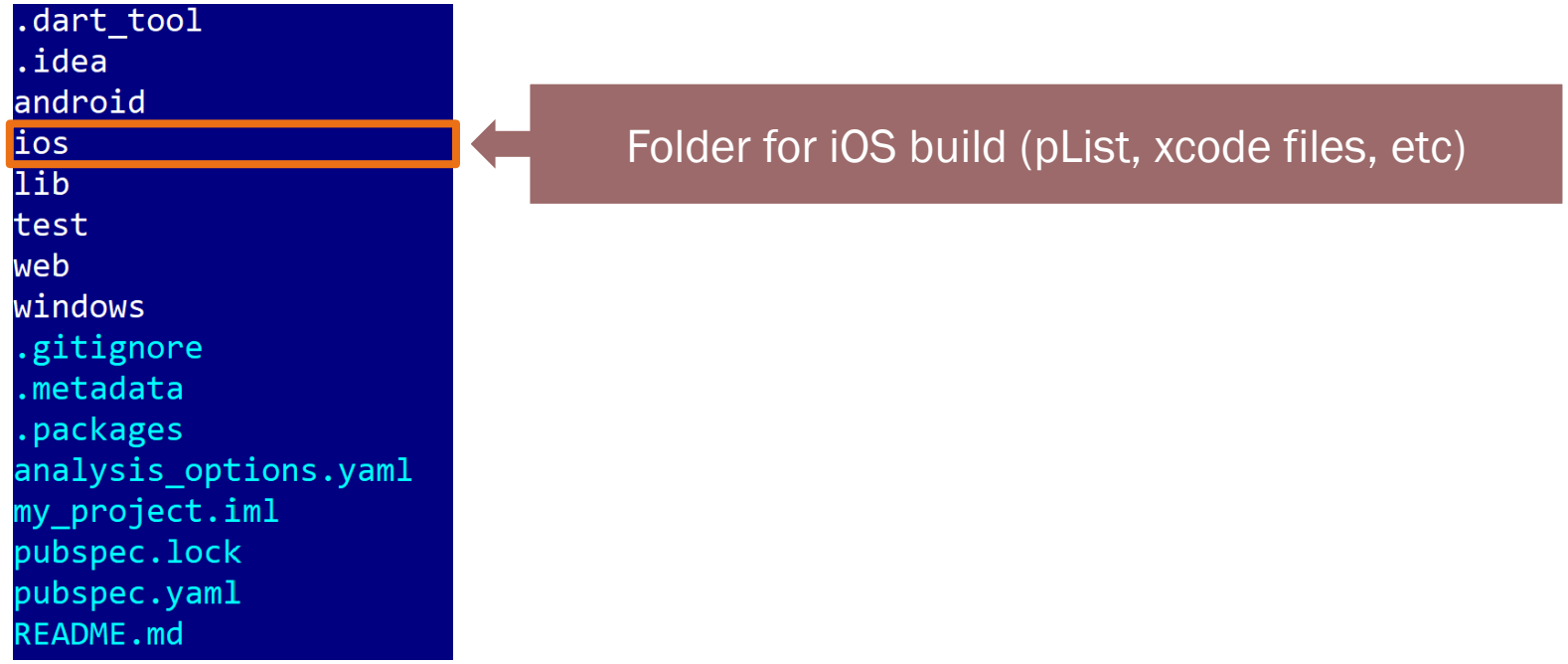
To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output. You should see a new folder “my_project” that has the following content:

```
.dart_tool
.idea
android
ios
lib
test
web
windows
.gitignore
.metadata
.packages
analysis_options.yaml
my_project.iml
pubspec.lock
pubspec.yaml
README.md
```

Folder for Android build (manifest, resource, class wrappers, etc)

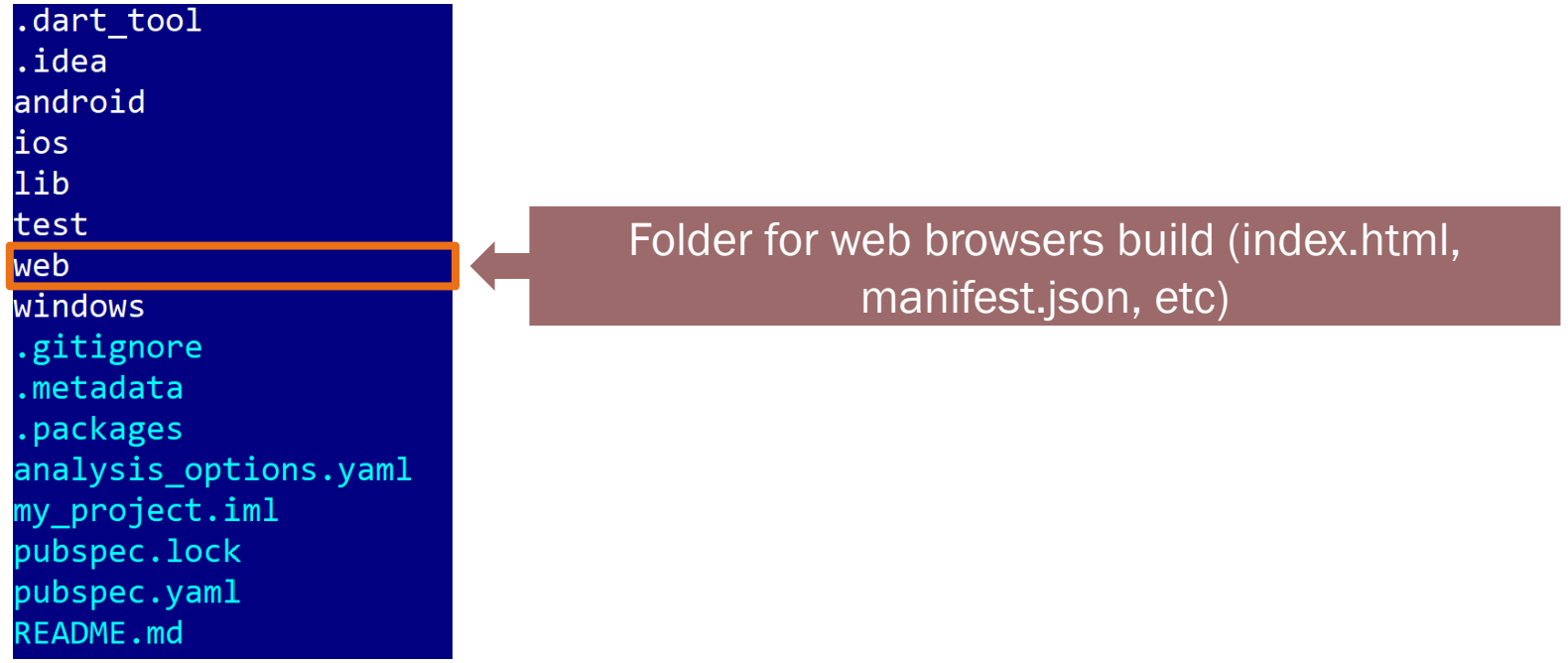
Flutter CLI

To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output. You should see a new folder “my_project” that has the following content:



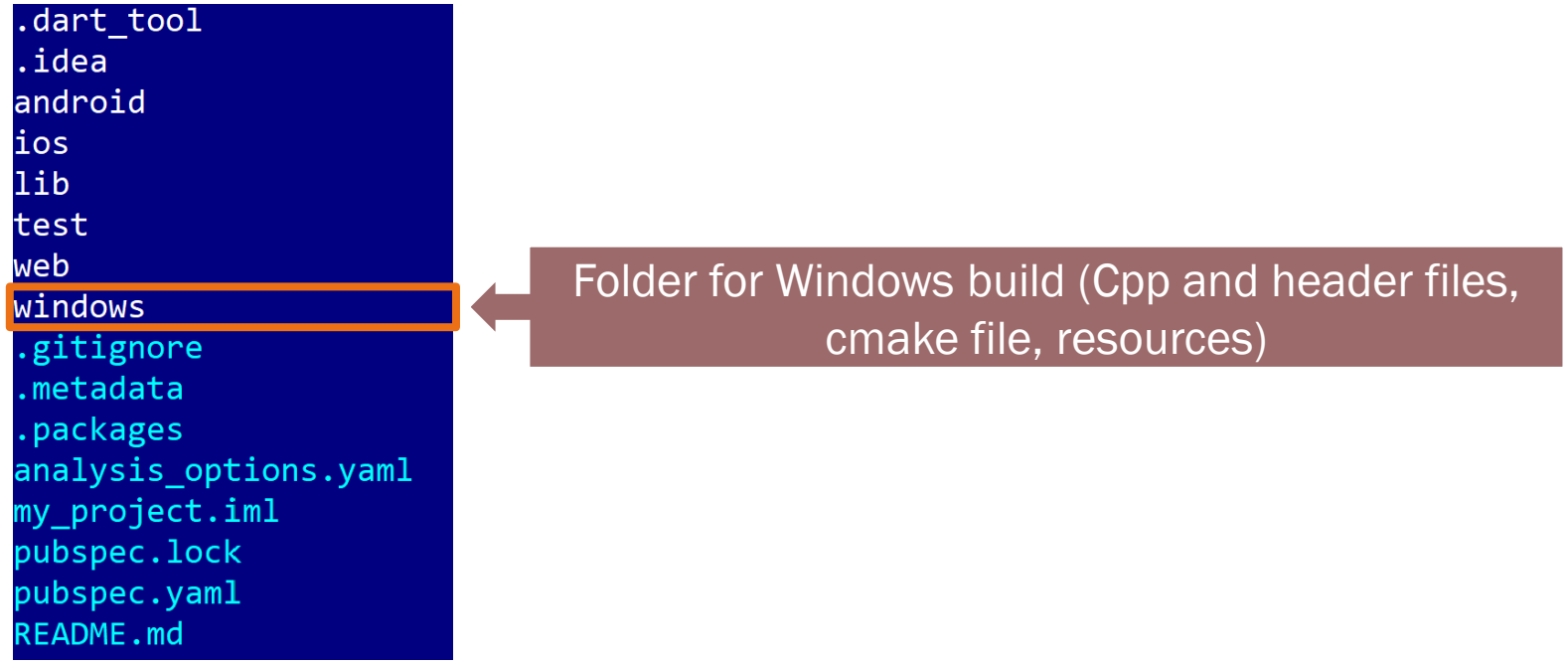
Flutter CLI

To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output. You should see a new folder “my_project” that has the following content:



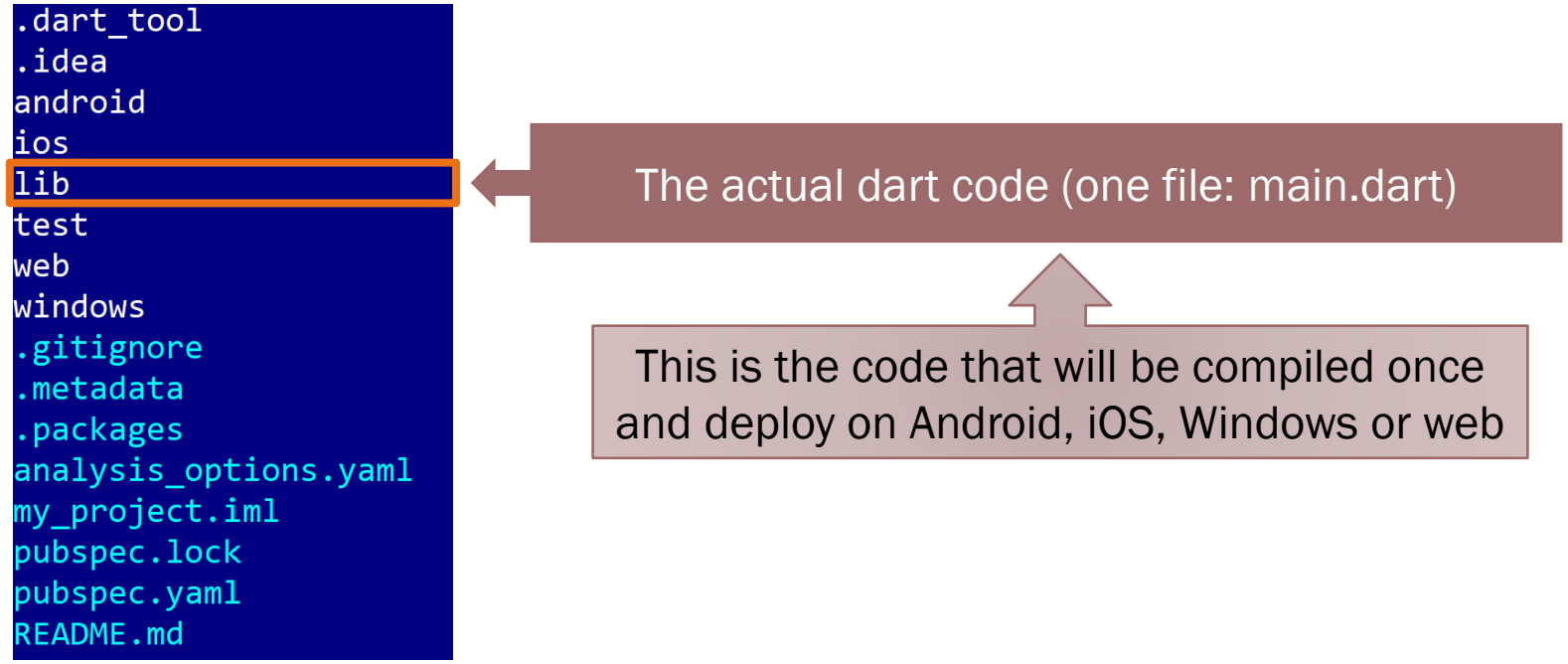
Flutter CLI

To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output. You should see a new folder “my_project” that has the following content:



Flutter CLI

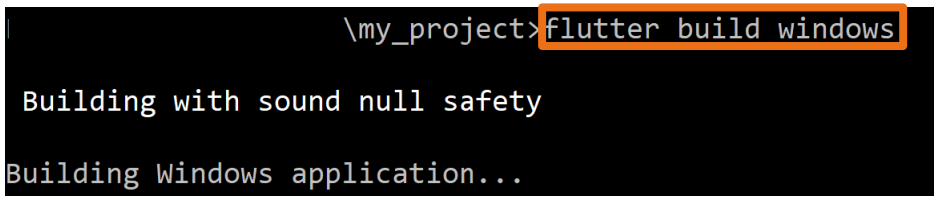
To create a project, you can either use an option from the IDE or type “flutter.bat create <name>” from command line. For example: running “flutter create my_project” will should create the following output. You should see a new folder “my_project” that has the following content:



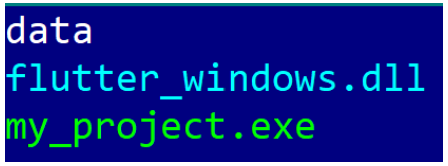
Flutter CLI

To build a flutter-based application, one can use an IDE command or just enter the project folder and run “flutter.bat build <system>” from command line, where <system> can be “**windows**”, “**android**”, etc.

For example: running “flutter build windows” from the folder “my_project” created on the previous step should result in the following:



And you should see the following files in my_project\build\windows\runner\Release



Flutter CLI

To run a flutter project you can:

1. Run a command from the IDE (e.g. F5 from Visual Studio Code)
2. Build and run the executable from <project folder>\build\windows\runner\Release (this is specific for Windows only → for other environments the location is different or you might need to install on a specific device first)
3. Type "flutter.bat run" in the project folder from the command line. If multiple options are available, you will be asked to select one. For example: if both windows and web based are possible, you should see something like this:

```
Multiple devices found:
Windows (desktop) • windows • windows-x64    • Microsoft Windows [Version 10.0.22000.556]
Chrome (web)      • chrome  • web-javascript • Google Chrome 99.0.4844.51
Edge (web)        • edge    • web-javascript • Microsoft Edge 97.0.1072.55
[1]: Windows (windows)
[2]: Chrome (chrome)
[3]: Edge (edge)
Please choose one (To quit, press "q/Q"): 1
Launching lib\main.dart on Windows in debug mode...
Building Windows application...
Syncing files to device Windows...                                 119ms
```

Material design

Material Design

A concept design and adopted by Google related to how an application (either for web or mobile) should be constructed (in terms of visibility, accessibility, usability, etc)

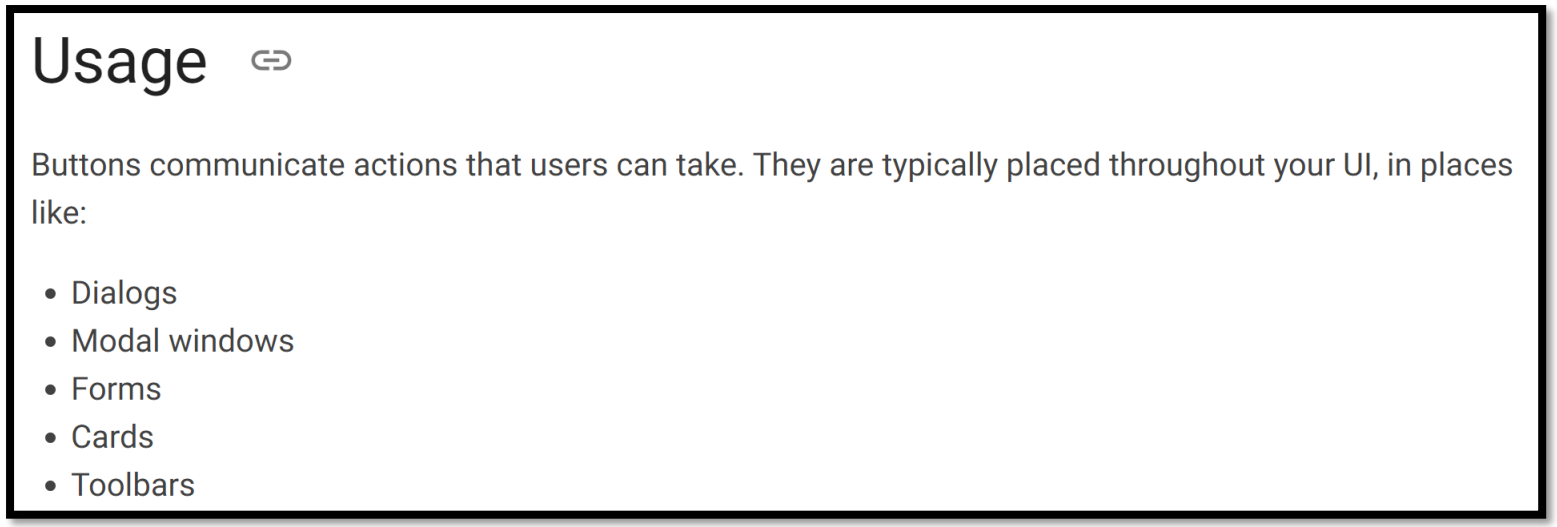
Characteristics:

- Material Design was announced in 2014
- Since then it has been adopted in all major Google platforms (such as gmail, google drive, google docs)
- Website (https://material.io/)
- Current version: 3
- It provides a sort of rules (“best practices”) on how this widgets should be used.
- It is available for:
  - Android
  - iOS
  - Web
  - Flutter

Material Design

For example, for a Button, you can read more on how-to-use on the following link:

https://material.io/components/buttons



Usage

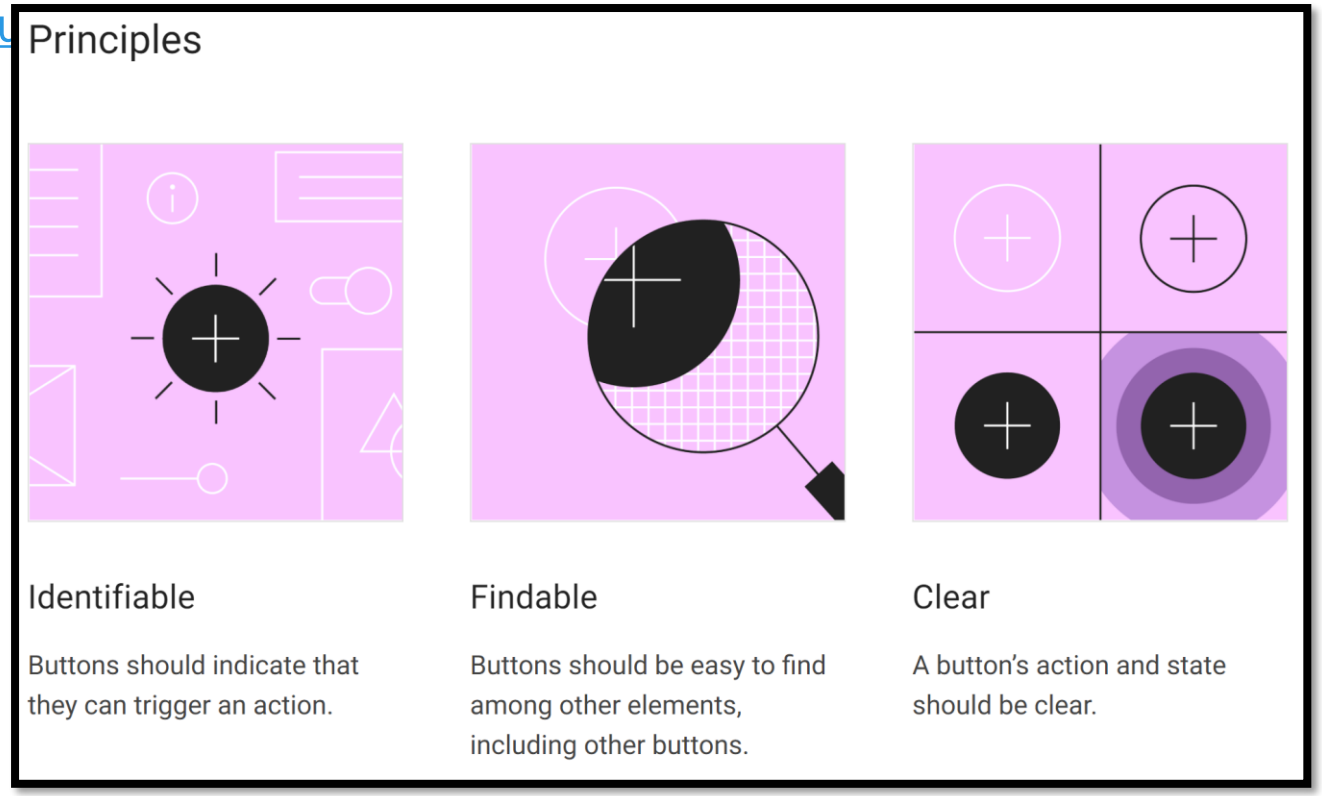
Buttons communicate actions that users can take. They are typically placed throughout your UI, in places like:

- Dialogs
- Modal windows
- Forms
- Cards
- Toolbars

Material Design

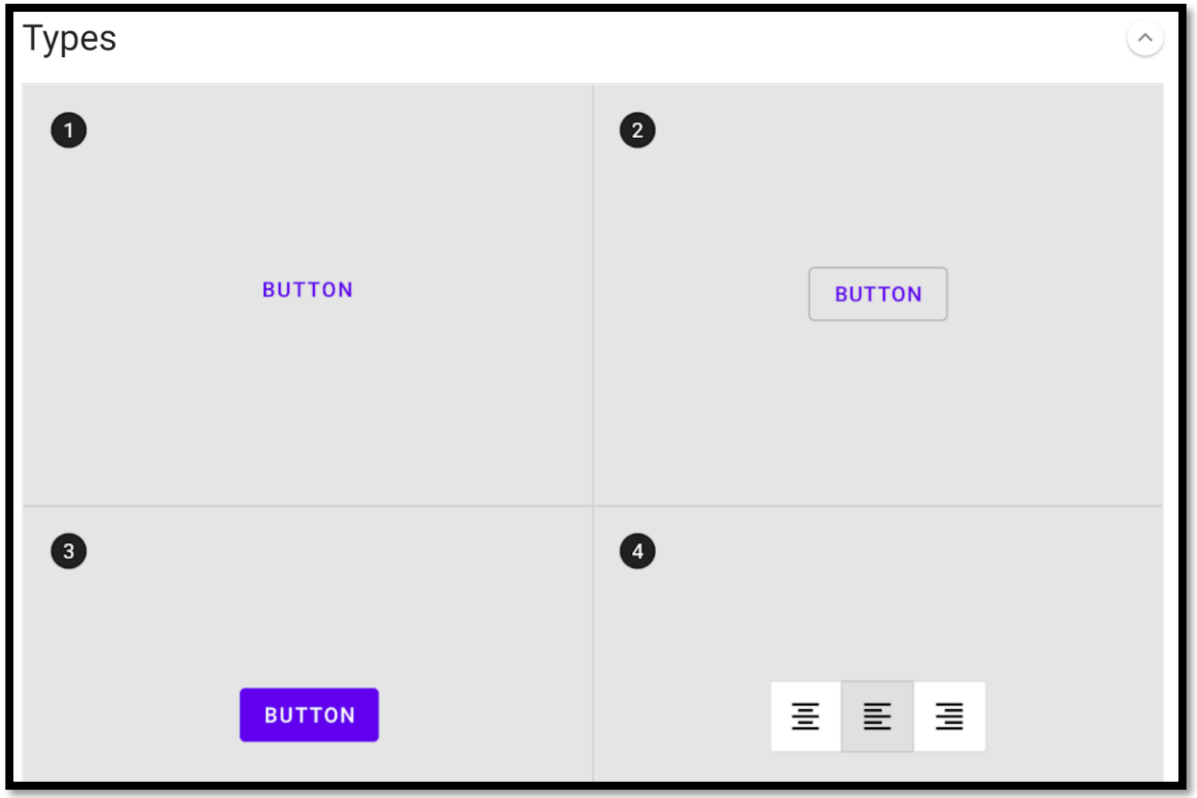
For example, for a Button, you can read more on how-to-use on the following link:

https://material.io/components/bu



Material Design

For example, for a Button, you can read more on how-to-use on the following link:



1. Text button (low emphasis)
Text buttons are typically used for less important actions.

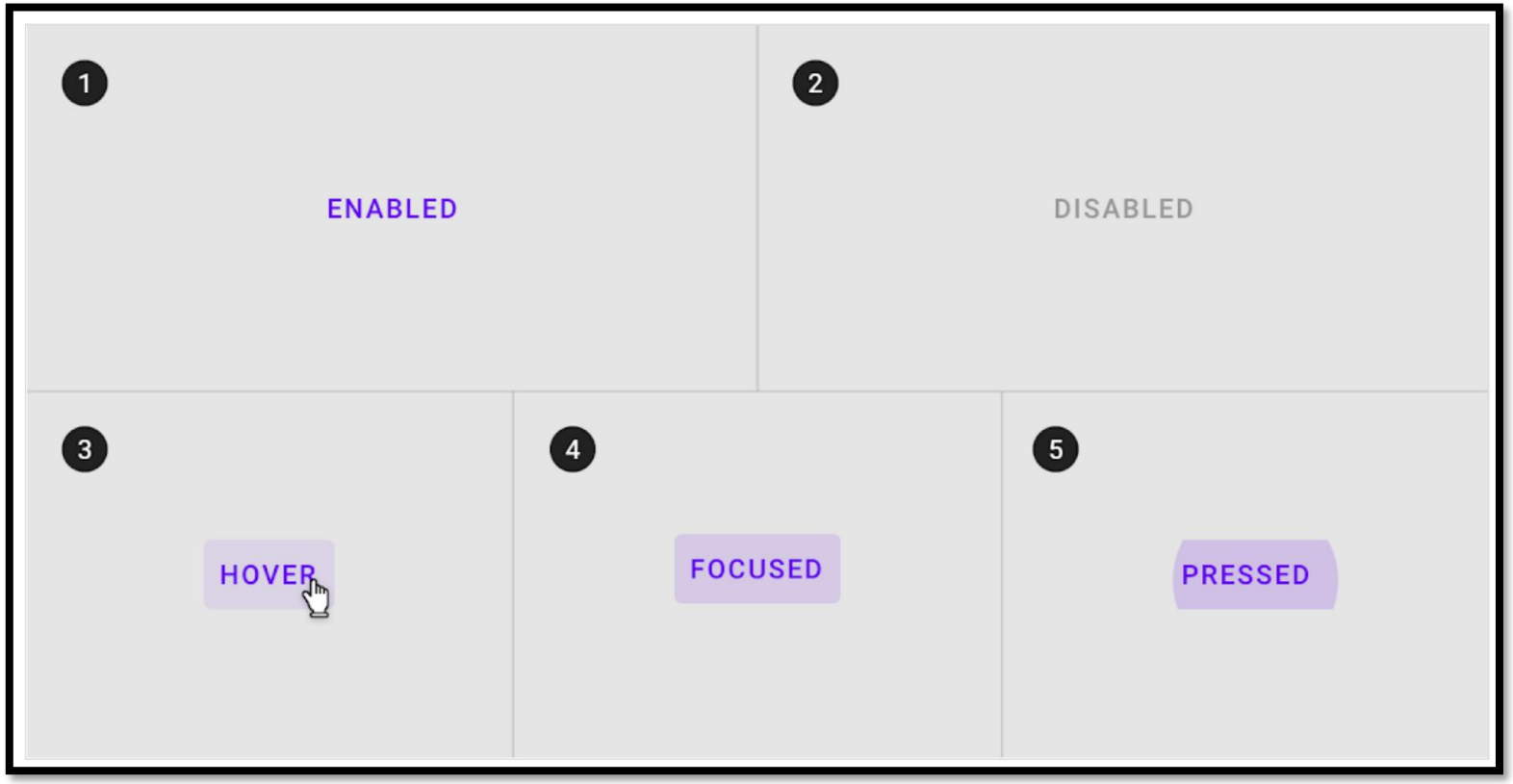
2. Outlined Button (medium emphasis)
Outlined buttons are used for more emphasis than text buttons due to the stroke.

3. Contained button (high emphasis)
Contained buttons have more emphasis, as they use a color fill and shadow.

4. Toggle button
Toggle buttons group a set of actions using layout and spacing. They're used less often than other button types.

Material Design

For example, for a Button, you can read more on how-to-use on the following link:



Material Design

For example, for a Button, you can read more on how-to-use on the following link:

First flutter App

First flutter App

Create a new flutter app and replace the code from main.dart with the following one:

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp();
  }
}
```

Run “flutter run” and try both windows and browser versions. In both cases you should see an empty window or an empty chrome tab.

First flutter App

Create a new flutter app and replace the code from main.dart with the following one:

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp();
  }
}
```

The main function usually consists in a runApp method that receives a Widget

Every UX control in Flutter is a Widget.

First flutter App

Create a new flutter app and replace the code from main.dart with the following one:

The most important method a **StatelessWidget** or a **StatefulWidget** is:

```
Widget build (BuildContext context)
```

This is the method that is used by flutter framework to create a tree of widgets that describe how elements are going to be draw on the screen.

First flutter App

Create a new flutter app and replace the code from main.dart with the following one:

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp();
  }
}
```

In our example, the method build returns a new Widget of type MaterialApp().

First flutter App

MaterialApp constructor looks has multiple parameters, the most important ones being:

```
MaterialApp (
  Widget? home,
  TransitionBuilder? builder,
  String title = '',
  Color? color,
  ThemeData? theme,
  ThemeMode? themeMode = ThemeMode.system,
  Locale? locale,
  bool debugShowMaterialGrid = false,
  bool showPerformanceOverlay = false,
  bool checkerboardRasterCacheImages = false,
  bool checkerboardOffscreenLayers = false,
  bool showSemanticsDebugger = false )
```

Out of these, the most important one is the home parameter that describes the top-level widget of the app.

This mean that at minimum, we need to add one widget to display something on our app.

First flutter App

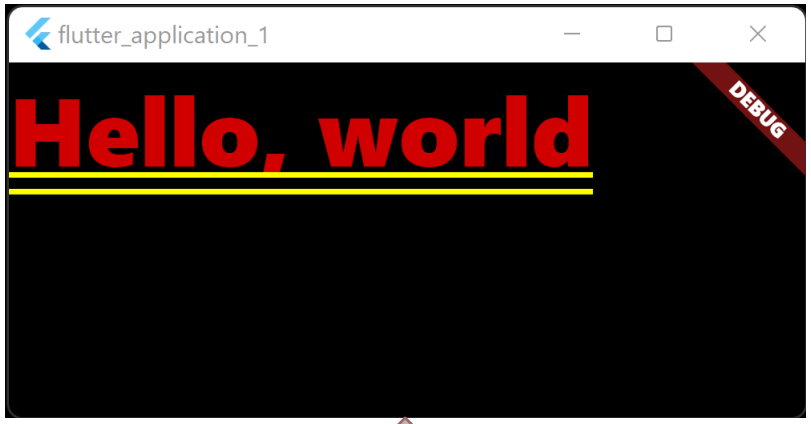
So … let's change the previous app code and add another widget (a very simple one):

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

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(home: const Text("Hello, world"));
  }
}
```

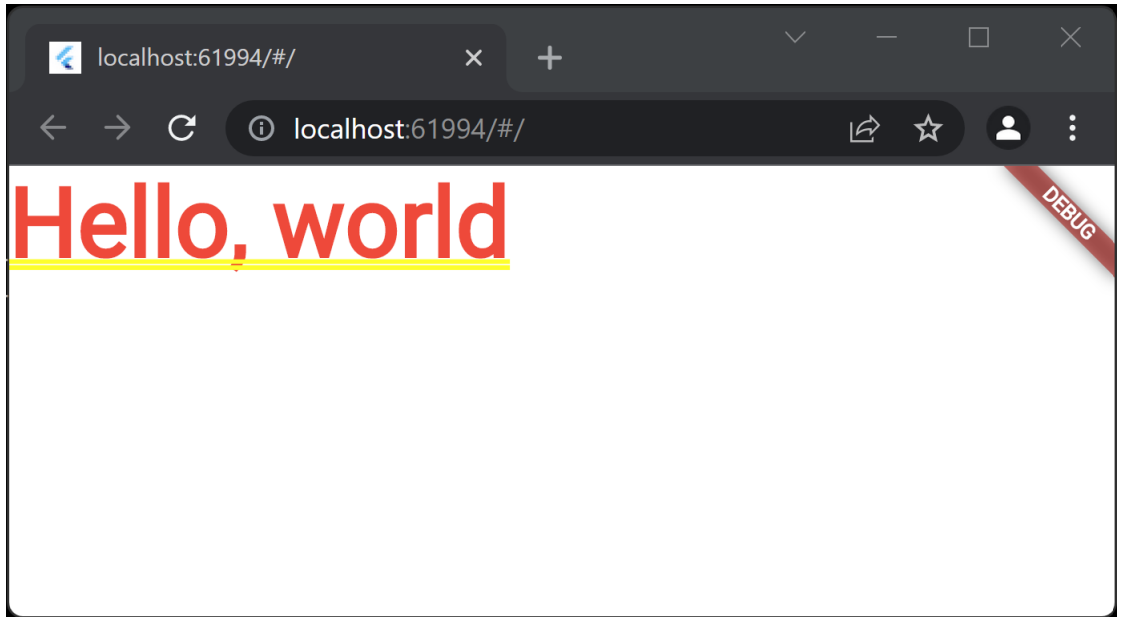
In this case we have used a Text widget to print the classical text "Hello, world"

First flutter App

When we execute the previous code via “flutter run” command we get:



or



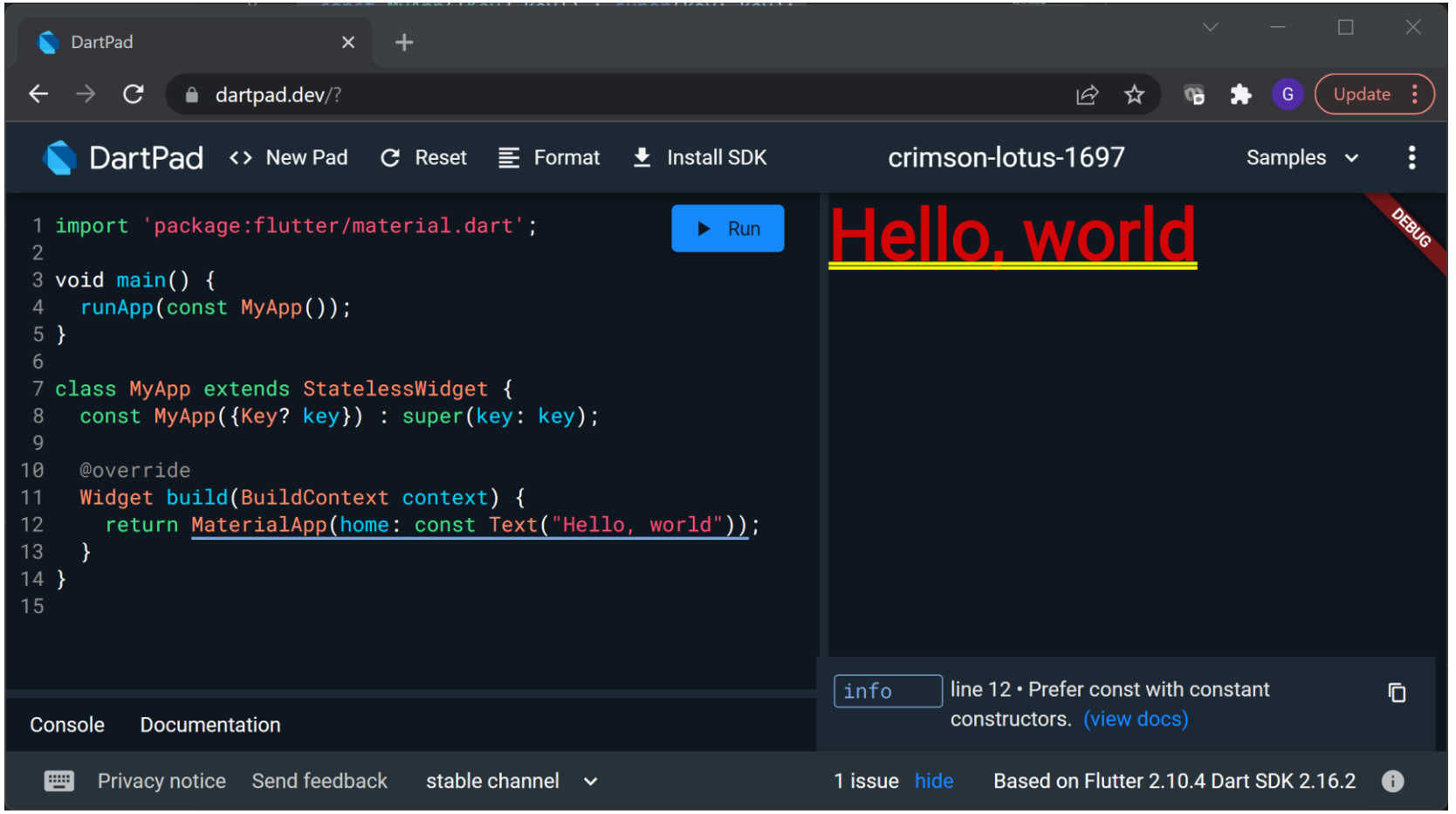
Window app

Chrome browser

The difference in this case is that the windows app uses system theme (Dark mode) while the browser just renders in white.

First flutter App

You can also test the same code in DartPad (if you don't have the flutter framework installed).



Basic widgets

Scaffold widget

A scaffold widget is a widget that allows creating a material view for your app that has:

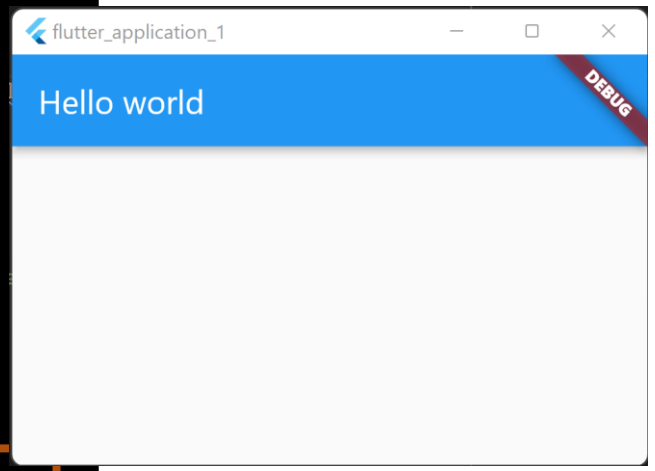
- An application bar
- A body

This is very similar to how Android apps or in general web base google apps look like:

Scaffold widget

Create a new flutter app and replace the code from main.dart with the following one:

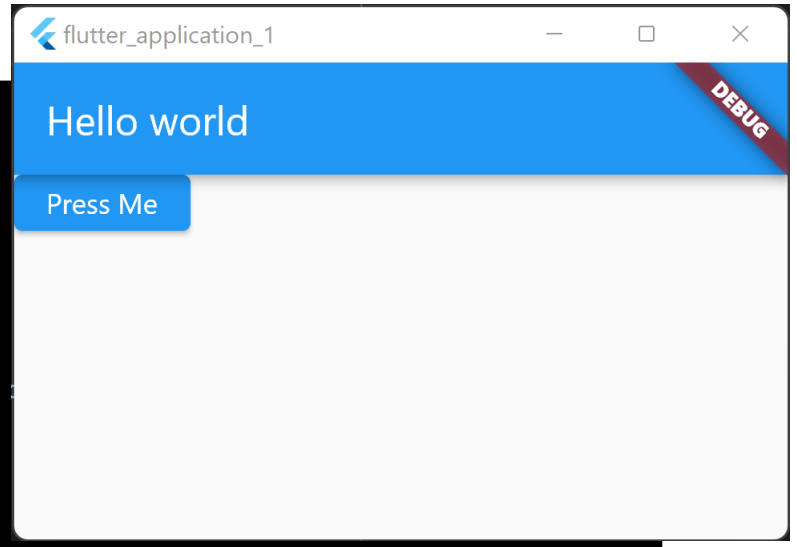
```dart
import 'package:flutter/material.dart';
void main() { runApp(const MyApp()); }

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
              appBar: AppBar(title: Text("Hello world")),
              )
        );
  }
}
```



ElevatedButton widget

Let's add a body with a button:

```
import 'package:flutter/material.dart';
void main() { runApp(const MyApp()); }
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  Widget GetBody() {
    return ElevatedButton( child: Text("Press Me"),
                           onPressed: () => {});
  }
  @override
  Widget build(BuildContext context) {
    return MaterialApp( home: Scaffold(
           appBar: AppBar(title: Text("Hello world")), body: GetBody()));
  }
}
```



ElevatedButton widget

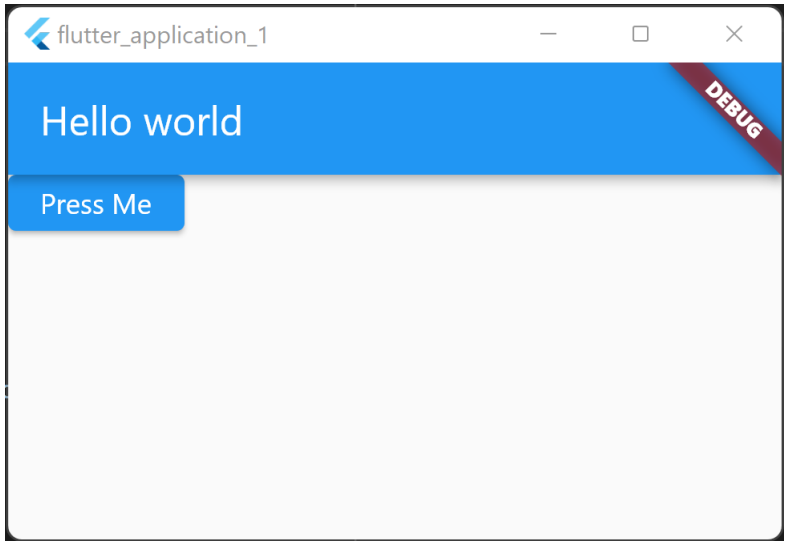
To simplify the code even more, let's consider the GetBody the method we are going to change from this moment on, and the rest of the code will remain the same.

```
import 'package:flutter/material.dart';
void main() { runApp(const MyApp()); }
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  Widget GetBody() { … }
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
           home: Scaffold(
                  appBar: AppBar(title: Text("Hello world")),
                  body: GetBody()));
  }
}
```

ElevatedButton widget

Let's add a body with a button:

```
Widget GetBody() {
  return ElevatedButton( child: Text("Press Me"),
                         onPressed: () => {});
}
```



One observation in this case is that the button is located in the top-left corner. To change this behavior we need to use various containers and layouts.

Layout widgets

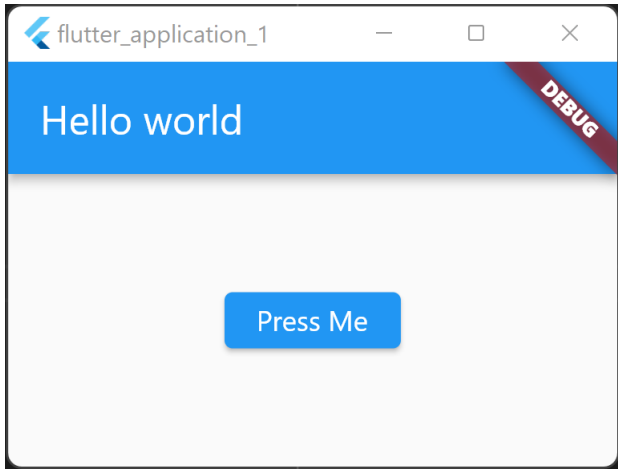
Flutter has multiple layouts – the most common one being:

1. Center → the entire layout is position to the center of the app
2. SizedBox → allows one to specify the size of a container
3. Padding → allows one to add a margin
4. Column → multiple elements are added in a vertical flow
5. Row → multiple widgets are added in a horizontal flow
6. Container → a generic container
7. Positioned → to select a specific location for another widget
8. FractionallySizedBox → Similar to SizedBox with the only difference that the width and height are provided as percentages from the parent

Center widget

A center widget has one child, and its child will be center to the parent of the center widget (in this case the body from the scaffold widget from the app.:

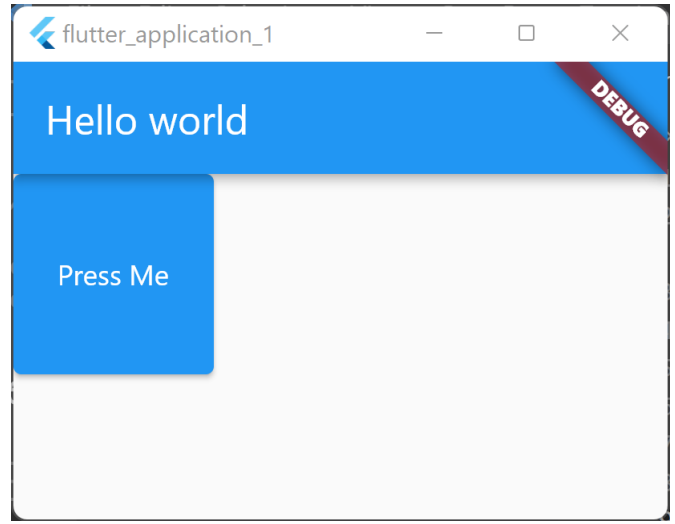
```
Widget GetBody() {
  return Center(
      child: ElevatedButton(
              child: Text("Press Me"),
              onPressed: () => {},
  ));
}
```



SizeBox widget

A SizeBox widget has one child, a width and a height property. In this example, width and height are set to 100.

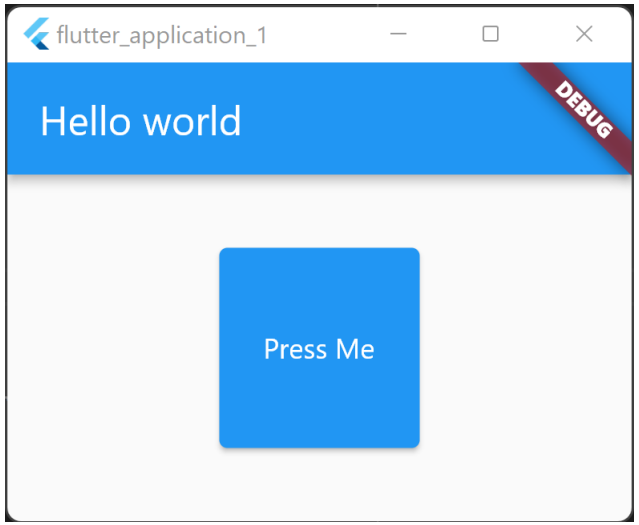
```
Widget GetBody() {
  return SizedBox(
          width: 100,
          height: 100,
          child: ElevatedButton(
                  child: Text("Press Me"),
                  onPressed: () => {},
  ));
}
```



SizeBox widget

All containers / layouts can be used together. This means that a SizedBox can be the child of a Center widths, or vice-versa.

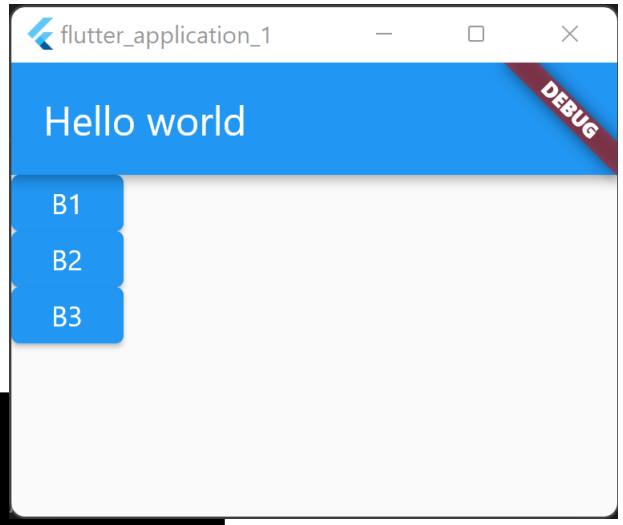
```
Widget GetBody() {
  return Center(
    child: SizedBox(
            width: 100,
            height: 100,
            child: ElevatedButton(
                  child: Text("Press Me"),
                  onPressed: () => {},
      )));
}
```



Now we have a 100x100 button centered.

Column widget

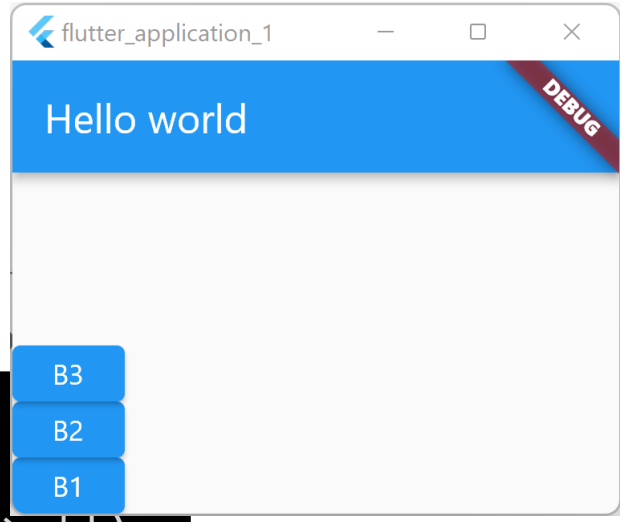
A Column or a Row widgets have multiple children. This means that their main property is children (that is a list of other widgets that are going to be added one after another (just like in a stack). In case of Column the stack will be vertical, while in case of Raw the stack will be displayed horizontal.




```
Widget GetBody() {
  return Column(children: [
    ElevatedButton(child: Text("B1"), onPressed: () => {}),
    ElevatedButton(child: Text("B2"), onPressed: () => {}),
    ElevatedButton(child: Text("B3"), onPressed: () => {})
  ]);
}
```

Column widget

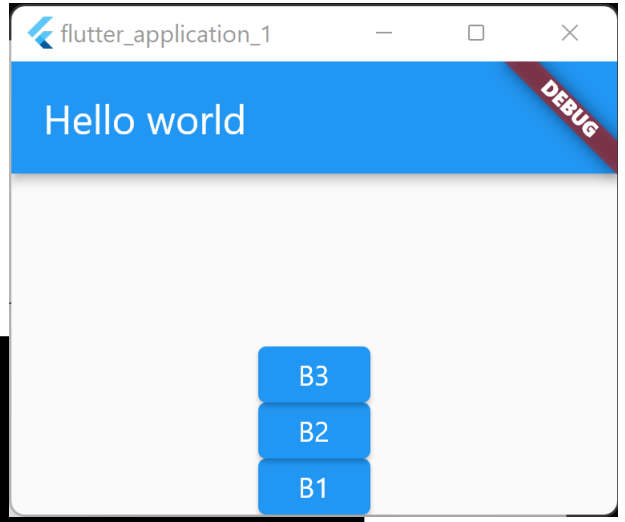
Another property of Column widget is `VerticalDirection` with two possible value (up or down). If set to `up` the children are going to be arrange from the bottom to top the reverse order (last one will be on the bottom, previous one will be on top of the last one, and so on ...)




```
Widget GetBody() {
  return Column(children: [
    ElevatedButton(child: Text("B1"), onPressed: () => {}),
    ElevatedButton(child: Text("B2"), onPressed: () => {}),
    ElevatedButton(child: Text("B3"), onPressed: () => {})
  ], verticalDirection: VerticalDirection.up);
}
```

Column widget

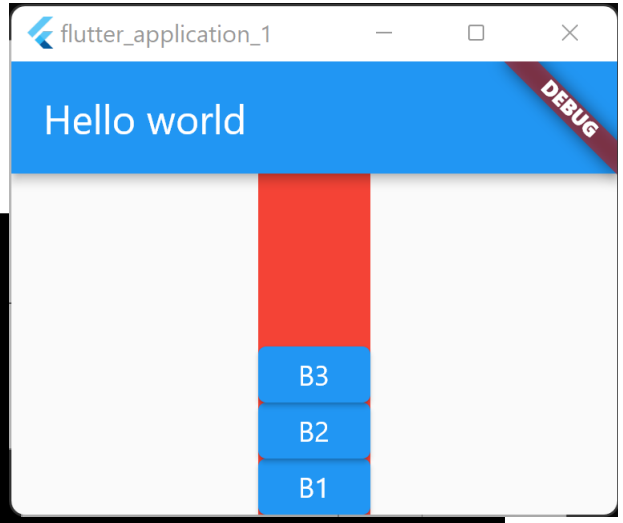
We can use a Center widget to center the column stack horizontally. Keep in mind that the column stack is as large as the height of its parent (in this case the body of the scaffold object).




```
Widget GetBody() {
  return Center(
      child: Column(children: [
          ElevatedButton(child: Text("B1"), onPressed: () => {}),
          ElevatedButton(child: Text("B2"), onPressed: () => {}),
          ElevatedButton(child: Text("B3"), onPressed: () => {})
        ], verticalDirection: VerticalDirection.up));
}
```

Column widget

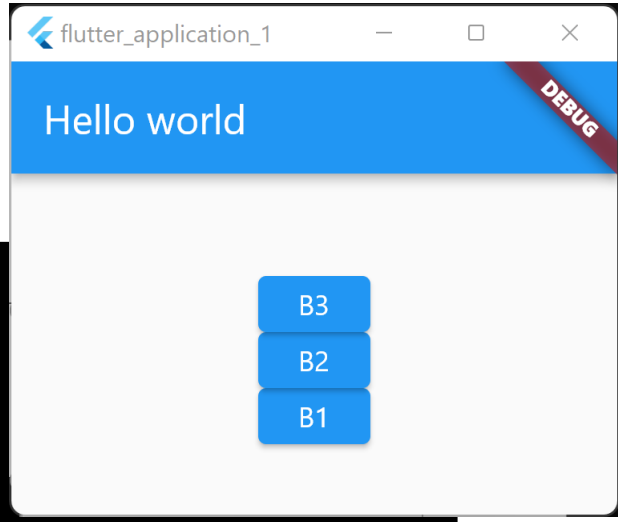
To validate the previous assumption, lets use a container widget with a background color of red set up.




```
Widget GetBody() {
  return Center(
      child: Container(
          color: Colors.red,
          child: Column(children: [
            ElevatedButton(child: Text("B1"), onPressed: () => {}),
            ElevatedButton(child: Text("B2"), onPressed: () => {}),
            ElevatedButton(child: Text("B3"), onPressed: () => {})
          ], verticalDirection: VerticalDirection.up)));
}
```

Column widget

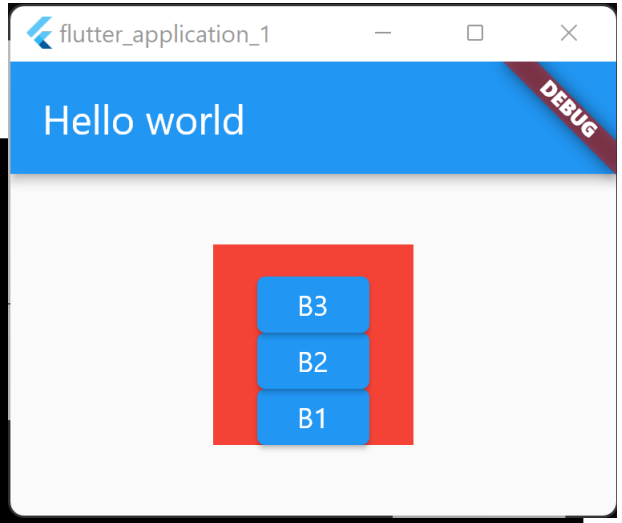
We can use a SizeBox between Center widget and ColumnWidget to limit the height to 100. This will somehow center all of the buttons (as long as their combine height is 100 ... or close to it !).




```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: Column(children: [
            ElevatedButton(child: Text("B1"), onPressed: () => {}),
            ElevatedButton(child: Text("B2"), onPressed: () => {}),
            ElevatedButton(child: Text("B3"), onPressed: () => {})
          ], verticalDirection: VerticalDirection.up)));
}
```

Column widget

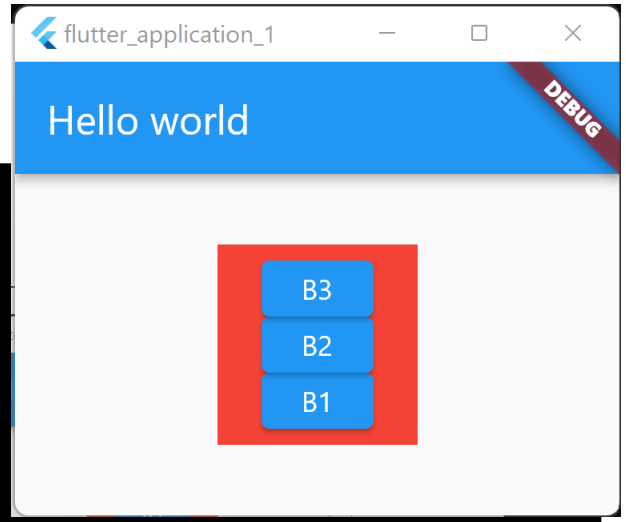
The reason we are formulating that it will not be perfect can be checked with a Container with color property set.




```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: Container(
              color: Colors.red,
              child: Column(children: [
                ElevatedButton(child: Text("B1"), onPressed: () => {}),
                ElevatedButton(child: Text("B2"), onPressed: () => {}),
                ElevatedButton(child: Text("B3"), onPressed: () => {})
              ], verticalDirection: VerticalDirection.up))));
}
```

Column widget

But, what if we want to make sure that the pile of buttons are perfectly centered → we can use the mainAxisAlign property.



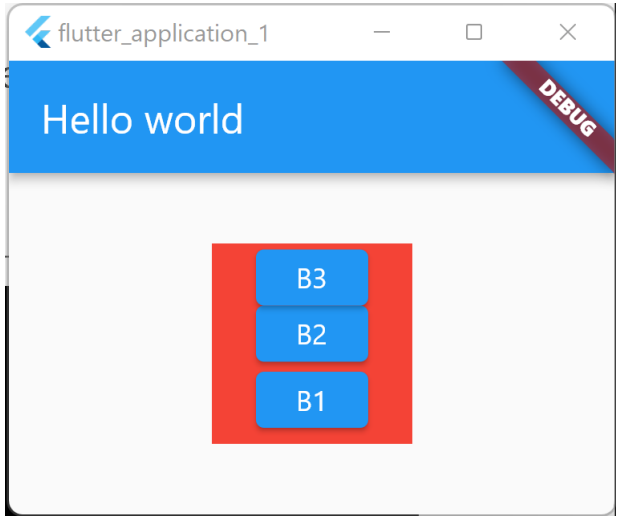

```
Widget GetBody() {
  return Center(
      child: SizedBox( width: 100, height: 100,
          child: Container( color: Colors.red,
              child: Column(
                children: [
                  ElevatedButton(child: Text("B1"), onPressed: () => {}),
                  ElevatedButton(child: Text("B2"), onPressed: () => {}),
                  ElevatedButton(child: Text("B3"), onPressed: () => {})
                ],
                verticalDirection: VerticalDirection.up,
                mainAxisAlignment: MainAxisAlignment.center))));
}
```

Padding widget

If we want to add some margins between widgets, we can use the Padding widget and select a specific margins (for left, right, bottom or top).

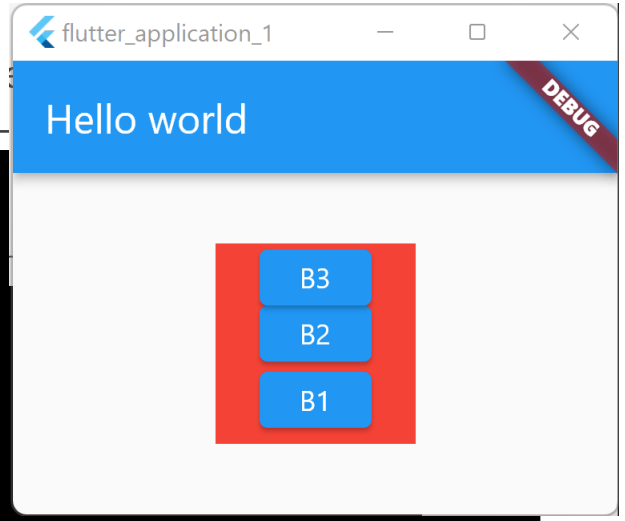
After we do this (the code will be displayed in the next slide) the outcome should be similar as the one from the next picture (you can see that button B1 has some margins between him and the rest of the bottoms).

To do this, use the property padding from the Padding widget.



Padding widget

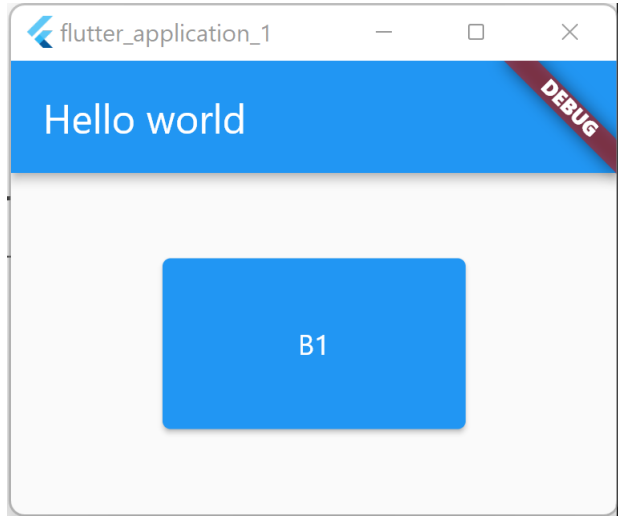
```
Widget GetBody() {
  return Center(
      child: SizedBox( width: 100,height: 100,
          child: Container( color: Colors.red,
              child: Column(
                children: [
                  Padding(
                      padding: EdgeInsets.fromLTRB(5, 5, 5, 5),
                      child: ElevatedButton(
                          child: Text("B1"), onPressed: () => {})),
                  ElevatedButton(child: Text("B2"), onPressed: () => {}),
                  ElevatedButton(child: Text("B3"), onPressed: () => {})
                ],
                verticalDirection: VerticalDirection.up,
                mainAxisAlignment: MainAxisAlignment.center,))));
}
```



FractionallySizedBox widget

Using FractionallySizedBox can provide a way to select the size of an object based on percentages from the width and height of its parent (in this case 50% or with and 50% of height)

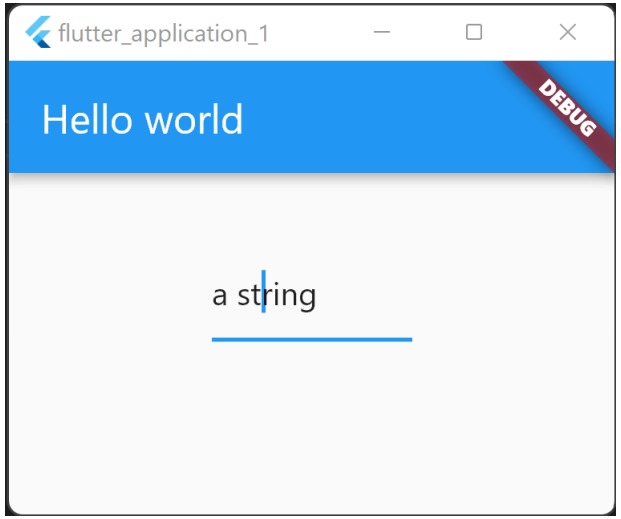
```
Widget GetBody() {
  return Center(
      child: FractionallySizedBox(
                  widthFactor: 0.5,
                  heightFactor: 0.5,
                  child: ElevatedButton(
                          child: Text("B1"),
                          onPressed: () => {}),
  ));
}
```



TextFormField widget

The TextFormField widget can be used to write / arrange or edit a text. It can also be used as a label if we set the read only property.

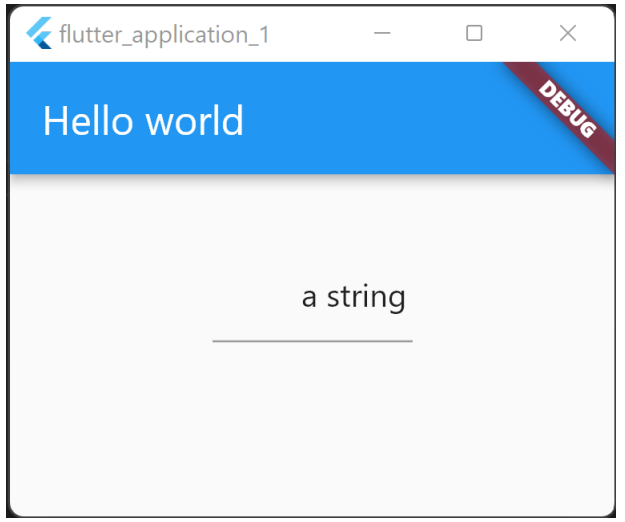
```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: TextFormField(
            initialValue: "a string",
          )));
}
```



TextFormField widget

We can also align the text (left or right) or make it read-only.

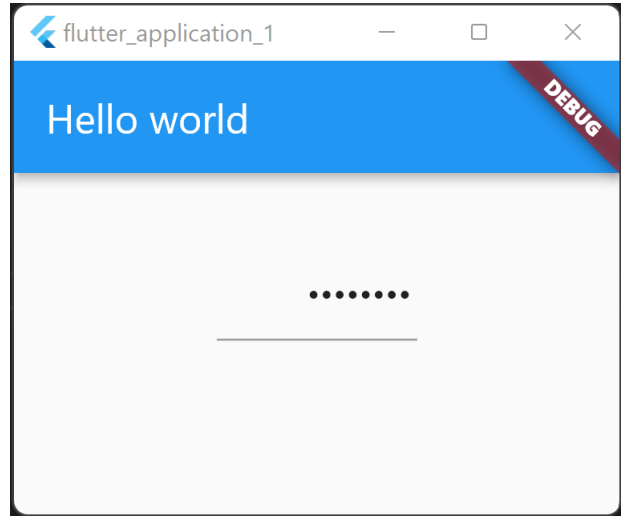
```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: TextFormField(
            initialValue: "a string",
            textAlign: TextAlign.end,
            readOnly: true,
          )));
}
```



TextFormField widget

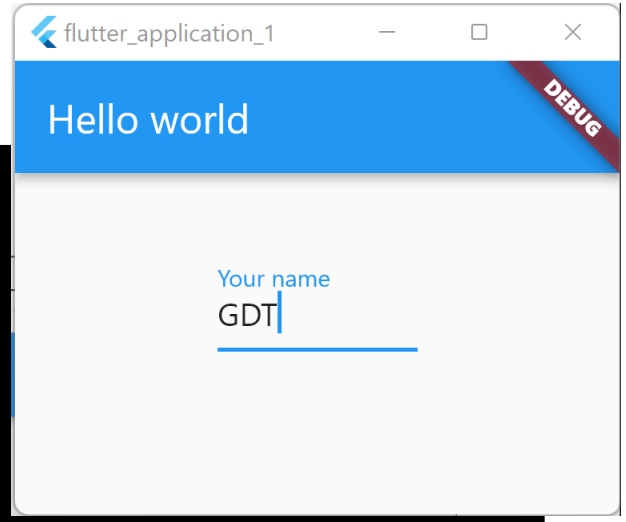
We can make it work like a password control (via obscureText parameter).

```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: TextFormField(
            initialValue: "a string",
            textAlign: TextAlign.end,
            obscureText: true,
          )));
}
```



TextFormField widget

You can also use an InputDecorator to add a label that explain what that text form is all about.

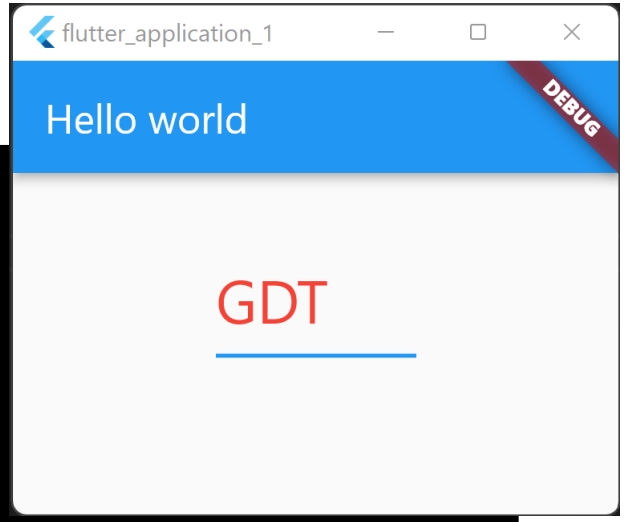



```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: TextFormField(
            initialValue: "GDT",
            decoration: const InputDecoration(labelText: "Your name"),
          )));
}
```

TextFormField widget

You can also change the font, size, color and a bunch of other font/text properties via style parameter.

```
Widget GetBody() {
  return Center(
      child: SizedBox(
          width: 100,
          height: 100,
          child: TextFormField(
            initialValue: "GDT",
            style: TextStyle(color: Colors.red, fontSize: 30),
          )));
}
```



Q & A